Technical Report

Nr. TUD-CS-2015-1268
November 24th, 2015

Time for Addressing Software Security Issues: Prediction Models and Impacting Factors

Authors
Lotfi ben Othmane, Fraunhofer SIT, Germany
Golriz Chehrazi, Fraunhofer SIT, Germany
Eric Bodden, Fraunhofer SIT, Germany
Petar Tsalovski, SAP SE, Germany
Achim D.~Brucker, SAP SE, Germany

Time for Addressing Software Security Issues: Prediction Models and Impacting Factors

Lotfi ben Othmane, Golriz Chehrazi, Eric Bodden, Petar Tsalovski, Achim D. Brucker

***Abstract*—Finding and fixing software vulnerabilities has become a major struggle for most software-development companies. While generally without alternative, such fixing efforts are a major cost factor, which is why companies have a vital interest in focusing their secure software development activities such that they obtain an optimal return on this investment.**

We investigate, in this paper, quantitatively the major factors that impact the time it takes to fix a given security issue based on data collected automatically within SAP's secure development process and we show how the issue fix time could be used to monitor the fixing process. We use three machine-learning methods and evaluate their predictive power in predicting the time to fix issues. Interestingly, the models indicate that the impact of vulnerability type has a small impact on issue fix time. The time it takes to fix an issue instead seems much more related to the component in which the potential vulnerability resides, the project related to the issue, the development groups that address the issue, and the closeness of the software release date. This indicates that the software structure, the fixing processes, and the development groups are the dominant factors that impact the time spent to address security issues.

SAP can use the models to implement a continuous improvement of its secure software development process and to measure the impact of individual improvements. Other companies can use similar models and mechanisms an be a learning organization.

***Index Terms*—Human factors, secure software, issue fix time.**

I. Introduction

FIXING vulnerabilities, before and after a release, is one of the most costly and unproductive software-engineering activities. Yet, it comes with few alternatives, as code-level vulnerabilities in the application code are the basis of increasingly many exploits [1]. Large software development enterprises, such as SAP, embed in their development process activities for identifying vulnerabilities early, such as dynamic and static security testing [2]. Next to that, SAP's security development lifecycle (see, e.g., [3] for Microsoft's security development lifecycle) includes a process for fixing vulnerabilities after a software release.

Analyzing and fixing security issues is a costly undertaking that impacts a software's time to market and increases its overall development and maintenance cost. In result, software development companies have an interest to determine the factors that impact the effort, and thus, the time it takes to fix security issues, in particular to:

- identify time-consuming factors in the secure development process,
- better understand affecting factors,
- focus on important factors to enhance software's security level,
- accelerate secure software development processes, and to
- enhance security-cost planning for software development projects.

In a previous study [4] we conducted expert interviews at SAP to identify factors that impact the effort of fixing vulnerabilities. SAP collects data about fixing security issues (potential vulnerabilities that need to be analyzed further manually to ensure whether they are vulnerabilities or false positive issues) both during a software's development and after its release. With this study we supplement the previous qualitative, interview-based results with objectively gathered system data. In this study, we used this data to identify and quantify, using machine learning, to what extent automatically measured factors impact a given issue's fix time. By *issue fix time* we mean the duration between the time at which a security issue is reported to SAP and the time at which the issue is marked as closed in number of days. For simplicity, we use the term issue to refer to a security issue in the remaining of the paper.

For the analysis we use five data sources based on distinct system tools available at SAP. The first three main data sources relate to security issues; issues found by code scanners for the programming language ABAP [5] (Data source 1) and for Java, JavaScript, and C (Data source 2), as well as issues found in already released code, which are communicated through so-called security messages, for instance reported by customers, security experts or SAP's own security team (Data source 3). The other two data sources comprise support data. They describe the components, i.e., a group of applications that performs a common business goal such as sales order or payroll (Support data 1), and the projects (Support data 2).

After cleaning the data, we used three methods to develop prediction models, based on (1) linear regression, (2) Recursive PARTitioning (RPART), and (3) Neural Network Regression (NNR). Next, we measured the models' accuracy using three different metrics. Interestingly, the models indicate that the impact of a vulnerability's type (buffer overflow, cross-side-scripting, etc.) has a less dominant impact than previously believed. Instead, the time it takes to fix an issue is more related to the component in which the vulnerability resides, the project related to the issue, to the development groups that address the issue, and to the closeness of the software release date.

SAP can use the results of this study to identify costly pain points and important areas in the secure development process, and to prioritize improvements to this process. Such models can be used to establish a learning organization, which learns

Lotfi ben Othmane, Golriz Chehrazi, and Eric Bodden are with Fraunhofer Institute for Secure Information Technology, Darmstadt, Germany.

Petar Tsalovski, Achim D. Brucker are with SAP SE, Walldorf, Germany.

Table I
EXAMPLES OF TIME REQUIRED FOR FIXING VULNERABILITIES [7].

Vulnerability type	Average fix time (min)
Dead Code (unused methods)	2.6
Lack of authorization check	6.9
Unsafe threading	8.5
XSS (stored)	9.6
SQL injection	97.5

and improves its processes based on the company-specific actual facts reflected in the collected data [6]. Since SAP collects the models' input data continuously, the models can be used to analyze the company's processes and measure the impact of enhancements over time.

This paper is organized as follows. First, we give an overview of related work (Section II), discuss SAP's approach to secure software development (Section III), and provide an overview of the regression methods and model accuracy metrics that we use in the study (Section IV). Next, we describe the research methodology that we applied (Section V), report about our findings (Section VI) and analyze the factors that impact the issue fix time (Section VII). Subsequently, we discuss the impacts and the limitations of the study (Section VIII), the main lessons (and surprises) that we learned (Section IX) and conclude the paper.

II. RELATED WORK

The are related work on prediction models for development efforts and time to fix bugs but work in the area of effort estimation for fixing security issues is scarce. Thus, we discuss in this section related work that investigate influencing factors on issue fix time or vulnerability fix time and also the development of prediction models for effort estimations, and differentiate them from our work.

Cornell measured the time that the developers spent fixing vulnerabilities in 14 applications [7]. Table I shows the average time the developers in the study take to fix vulnerabilities for several vulnerability types. Cornell found that there are vulnerability types that are easy to fix, such as dead code, vulnerability types that require applying prepared solutions, such as a lack of authorization, and vulnerability types that, although simple conceptually, may require a long time to fix for complex cases, such as SQL injection. The vulnerability type is thus one of the factors that indicates the vulnerability fix time but is certainly not the only one [4].

In previous work [4] we reported on a qualitative study conducted at SAP to identify the factors that impact the effort of fixing vulnerabilities and thus, the vulnerability fix time. The study involved interviews with 12 security experts. Through these interviews we identified 65 factors that include, beside the vulnerabilities characteristics, the structure of the software involved, the diversity of the used technologies, the smoothness of the communication and collaboration, the availability and quality of information and documentation, the expertise and knowledge of developers and security coordinators, and the quality of the code-analysis tools.

Several studies aim at predicting the time to fix bugs [8], [9], [10], [11], [12], [13], [14]. Zhang et al. [15] conducted an empirical study on three open-source software to examine what factors affect the time between bug assignment to a developer and the time bug fixing starts, that is the developer's delay (when fixing bugs), along three dimensions: bug reports, source code, and code changes. The most influencing factor found was the issue's level of severity. Other factors are of technical nature, such as sum of code churn, code complexity or number of methods in changed files as well as the maximum length of all comments in a bug report. Similar to our study, Zhang et al. were interested in revealing factors that impact time, but as opposed to them we focus on security issues, not on bugs, and include in our analysis not only automatically collected information about security issues before and after release, but additionally component- and project-related factors from which human-based and organizational factors can be derived. In contrast to Zhang et al., we consider the overall fix time that starts at the time when a security issue is reported and ends when the issue is marked as closed.

Menzies et al. [16] estimated projects development-effort, using project related data, such as the type of teams involved, the development time of the projects, and the number of high-level operations within the software. They found that it is better to use local data based on related projects instead of global data, which allows to account for project-related particularities that impact the development effort. Their data sample is a "global dataset" that includes data from several research software projects conducted by different entities. Instead, our study uses only data sets from one company, SAP.

In another study, Menzies et al. [17] reassured the usefulness of static code attributes to learn defect predictors. They showed that naive-Bayes machine-learning methods outperform rule-based or decision-tree learning methods and they showed, on the other hand, that the choice of learning methods used for defect predictions can be much more important than used attributes. Unlike this previous work, we use static code attributes to predict issue fix time and we use neural networks as additional method for prediction.

Following the objective to reduce effort for security inspection and testing, Shin et al. [18] used in their empirical study code complexity, code churn, and developer activity metrics obtained to predict vulnerable code locations with logistic regressions. They also used J48 decisions trees, random forest, and Bayesian network classification techniques based on data obtained from two large-scale open source projects using code characteristics and version control data. They found out that the combination of these metrics is effective in predicting vulnerable files. Nevertheless, they state that further effort is necessary to characterize differences between faults and vulnerabilities and to enhance prediction models. Unlike Shin et al., our empirical research focuses on predictions using system-based data to predict vulnerability fix time.

Hewett and Kijsanayothin [14] developed models for defect repair time prediction using seven different machine learning algorithms, e.g. decision trees and support vector machines. Their predictive models are based on a case study with data from a large medical software system. Similar to our approach they consider the whole repair time including all phases of a defect lifecycle. They use twelve defect attributes selected by domain experts for their estimations such as component, severity, start and end date, and phase. Unlike them we are interested in estimating vulnerability fix time not defect fix time.

In contrast to prior work, which often is based on open-source software, we estimate the vulnerability fix time based on an industrial case study of a major software development company, based on distinct data sets that include security issues before and after release and combine them with project and component-related data. Our objective is to identify the impacting strength of the factors on vulnerability fix time as well as to predict issue fix time in general.

III. Secure software development at SAP

To ensure a secure software development, SAP follows the SAP Security Development Lifecycle (S^{2}DL). Figure 1 illustrates the main steps in this process, which is split into four phases: preparation, development, transition, and utilization.

To allow the necessary flexibility to adapt this process to the various application types (ranging from small mobile apps to large-scale enterprise resource-planning solutions) developed by SAP as well as the different software-development styles and cultural differences in a worldwide distributed organisation, SAP follows a two-staged security-expert model:

1) a central security team defines the global security processes, such as the S^{2}DL, provides security trainings, risk identification methods, offers security testing tools, or defines and implements the security response process;
2) local security experts in each development area/team are supporting the developers, architects, and product owners in implementing the S^{2}DL and its supporting processes.

For this study, the *development* and *utilization* phases of the S^{2}DL are the most important ones, as the activities carried out during these phases detect most of the vulnerabilities that need to be fixed:

- during the actual software development (in the steps *secure development* and *security testing*) vulnerabilities are detected, e.g., by using manual and automated as well as static and dynamic methods for testing application security [19], [2]. Most vulnerabilities detected are found during this step, i.e., most vulnerabilities are fixed in unreleased code (e.g., in newly developed code that is not yet used by customers);
- *security validation* is an independent quality control that acts as "first customer" during the transition from software development to release, i.e., security validation finds vulnerabilities after the code freeze, (called correction close) and the actual release;
- *security response* handles issues reported after the release of the product, e.g., by external security researchers or customers.

If an issue is confirmed (e.g., by an analysis of a security expert), from a high-level perspective developers and their local security experts implement the following four steps: 1) analyze the issue, 2) design or select a recommended solution, 3) implement and test a fix, and 4) validate (e.g., by re-testing the fixed solution) and release this fix. Of course, the details differ depending of the development model of the product team and, more importantly, depending on whether the issue is detected in code that is used by customers or not.

While the technical steps for fixing an issue are the same regardless of whether the issue is in released code or currently developed code, the organizational aspects differ significantly: for vulnerabilities in unreleased development code, detecting, confirming, and fixing vulnerabilities is a lightweight process defined locally by the development teams. Vulnerabilities detected by security validation, e.g., after the code freeze, even if in unreleased code, involve much larger communication efforts across different organisations for explaining the actual vulnerabilities to development as well as ensuring that the vulnerability is fixed before the product is released to customers.

Fixing vulnerabilities in released code requires the involvement of yet more teams within SAP, as well as additional steps, e.g., for back-porting fixes to older releases and providing patches (called *SAP Security Notes*) to customers.

Let us have a closer look on how an externally reported vulnerability in a shipped software version is fixed: First, an external reporter (e.g., customer or independent security researcher) contacts the *security response team*, which assigns a case manager. The case manager is responsible for driving the decision if a reported problem is a security vulnerability that needs to be fixed, and for ensuring that the confirmed vulnerability is fixed and that a patch is released. After a vulnerability is confirmed, the case manager contacts the development team and often also a dedicated maintenance team (called IMS) to ensure that a fix is developed and back-ported to all necessary older releases (according to SAP's support and maintenance contracts). The developed fixes are subject to a special security test by the security validation team and, moreover, the response teams reviews the SAP Security Note. If the technical fix as well as the resulting Security Note pass the quality checks, the Security Note is made available to customers individually and/or in form of a support package (usually on the first Tuesday of a month). Support packages are functional updates that also contain the latest security notes.

IV. Background

Assume a response variable y and a set of independent variables x_i such that $y = f(x_1, x_2, \ldots, x_n)$ where f represents the systematic information that the variables x_i provide about y [20]. Statistical learning approaches can be seen as a means to infer f in such a way that its input/output-relation is consistent with those observed during learning [20]. In machine learning, regression models relate the *quantity* of a response factor, i.e., dependent variable, of a given object to other prediction factors, i.e., independent variables, of that same object.

Different regression models have different capabilities, e.g., in terms of their resistance to outliers, their fit for small datasets, and their fit for a large number of predicting factors [21]. However, in general, a regression model is assumed to be good, or useful, if it predicts responses close to the actual values observed in reality. In this study, a model's performance (i.e., indicated by the accuracy of the predictions) is judged

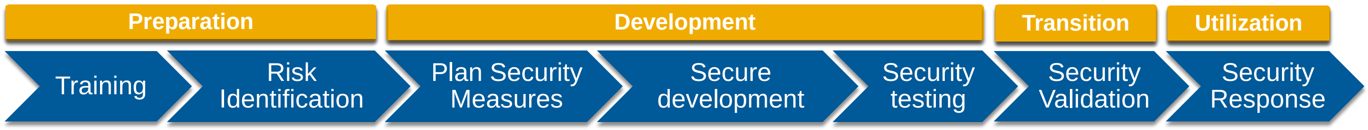


Figure 1. Overview of the SAP Security Development Lifecycle (S^{2}DL)

by determining its prediction errors, and the goal must be to minimize those errors.

This section provides background about the regression methods, possible performance metrics for generated-regression models, and a metric for measuring the relative importance of the prediction factors used in the models.

A. Overview of used regression methods

We give next an overview of the three methods used in this study.

Linear regression. This method assumes that the regression function is linear in the input [22], i.e., in the prediction factors. The linear method has the advantage of being simple and allows for an easy interpretation of the correlations between the input and output variables, i.e., of how the output of the function relates to the predicting variables.

Tree-based regression. This method recursively partitions the observations, i.e., the data records of the object being analyzed, for each of the prediction factors (aka features) such that it reduces the value of a metric that measures the information quantity of the splits [23]. In this study we use the method *recursive partitioning and regression trees (RPART)* [24], a commonly used tree-based method.

Neural-networks regression. This method represents functions that are non-linear in the prediction variables. It uses a multi-layer network that relates the input to the output through intermediate nodes. The output of each intermediate node is the sum of weighted input of the nodes of the previous layer. The data input is the first layer [25].

These three regression methods are the basic ones that are commonly used in data analytics. In this study, we use their implementations in packages for the statistics language R:[1] rpart[2] for RPART, and nnet[3] for NNR. The implementation “lm” of the Linear Regression (LR) is already contained within the core of R.

B. Model performance metrics

Regression methods infer prediction models from a given set of training data, such that prediction errors are minimized. Several metrics have been developed to compare the performance of the models in terms of accuracy of the generated predictions, also known as goodness-of-fit [26]. The metrics indicate how well the models predict accurate responses for future inputs. Next, we describe the three metrics that we used in this work, the Coefficient of determination (R^2) [27], the Akaike Information Criterion (AIC) [26] and the Prediction at a given level (PRED) [27].[4]

Coefficient of determination (R^2). This metric “summarizes” how well the generated regression model fits the data. It compares the residues’ deviance to the null deviance (deviance from the mean value); it computes the proportion of the variation of the response variable as estimated using the generated regression compared to the variation of the response variable computed using the null model, i.e., the mean of the values [26]. The following equation formulates the metric.

$$R^2 = 1 - \frac{\sum_{k=0}^{n} (x_i - \hat{x}_i)^2}{\sum_{k=0}^{n} (x_i - \bar{x}_i)^2} \qquad (1)$$

R^2 is commonly used to evaluate linear regression models; it measures how the regression line fits the data. We use it for the three regression methods to measure how, in general, a given regression function fits the data. An R^2 of 1 indicates that the model perfectly fits the data and R^2 of 0 indicates that the model does not explain the data. A value such as 0.5 indicates that about half of the variation in the data can be predicted or explained using the model [26].

The LR method focuses on minimizing R^2. Thus, Spiess and Neumeyer, for example, consider that the metric is not appropriate for evaluating non-linear regression models [29]. Nevertheless, the metric is often used to compare models, e.g., [26]. In this study we use the metric to evaluate the performance of the prediction models in predicting the test dataset and not the training dataset. The metric provides a “summary” of the errors of the predictions.

Akaike Information Criterion. This metric estimates the information loss when approximating reality. The following equation formulates the metric [26].

$$AIC = N \times log(\sum_{k=0}^{n} (x_i - \hat{x}_i)^2 / N) + 2(k+2) \qquad (2)$$

Here N is the number of observations and k is the number of variables. A smaller value indicates a better model.

Prediction at a given level. This metric computes the percentage of prediction falling within a threshold h [30]. The following equation formulates the metric

$$PRED(h) = \sum_{i=1}^{n} \begin{cases} 1 & \text{if } \frac{x_i - \hat{x}_i}{x_i} \leq h \\ 0 & \text{otherwise} \end{cases} \qquad (3)$$

[1] https://www.r-project.org/about.html

[2] https://cran.r-project.org/web/packages/rpart/rpart.pdf

[3] https://cran.r-project.org/web/packages/nnet/nnet.pdf

[4] We avoided the metric Mean of the Magnitude of the Relative Error (MMRE) as it was shown to be misleading [28].

Here N is the number of observations, x_i is the response variable for observation i and h is the threshold, e.g., 25%.

The perfect value for the metric is 100%.

C. Variable importance

This metric measures the relative contributions of the different predicting factors used by the regression method to the response variable. For statistical use, such metric could be, for example, the statistical significance while for business use, the metric could be the "impact on the prediction factor" on the (dependent) response variable. In addition, the metric is often tailored to regression methods, although the metrics may exhibit similarities [31].

In this work we use the variable-importance metric employed in the RPART regression method. The metric measures the sum of the weighted reduction in the impurity method (e.g. the Shannon entropy and the variance of response variable) attributed to each variable [32], [33].[5] It associates scores to each variable, which can be used to rank the variables based on their contribution to the model.

V. Methodology

Figure 2 depicts the process that we used in this study; a process quite similar to the one used by Bener et al. [34]. First, we define the goal of the data-analytics activity, which is: develop a function for predicting the issue fix time using the data that SAP collects on it's processes for fixing vulnerabilities in pre-release and post-release software. The following steps are: collect data that could help achieve the goal; prepare the data to be used to derive insights using statistical methods; explore the collected data sets to understand the used coding scheme, its content, and the relationships between the data attributes; develop prediction models for each of the collected datasets; compute metrics on the model and try to find explanations and arguments for the results. The results of the models analysis were used to identify ways to improve the models. The improvements included the collection of new datasets for dependent information, e.g., about projects. We discuss next the individual steps in more details.

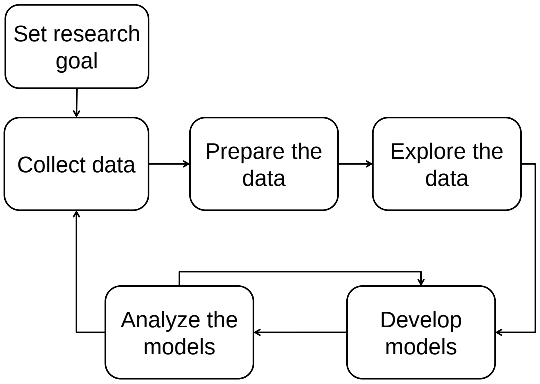


Figure 2. Analysis method.

A. Data collection

SAP maintains three data sets on fixing security vulnerabilities, which we refer to as our main *data sources*. In addition, it maintains a data set about components, and a dataset about projects, which we call *support data*. Table II lists the different datasets we use. The datasets used in our study span over distinct time periods for each dataset (e.g., about 5 years).

Table II
Datasets collected from SAP's tools

Dataset	Description
Data source 1	Vulnerabilities found in ABAP code
Data source 2	Vulnerabilities found in Java and C++ code
Data source 3	Security messages
Support data 1	Components
Support data 2	Projects
Extended data source 2	Extend data source 2 with information about the projects (support data 2)
Extended data source 3	Extend data source 3 with information about the components (support data 1)

The security-testing process records data about fixing issues in two data sets. First, ABAP developers use SLINT for security code analysis. In Data source 1, the tool records data related to a set of attributes about each of the issues it discovers and the tasks performed on these issues. Table III lists these attributes.

Second, Java and JavaScript developers use Fortify[6] and C++ developers use Coverity to analyze software for security issues. In Data source 2, these tools record data related to a set of attributes about each of the vulnerabilities they discover and the tasks performed on these vulnerabilities. Table IV lists these attributes.

In Data source 3, the security response process maintains data about fixing issues discovered in released software. The data is collected and maintained through a Web form; it is not collected automatically as in the case of data sources 1 and 2. The attributes of this data source are listed in Table V.

Each issue can relate to a concrete component. Components are groups of applications that perform a common business goal. A system consists of a set of components. Table VI lists the components attributes.

A software is developed in the context of a project. Table VII lists the attribute of the projects dataset (support

Table III
List of the attributes of ABAP issue fixing (Data source 1).

Attribute	Description
Date_found	Date on which the issue was found
Date_solved	Date on which the issue was closed
Vulnerability_name	Vulnerability types such as memory corruption and buffer overflow
Project_ID	Project identifier
Priority	The priority of fixing the vulnerability. Range: 1 to 4, with 1 highest, 4 lowest priority.

[5] The common approach for evaluating the importance of prediction factors for LR is based on Pearson correlations. This may not apply for the categorical variables, which are common in this study.

[6] Since 2013, SAP uses Checkmarx for analyzing JavaScript. Thus, the use of Fortify by JavaScript developers declines since then.

Table IV
List of the attributes of Java and C++ issue fixing (Data source 2).

Attribute	Description
Date_found	Date on which the issue was found
Date_solved	Date on which the issue was closed
Vulnerability_name	Vulnerability types such as memory corruption and buffer overflow
Scan_source	Tool that performed the scan, i.e., Coverity (for C++ code) or Fortify (for Java code)
Project_name	Project identifier
Folder_name	Indicates the required behavior of the developer towards the issue, e.g., must fix, fix one of the a set, optional, etc.
Scan_status	Status of the issues, i.e., new, updated, removed and reintroduced (i.e. removed but reopened). It allows to identify whether the issue is addressed or not, and is a false positive or not.
Vulnerability_count	Number of issues of the same vulnerability found at once. This indicates that the issues might be related to the same problem.
Priority	The priority of fixing the vulnerability. Range: 1 to 4, with 1 highest, 4 lowest priority.

Table V
List of the attributes for security messages (Data source 3).

Attribute	Description
CVSS_Score	Common Vulnerability Scoring System (CVSS). The score indicates also the urgency of fixing the vulnerability.
Processor	Identifier of development team/area and, thus, implicitly for the local instantiation of the S^2DL
Reporter	Identifier of the external researcher/company who reported the issue
Source	The source of the reported issue such as internal, security testing tool, customers
Vulnerability_type	Vulnerability type
Priority	Priority of the issue to be fixed: low, medium, or high
Component	Group of applications that perform a common business goal such as sales order or payroll

data 2). We extended data source 2 with project descriptions data; we joined data source 2 and support data 2. We also added three computed fields to the data set:

1) FixtoRelease_period: The time elapsed from fixing the given issue to releasing the software.
2) Dev_period: The time elapsed from starting the development to closing the development of the software that contains the issue.
3) FoundtoRelease_period: The time elapsed from discovering the issue to the releasing of the software that contains the issue.

Table VI
List of the attributes for the components (Support data 1).

Attribute	Description
PTU_area	The area of the component, e.g., CRM, IMS, ERP
Gr_component	Component group, i.e., semantic aggregation of components based on superordinate level
Language	The language(s) used to develop the component: ABAP, Java, ABAP and Java, or unknown
PPMS_product	The name of product that the component is part of, as stated in PPMS (Projects Management System)
Comp_owner	The component's development group
Product_owner	The product's development group

Table VII
List of the attributes for projects (Support data 2).

Attribute	Description
Int_prg_name	The unofficial known name of the project (Internal program name)
Prg_typ_id	Release related vs release unrelated (RR / UR)
Rel_type_id	Project type (standard, etc.)
Rel_typ_id	Release type ID (standard, pilot, etc.)
Delivery_mode_id	Mode of delivery to the customer. Values are on premise, on demand, on mobile, etc.
Maintstrategy_id	Maintenance strategy. There is a codification for the strategies.
Deploy_type	Deployment type. There is a codification for the deployment
D2t_date	Planned end of the test period. The period starts after the development closes
Devclose_date	Closing date of the development
P2d_date	Planned development starting date
P2r_date	Planned release date
Prg_lead_resp	Development team responsible for the project
Risk_expert	Identifier of risk expert (anonymized data).

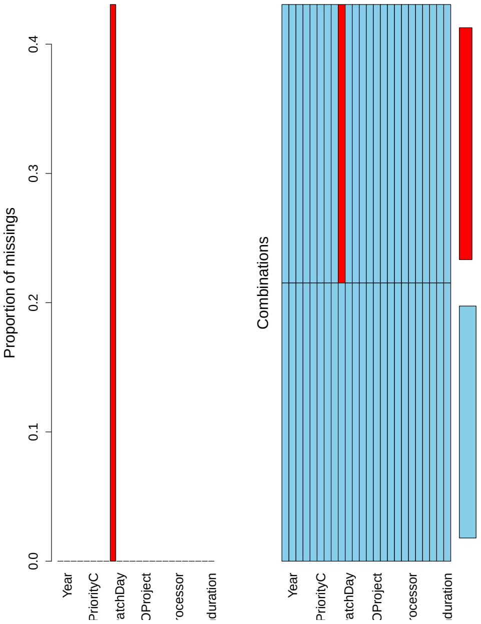


Figure 3. Plot that visualizes missing data for data source 3.

The number of records for each of the basic data sets range from thousands of records to hundred of thousands of records. We did not provide the exact numbers to avoid their misuse (in combination with potentially other public data) to derive statistics about vulnerabilities in SAP products, which would be out of the scope of this work.

B. Data preparation

Using the collected data required us to prepare them for the model-generation routines. The preparation activities required cleaning the data and transforming them as needed for processing.

Data cleaning. First, we identified the data columns where data are frequently missing. Missing values impact the results

Table VIII
COEFFICIENTS OF THE LINEAR REGRESSION OF ISSUE FIX TIME TO SECURITY MESSAGE SOURCE.

Message source	Coefficient	p-value
(Intercept)	249.17	< 0.001
Code scan tool	-50.04	< 0.001
Central security department	-38.05	< 0.001
Customers	-60.68	< 0.001
External research organizations	-102.78	< 0.001
Internal development departments	-12.21	0.304
Test services	-124.74	< 0.001
Validation services	-21.88	0.136

of the regression algorithms because these algorithms may incorrectly assume default values for the missing ones. We used plots such as the one of Figure 3 to identify data columns that require attention.

Second, we developed a set of plots to check outliers – values that are far from the common range of the values of the attributes. We excluded data rows that include semantically wrong values, e.g., we removed records from Data source 1 where the value of ”Date_found” is 1 Dec. 0003.

Third, we excluded records related to issues that are not addressed yet; we cannot deduce issue fix time of such records.

Fourth, we excluded records that include invalid data. For example, the vulnerability type attribute of Data source 2 includes values such as “not assigned”, “?”, and “&novuln.” The records that have these values are excluded. There is no interpretation of prediction results that include these values.

Fifth, we excluded non-useful data attributes. These include, for example, the case where the attribute is derived from other attributes that are considered in the models.

Data transformation. First, we transformed the data of some columns from type text to appropriate types. For instance, we transformed the data of the CVSS column to numeric. Next, we computed new data columns from the source (original) data. For example, we computed the issue fix time from the issue closing date and issue discovery date or we performed some attributes' value transformations to obtain machine readable data for model generation. Some attributes contain detailed information that reduces the performance of the regression algorithms. We addressed this issue by developing a good level of data aggregation for the prediction algorithm. For example, the original dataset included 511 vulnerability types. We grouped the vulnerabilities types in vulnerability categories, which helps to derive better prediction models. Also, we aggregated the “component” variable to obtain ”Gr_component” to include in our regression.

C. Data exploration

We developed a set of plots and statistics about the frequencies of values for the factors and the relationship between the issue fix time and some of the prediction factors. For example, Figure 4 shows the relationship between the issue fix time in days and vulnerability type. This gives us a first impression of the relations among the attributes of a given data set. Also, Table VIII shows the coefficients of the Linear Regression (LR) of the issue fix time using the factor message source, that identifies the source of the reported issue. The table shows that the coefficients in this categorical factor indicate the different contributions of the factor on the issue fix time. The results indicate different impacting strengths of the different sources of security messages (e.g., external parties, customers or the security department) on the issue fix time.

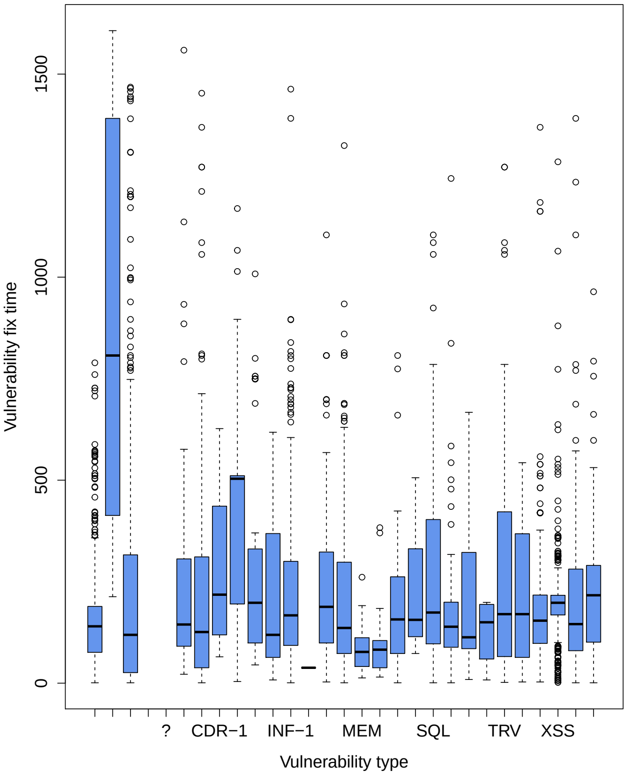


Figure 4. Relationship between issue fix time (in days) and vulnerability types in the context of Data source 3. CDR-1, INF-1, MEM, SQL, TRV, XSS are internal codes for vulnerabilities types and code “?” indicates unknown or uncategorized type of reported vulnerabilities. (Some vulnerability types do not appear on the X axis to ensure clarity.)

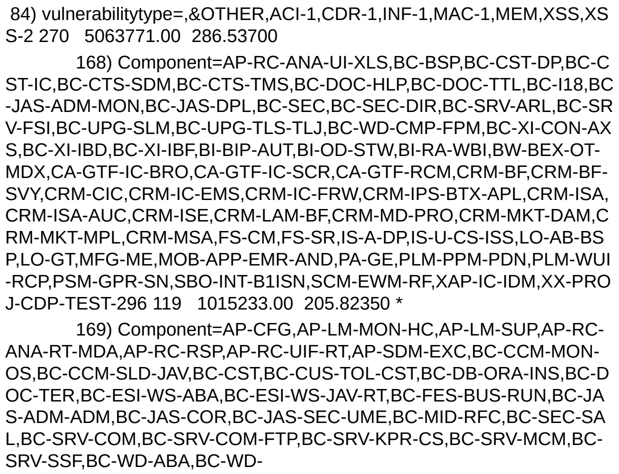
D. Models development

We partitioned each prepared data source into a training set that includes 80% of the data and a test set that includes the remaining 20%. We used the training set to develop the prediction models, or fits, and the test set to assess the goodness of the generated models. The selection of the records for both sets is random.

Next, we performed three operations for each of the main data sources. First, we generated three prediction models using the training set, one using the linear regression method, one using the RPART method, and one using the NNR method. The three data sources have different data attributes and cannot be combined. Thus, we cannot use them together to develop a generic prediction model.

E. Models analysis

We used the variable-importance metric described in Section IV-C to assess the impact of the different prediction



```
 84) vulnerabilitytype=,&OTHER,ACI-1,CDR-1,INF-1,MAC-1,MEM,XSS,XS
S-2 270   5063771.00  286.53700
          168) Component=AP-RC-ANA-UI-XLS,BC-BSP,BC-CST-DP,BC-C
ST-IC,BC-CTS-SDM,BC-CTS-TMS,BC-DOC-HLP,BC-DOC-TTL,BC-I18,BC
-JAS-ADM-MON,BC-JAS-DPL,BC-SEC,BC-SEC-DIR,BC-SRV-ARL,BC-SR
V-FSI,BC-UPG-SLM,BC-UPG-TLS-TLJ,BC-WD-CMP-FPM,BC-XI-CON-AX
S,BC-XI-IBD,BC-XI-IBF,BI-BIP-AUT,BI-OD-STW,BI-RA-WBI,BW-BEX-OT-
MDX,CA-GTF-IC-BRO,CA-GTF-IC-SCR,CA-GTF-RCM,CRM-BF,CRM-BF-
SVY,CRM-CIC,CRM-IC-EMS,CRM-IC-FRW,CRM-IPS-BTX-APL,CRM-ISA,
CRM-ISA-AUC,CRM-ISE,CRM-LAM-BF,CRM-MD-PRO,CRM-MKT-DAM,C
RM-MKT-MPL,CRM-MSA,FS-CM,FS-SR,IS-A-DP,IS-U-CS-ISS,LO-AB-BS
P,LO-GT,MFG-ME,MOB-APP-EMR-AND,PA-GE,PLM-PPM-PDN,PLM-WUI
-RCP,PSM-GPR-SN,SBO-INT-B1ISN,SCM-EWM-RF,XAP-IC-IDM,XX-PRO
J-CDP-TEST-296 119   1015233.00  205.82350 *
          169) Component=AP-CFG,AP-LM-MON-HC,AP-LM-SUP,AP-RC-
ANA-RT-MDA,AP-RC-RSP,AP-RC-UIF-RT,AP-SDM-EXC,BC-CCM-MON-
OS,BC-CCM-SLD-JAV,BC-CST,BC-CUS-TOL-CST,BC-DB-ORA-INS,BC-D
OC-TER,BC-ESI-WS-ABA,BC-ESI-WS-JAV-RT,BC-FES-BUS-RUN,BC-JA
S-ADM-ADM,BC-JAS-COR,BC-JAS-SEC-UME,BC-MID-RFC,BC-SEC-SA
L,BC-SRV-COM,BC-SRV-COM-FTP,BC-SRV-KPR-CS,BC-SRV-MCM,BC-
SRV-SSF,BC-WD-ABA,BC-WD-
```

Figure 5. Part of the prediction model generated from data source 3 using RPART method.

factors on the issue fix time for each of the three data sources. The metric indicates that the factor "project name" is very important for Data source 2 and the factor "component" is very important forDdata source 3. The results and their appropriateness were discussed with the security experts at SAP. We extended Data source 2 with Support data 2 (i.e., projects data set) and we extended Data source 3 with Support data 1 (i.e., components data set). Next, we performed the model development phase (section V-D) using the extended datasets. Then, we used each of the prediction models to predict the issue fix time for the test data set and computed the performance metrics (see subsection IV-B) for each model. We discuss the results in the next section.

VI. Study results

This section discusses the developed prediction models addressing issue fix time and their performance, the relative importance of the prediction factors, and the evolution of mean vulnerability fix time over time.

A. *Issue fix time prediction models*

This section aims to address the question: How well do the chosen models (LR, RPART, and NNR) predict the issue fix time from a set of given factors?

Most of the data that we use are not numeric; they are categorical variables, e.g., vulnerability types and component IDs. The number of categories in each of these variables can be very high, for instance there are about 2300 components. The regression algorithms cluster the elements of these categorical variables automatically. However, the clustering does not follow a given semantics, such as aggregation on superordinate component level, i.e., Gr_component in support data 1.

In addition, the prediction models are large, e.g., in the order of a couple of hundred of nodes for the tree-based model and few thousands for the neural-network model. Because of this, it was impractical to plot the prediction models. Figure 5, for instance, shows a prediction model that we generated from data source 3 by using the RPART method. It shows that there is a long list of component IDs (among a set of 2300 components) for the selection of nodes 168 and 169 while also the parent node uses a set of vulnerability types. The dependency on all those values makes it difficult to clearly visualize the generated models.

Nevertheless, it is interesting to observe that the component factor is built upon a set of distinct component classes (i.e. the first three digits indicate the superordinate component level, e.g. CRM for Customer Relationship Management). An investigation of underlying reasons for such kind of automated clustering might reveal project or process-based issue fix time related coherences between these.

B. *Performance of selected regression methods on the prediction of issue fix time*

This subsection addresses the question: Which of the developed regression models gives the most accurate predictions? It reports and discusses the measurements of the performance-metrics (introduced in Section IV-B) that we performed on the models that we generated for predicting the issue fix time. Table IX summarizes the measurements that we obtained.

Coefficient of determination metric. We observe that the LRs method outperforms the RPART and NNR methods for the five data sets. The metric values indicate that the prediction models generated using LR explain about half of the variation of the real values for data source 1 and for data source 2 and explains most of the variations for the remaining data sources. Indeed, the estimates of the model for the extended data source 2 perfectly match the observed values. We note also that the residues metric values indicate that the prediction models generated using the NNR perform worse than the null model, that is, taking the average of the values.

AIC metric. We observe that the LR method outperforms the RPART and NNR methods for two data sets and that the RPART method outperforms the LR and NNR methods for the remaining three data sets. Thus, this metric gives mixed results with respect to performance of the three regression methods.

PRED metric. We observe that the LR method outperforms the RPART and NNR methods for two data sets, the RPART method outperforms the LR and NNR methods for one data set, and the NNR method outperforms the RPART and LR methods for two datasets. This gives mixed results with respect to performance of the three regression methods. However, the NNR performance improves when the data set is extended with related data. For instance, the PRED value increased from 0.73% in the case of Data source 3 to 65.05% for the case of the Extended data source 3. We acknowledge that the PRED value improved also for the LR method for the case of Data source 2 and Extended data source 2. However, the number of records (N) for the Extended data source 2 is low ($N = 380$); the result should be taken with caution.

Different regression methods have shown conflicting performance measurements towards the problem of effort estimation. For example, Gray and MacDonell [21] compared a set

Table IX
MEASUREMENT OF THE PERFORMANCE METRICS OF THE PREDICTION MODELS.

Data set	LR	RPART	NN	Best method
Residuals metric				
Data source 1	0.526	0.498	-1.252	LR (0.526)
Data source 2	0.461	0.44	-0.294	LR (0.461)
Extended data source 2	1	0.956	-0.587	LR(1)
Data source 3	0.944	0.6585	1.944	LR(0.944)
Extended data source 3	0.909	0.701	1.97	LR(0.909)
AIC metric				
Data source 1	122465	123157	141462	LR(122465)
Data source 2	334565	335936	365665	LR(334565)
Extended data source 2	-4877	463	793	RPART(463)
Data source 3	6632	6507	6958	RPART(6507)
Extended data source 3	6581	6421	7057	RPART(6421)
PRED metric				
Data source 1	31.81%	31.74%	0.156%	LR(31.81%)
Data source 2	14.93%	13.96%	33.81%	NN(33.81%)
Extended data source 2	100%	30.32%	39.40%	LR(100%)
Data source 3	33.98%	34.71%	0.73%	RPART(34.71%)
Extended data source 3	35.41%	34.52%	65.05%	NN(65.05%)

of regression approaches using MMRE and PRED metrics. The methods have shown conflicting results; their rank change based on the used performance metrics. For example, they found that based on the MMRE metric, LR outperforms NNR and based on the PRED metric, NNR outperforms LR. This finding was confirmed by Wen et al. [35] who analyzed the performance of several other regression methods. The regression methods have different strengths and weaknesses. Most importantly they perform differently in the presence of small datasets, outliers, categorical factors, and missing values. We found in this study that it is not possible to claim that a regression method is better than the other in the context of predicting the issue fix time. This result supports the findings of Gray and MacDonell [21] and of Wen et al. [35].

C. Relative importance of the factors contributing to issue fix time

This section aims to address the question: What are the main factors that impact the issue fix time? To answer this question, we used RPART [36] to develop prediction models for the five data sources. Given that the factors used in the datasets are different, we present and shortly discuss each dataset separately. In the next chapter, we analyze the factors impact in depth.

Data source 1. Table X reports the importance of the factors used in Data source 1 on issue fix time. The most important factor in this dataset is "Project_ID." followed by "Vulnerability_name". This implies that there is a major contribution of the project characteristics to issue fix time. Unfortunately, there was no additional metadata available on the projects that could have been joined with data source 1 to allow us to further investigate aspects of projects that impact the fixing time.

Data source 2. Table X reports the importance of the factors used in data source 2 on issue fix time. The most important factor in this dataset is "Scan_status." This shows that depending on whether the issue is false positive or not impacts the issue fix time.[7] The second ranking factor is "Project_name", followed by "Vulnerability_name." This results support the observation we had with data source 1. We observe also that the factor "Scan_source," which indicates the static code-analysis tool used to discover the vulnerabilities (i.e., Fortify or Coverity) is ranked at place 5.

Extended data source 2. We extended data source 2 with data that describe the projects and computed three additional variables: the time elapsed between fixing the vulnerability and releasing the software, called FixtoRelease_period; the development period, called Dev_period; and the time elapsed between discovering the vulnerability and releasing the software, called FoundtoRelease_period.

Table X reports the importance of the factors used in the extended data source on issue fix time. We observe that the most important factor is FixRelease_period while a related factor, FoundtoRelease_period has less importance (rank 6). The other main important factors include the development period, the program name, the program development team, the risk expert, and the vulnerability name. We observe that vulnerability name is ranked only at the seventh position.

Data source 3. Table X reports the importance of the factors used in data source 3 on the issue fix time. The most important factor in this dataset is the development team who addresses the issue (processor) followed by the software component that needs to be changed. We observe that the vulnerability name (i.e., vulnerability type) has a moderate importance, ranked 4th, while the CVSS score is ranked on the 6th position.

Extended data source 3. We extended data source 3 with data that describe the components. Table X reports the importance of the factors used in data source 3 on the issue fix time. The most important factors in this extended dataset is the component, followed by the development team (processor), the development team responsible for the component, the reporter of the vulnerability, and a set of other factors. We observe that

[7] As indicated before, issues marked as e.g., new and updated are not considered in the models; they are for issues that are not addressed yet.

Table X
IMPORTANCE FACTORS.

	Data source 1		Data source 2		Extended data source 2		Data source 3		Extended data source 3	
Rank	**Factor**	**Metric**	**Factor**	**Metric**	**Factor**	**Metric**	**Factor**	**Metric**	**Factor**	**Metric**
1	Project_ID	0.819	Scan_status	0.962	FixtoRelease_period	0.929	Processor	2.917	Component	2.843
2	Vulnerability_name	0.122	Project_name	0.727	Dev_period	0.638	Component	2.756	Processor	2.661
3	Priority	0.089	Vulnerability_name	0.513	Int_prg_name	0.638	Reporter	1.769	Dev_comp_owner	1.677
4	Vulnerability_count	0.058	Folder_name	0.327	Prg_lead_resp	0.638	Vulnerability_type	1.709	Reporter	1.630
5			Scan_source	0.245	Risk_expert	0.638	Source	0.671	Vulnerability_type	0.827
6			Vulnerability_count	0.070	FoundtoRelease_period	0.541	CVSS_score	0.186	Dev_product_owner	0.375
7			Priority	0.053	Vulnerability_name	0.488	Source	0.172		
8					Folder_name	0.061	PPMS_product	0.133		
9					Priority	0.061				
10					Vulnerability_count	0.048				

the vulnerability name (i.e., vulnerability type) has a moderate importance, ranked 5th, and the importance of the factor CVSS score decreased considerably.

D. Evolution of the issue fix time

This section aims to address the question: Is the company improving in fixing security issues? The tendency of the issue fix time could be used as "indicator" of such improvement. For instance, increasing time indicates deteriorating capabilities and decreasing time indicates improving process. The information should not be used as an evidence but as indicator of a fact that requires further investigation.

We modeled the evolution of the mean issue fix time for the resolving (closing) issue month[8] for the three data sources using the Linear Regression (LR), which shows the trend of the response variable over time. Figure 6 depicts respectively the mean issue fix time for (a) data source 1 (pre-release ABAP-based code), (b) data source 2 (pre-release Java, C++, and JavaScript-based code), and (c) data source 3 (post-release security issues).

The figure indicates a fluctuation of the mean issue fix time but with an increasing trend. This trend indicates a deteriorating performance with respect to fixing security issues. A close look at the figure shows that there is a recent reverse in the tendency, which indicates a response to specific events such as dedicated quality releases or the development of new flag ship products. So called quality releases are releases that focus on improving the product quality instead of focusing on new features. To ensure a high level of product quality and security of SAP products, top level managements plans, once in a while, for such quality releases. Also the development of new flag shop products that change the development focus of a large fraction of all developers at SAP can have an influence. Such a shift might result in significant code simplifications of the underlying frameworks.

Figure 6 shows that the increasing global trend applies for pre-release as well as post-release issues. We believe that this indicates that the causes of the increase of the mean issue fix time applies to both cases. We again see that the management actions impacted both cases.

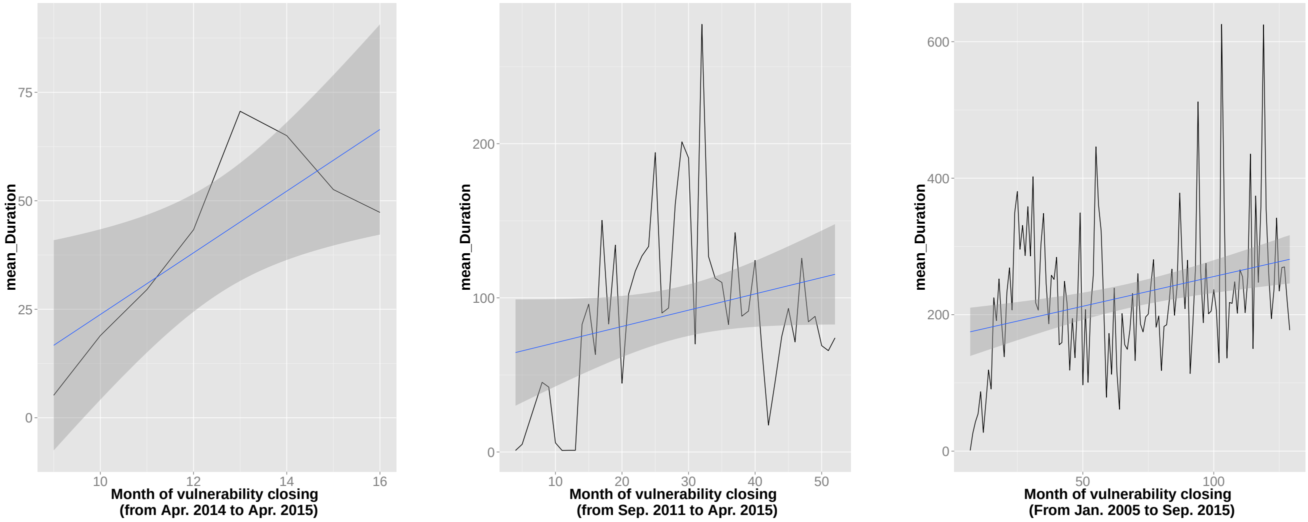
Berner et al. [34] advice that models are sensitive to time. This work supports the claim because it shows that the issue fix time is sensitive to the month of closing the issue.

VII. ANALYSIS OF THE IMPACTING FACTORS

We observe from Data source 1 and Data source 2 that projects (represented by e.g., Project_ID, and Project_name data attributes) have major contributions to issue fix time for the case of pre-release issue fixing. The extension of Data source 2 with project-related data confirmed our observation: the most impacting factors of pre-release issues on the time to fix are project characteristics. Among these characteristics we find the time between issue fixing and software release, project development-duration, and the development team (data attribute Int_prg_name). We believe that the factor time between issue fixing and software release indicates that developers tend to fix vulnerabilities as the release date becomes close. This is not surprising, since they must address all open issues before the software can pass the quality gate to be prepared for release. We expect that the factor project development-duration is related to e.g., the used development models, and the component-related characteristics. Further data analysis shall provide insights about the impact of the factors related to the project development-duration, such as component complexity. For instance, updating smaller component could be easy and be performed in short development cycles while updating complex components requires long development cycles. In addition, we believe that the factor development team indicates the level of expertise of the developers and the smoothness of communication and collaboration among the team. Nevertheless, it is interesting to observe that the influence of vulnerability type decreases when project-related factors are included.

There are two additional dominant factors for the issue fix time, based on the analysis of Data source 2: scan status and folder name. We believe that the factor scan status indicates that the developers address issues based on their perception of whether the given issue is a false positive or not and whether it is easy to address or not. For example, they may close false positive issues that are easy to analyze and postpone addressing issues that are difficult to analyze and/or fix to e.g., when the time for the quality gates becomes close. We also expect that the factor "folder name" indicates that the developers behave differently towards issues flagged must fix, fix one of the set, or optional to fix.

[8] Compute the mean issue fix time for the vulnerabilities resolved (addressed) in the specified month.



(a) Data source 1

(b) Data source 2

(c) Data source 3

Figure 6. Trend of the issue fix effort by closing month. The x axis indicates the number of months elapsed since the start date of the data. The y axis indicates the mean issue fix time in number of days.

The analysis of Data source 2 reveals that the security scan tools (represented by the data attribute Scan_source) is not a leading factor of issue fix time. It is possible that the developers rely on their expertise in analyzing security issues and not on the tool features as they get experts in addressing security issues. Further analysis may explain the finding better.

The results obtained from the analysis of Data source 3 (and its extension) suggests that the software structure and development-team characteristics are the dominant factors that impact the issue fix time for the case of post-release issue fixing. (Note that issues for post-release are not related to projects but to released components.). The analysis results show that the component factor is among the most impacting factors on the issue fix time, which indicates the impact of software structure characteristics. Unfortunately, we do not have, at this moment, data that describe the components, such as the component's complexity, which could be used to get detailed insights about these characteristics.

The results obtained from the analysis of Data source 3 shows the dominance of the impact of processor and reporter on the issue fix time, and thus, the importance of the human-related factors. The importance of the reporter factor is aligned with the results of Hooimeijer and Weimer [37], who found a correlation between a bug reporter's reputation and triaging time: we confirm the importance of the reporting source on vulnerability fix time. Just as Zhang et al. [15], we identified severity as an impacting factor, represented by the ”CVSS_score” in our study. The higher the score is, the faster the vulnerability gets fixed. However, in their study, the severity level was found to be the most influencing factor on a developer's delay before fixing. As opposed, the priority factor in our study–which represents the issue's severity level–has only a minor contribution on the issue fix time.

Our previous qualitative study [4], which was based on expert interviews at SAP, revealed several factors that impact the issue/vulnerability fix time, such as communication and collaboration issues, experience and knowledge of the involved developers and security coordinators, and technology diversification. The results of this study confirm the impact of some of these factors–and shows their importance. For example, the category technology diversification included factors related to technologies and libraries supported by the components associated with the given vulnerability. The impact of the component, found in our current study, might reflect these underlying factors. Unfortunately, it was only possible to relate components' attributes to security messages, i.e. post-release issues, not to pre-release issues to further investigate the components' impact on these. The impact of the development groups might reflect the importance of the experience and vulnerability- and software-related knowledge of the teams as well as the importance of the smoothness of communication and collaboration between the involved stakeholders.

At SAP, the project development-teams work independently; e.g., they choose their own development model and tools, as long as they confirm to the corporate requirements, such as the global security policies. Further investigation of component-, project-, human-, and process-related characteristics of the development teams might reveal more insights on the underlying factors that impact the issue fix time. Such investigation may reveal why certain products/teams are more efficient than others. Reasons may, for example, include the local setup of the communication structures, the used development model–SCRUM, DevOps, etc.–and the security awareness of teams. Another potential factor to check the impact of is the number of people involved in fixing the given issue. This factor was found to impact the fix time of bugs [38], [12]. Controlling these factors allows to control the issue fix time, and thus, the cost of addressing security issues.

A question worth also investigating is: Are the factors that impact the time for addressing pre-release and post-release issues similar? We argued in Section III that the processes for fixing pre- and post-release issues are different, which shall impact the issue fix time for both cases. Nevertheless, acquiring evidence to answer this question requires using the same data attributes for both cases, which may not be possible, at the moment, with data collected at SAP.

VIII. Study Validity and impacts

This section discusses the impacts of the finding and the limitations of the study.

A. Impacts of the findings

This study showed that the models generated using the LR, RPART, and NNR methods have conflicting accuracy measurements in predicting the issue fix time. This implies that the conflict in the performance measurements in estimating software development effort, e.g., in [35], applies to security issues. We infer from this result that there is no better regression method, from the analyzed ones, when it comes to predicting security issue time. We believe that more work needs to be done to develop regression methods appropriate for predicting issue fix time.

The second main finding of this study is that vulnerability type is not the dominant impacting factors for issue fix time. Instead, the dominant factors are the component in which the vulnerability resides, the project related to the issue, the development groups that address the issue, and the closeness of the software release date, a process-related information. This result implies that we should focus on the impact of software structure, developers' expertise and knowledge, and secure software development process when investigating ways to reduce the cost of fixing issues.

The third main finding is that the monthly mean issue fix time changes with the issue closing month. We can infer from this result that the prediction models are time sensitive; that is, they depend on the data collection period. This supports Berner et al. advice to consider recently collected data when developing prediction models [34]. We infer, though, that prediction models are not sufficient for modeling issue fix time since they provide a static view. We believe that prediction methods should be extended to consider time evolution; that is, combine prediction and forecasting.

Finally, SAP can use the models to implement a continuous improvement of its secure software development process and to measure the impact of individual improvements. Other companies can use similar models and mechanism to realize a learning organization.

B. Limitations

There is a consensus among the community that there are many "random" factors involved in software development that may impact the results of data analytics experiments [34]. This aligned with Menzies et al.'s [16] findings about the necessity to be careful about generalization of results related to effort estimations in a global context.

The data analysis described in this report suffers from the two common threats to validity that apply to effort estimation [17]. First, the conclusions are based on the data that SAP collects about fixing vulnerabilities in its software. Changes to the data-collection processes, such as changes to the attributes of the collected data, could impact the predictions and the viability of producing predictions in the first place. Second, the conclusions of this study are based on the regression methods we used, i.e., LR, RPART, and NNR. There are many other single and ensemble regression methods that we did not experiment with. We note that performance issues due to the size of the datasets inhibit us from using random forest [20] and boosting [20], two ensemble regression methods.

In addition, the data is collected over 5 years. During that time SAP refined and enhanced its secure software development processes. This could bias our results. The identification of major process changes along with the times of the changes and a partitioning of the data accordingly might reduce such bias and reveal measurable insights about impacts of process changes on issue fix time.

On the positive side, the conclusions are not biased by the limited data size and the subjectivity in the responses. First the number of records of each of the data set was high enough to derive meaningful statistics. Second, the data is generated automatically and do not include subjective opinions, except the CVSS score of datasource 3. This score is generated based on issue related information that is assessed by the security coordinator responsible for the issue.

Our findings are based on particular data sets of SAP and might mirror only the particularities of time to fix issues for this organization. However, SAP has a diversified software portfolio, the development teams are highly independent in using development processes and tools (as long as they follow generic guidelines such as complying with corporate security requirements), teams are located in different countries, and software are developed using several programming languages (e.g., ABAP, C++, and Java). These characteristics encourage us to believe that the findings apply to industrial companies in general and therefore contribute to the discussion about predicting the issue fix time.

IX. Lessons learned

Data analytics methods are helpful tools to make generalizations about past experience [34]. These generalizations require considering the context of the data being used. In our study we learned few lessons in this regard.

Anonymization. Companies prefer provide anonymized data for data analytics experiments and keep the anonymization map to trace the results to the appropriate semantics. There is a believe that the analyst would develop models and the data expert (from the company) would interpret them using the anonymization map. We initially applied the technique and we found that it prevents the analyst from even cleaning the data correctly. We worked closely with the owner of the data to understand them, interpret the results, and correct or improve the models. The better the data analyst understands the data, the more they are able to model them.

Table XI
PREDICTED VALUES FOR AUTOMATICALLY CLUSTERED COMPONENT FACTOR AND GR_COMPONENT.

Error Metric	LR		RPART		NNR	
	AC	MC	AC	MC	AC	MC
RSQ	0.98	0.76	0.80	0.7045	2.02	1.92
AIC	6586	6187	6461	6139	7033	6733
PRED	33.88	34.12	33.48	32.6915	0.48	0.67

Note: AC is for automatic clustering of components
and MC is for manual clustering of components

Prediction using time-series data. We initially sliced the data sequentially into folds (sliced them based on their order in the dataset) and used the cross-validation method in the regression.[9] We found that the performance metrics of the generated prediction models deviate considerably. To explore this further, we developed the tendency of the mean issue fix time shown by Figure 6. The figure shows a fluctuation of the issue fix time over time. This leads to believe that the prediction models are of temporal relevance as claimed by Berner et al. [34]. The lesson warns to check whether the data are time series or not when using cross-validation with sequential slicing of the data in the regression.

A more generic lesson that we learned concerns **Attribute values clustering.** We found in this study an insignificant small difference in the performance of the prediction models that automatically cluster components and the ones that use semantically clustered components instead. The latter aggregates components based on a semantic based on superordinate level, i.e. Gr_component. Manual investigation is necessary to infer the component characteristics that the algorithms silently used in the clustering. Table XI, for example, shows that the performance of the prediction models using the automated clustering and using the manual clustering are similar. This implies that manual clustering does not provide additional information.

X. CONCLUSIONS

We developed in this study prediction models for issue fix time using data that SAP, one of the largest software vendors worldwide, and the largest in Germany, collects about fixing security issues in the software development phase and also after release. The study concludes that none of the regression methods that we used (Linear Regression (LR), Recursive PARTitioning (RPART), Neural Network Regression (NNR)) outperforms the others in the context of predicting issue fix time. Second, it shows that vulnerability type does not have a strong impact on the issue fix time. In contrast, the development groups involved in processing the issue, the component, the project, and the closeness of the release date have strong impact on the issue fix time.

We also investigated in this study the evolution of the mean issue fix time as time progresses. We found that the issue fix time fluctuates over time. We suggest that better models for predicting issue fix time should consider the temporal aspect of the prediction models; they shall combine both prediction technique and forecasting techniques.

ACKNOWLEDGMENTS

This work was supported by SAP SE, the BMBF within EC SPRIDE, and a Fraunhofer Attract grant.

REFERENCES

[1] G. McGraw, *Software Security: Building Security In*, ser. Addison-Wesley Software Security Series. Boston, MA, USA: Pearson Education Inc, 2006.

[2] R. Bachmann and A. D. Brucker, "Developing secure software: A holistic approach to security testing," *Datenschutz und Datensicherheit (DuD)*, vol. 38, no. 4, pp. 257–261, apr 2014.

[3] M. Howard and S. Lipner, *The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software*. Microsoft Press, 2006.

[4] L. ben Othmane, G. Chehrazi, E. Bodden, P. Tsalovski, A. Brucker, and P. Miseldine, "Factors impacting the effort required to fix security vulnerabilities," in *Proc. Information Security Conference (ISC 2015)*, Trondheim, Norway, Sep. 2015, pp. 102–119.

[5] H. Keller and S. Krüger, *ABAP Objects*. SAP PRESS, 2007.

[6] G. Chehrazi, C. Schmitz, and O. Hinz, "QUANTSEC - ein modell zur nutzenquantifizierung von it-sicherheitsmaßnahmen," in *Smart Enterprise Engineering: 12. Internationale Tagung Wirtschaftsinformatik, WI 2015, Osnabrück, Germany, March 4-6, 2015.*, 2015, pp. 1131–1145. [Online]. Available: http://www.wi2015.uni-osnabrueck.de/Files/WI2015-D-14-00049.pdf

[7] D. Cornell, "Remediation statistics: What does fixing application vulnerabilities cost?" in *RSAConference*, San Fransisco, CA, USA, Feb. 2012. [Online]. Available: http://www.rsaconference.com/writable/presentations/file_upload/asec-302.pdf

[8] H. Zeng and D. Rine, "Estimation of software defects fix effort using neural networks," in *Proc. of the 28th Annual International Computer Software and Applications Conference (COMPSAC 2004)*, vol. 2, Hong Kong, China, Sept 2004, pp. 20–21 vol.2.

[9] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in *Proc. of the Fourth International Workshop on Mining Software Repositories*, ser. MSR '07, Washington, DC, USA, 2007, pp. 1–.

[10] L. D. Panjer, "Predicting eclipse bug lifetimes," in *Proceedings of the Fourth International Workshop on Mining Software Repositories*, ser. MSR '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 29–. [Online]. Available: http://dx.doi.org/10.1109/MSR.2007.25

[11] P. Bhattacharya and I. Neamtiu, "Bug-fix time prediction models: Can we do better?" in *Proceedings of the 8th Working Conference on Mining Software Repositories*, ser. MSR '11. New York, NY, USA: ACM, 2011, pp. 207–210. [Online]. Available: http://doi.acm.org/10.1145/1985441.1985472

[12] E. Giger, M. Pinzger, and H. Gall, "Predicting the fix time of bugs," in *Proceedings of the 2Nd International Workshop on Recommendation Systems for Software Engineering*, ser. RSSE '10. New York, NY, USA: ACM, 2010, pp. 52–56. [Online]. Available: http://doi.acm.org/10.1145/1808920.1808933

[13] M. Hamill and K. Goseva-Popstojanova, "Software faults fixing effort: Analysis and prediction," NASA Goddard Space Flight Center, Greenbelt, MD United States, Tech. Rep. 20150001332, Jan. 2014.

[14] R. Hewett and P. Kijsanayothin, "On modeling software defect repair time," *Empirical Software Engineering*, vol. 14, no. 2, pp. 165–186, 2009.

[15] F. Zhang, F. Khomh, Y. Zou, and A. Hassan, "An empirical study on factors impacting bug fixing time," in *19th Working Conference on Reverse Engineering (WCRE)*, Kingston, Canada, Oct 2012, pp. 225–234.

[16] T. Menzies, A. Butcher, A. Marcus, T. Zimmermann, and D. Cok, "Local vs. global models for effort estimation and defect prediction," in *Proc. of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '11, Washington, DC, USA, 2011, pp. 343–351.

[17] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *Software Engineering, IEEE Transactions on*, vol. 33, no. 1, pp. 2–13, Jan 2007.

[9] This slicing method allows for easily splitting all the data among the folds.

[18] Y. Shin, A. Meneely, L. Williams, and J. Osborne, “Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities,” *Software Engineering, IEEE Transactions on*, vol. 37, no. 6, pp. 772–787, Nov 2011.

[19] A. D. Brucker and U. Sodan, “Deploying static application security testing on a large scale,” in *GI Sicherheit 2014*, ser. Lecture Notes in Informatics, vol. 228, mar 2014, pp. 91–101. [Online]. Available: http://www.brucker.ch/bibliography/abstract/brucker.ea-sast-expierences-2014

[20] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning with Applications in R*. New York, US: Springer-Verlag, 2013.

[21] A. R. Gray and S. G. MacDonell, “A comparison of techniques for developing predictive models of software metrics,” *Information and Software Technology*, vol. 39, no. 6, pp. 425 – 437, 1997.

[22] R. F. J. Hastie, Trevor; Tibshirani, *The Elements of Statistical Learning*, 2nd ed. Springer, 2013.

[23] T. Menzies, “Data mining: A tutorial,” in *Recommendation Systems in Software Engineering*, M. P. Robillard, W. Maalej, R. J. Walker, and T. Zimmermann, Eds. Springer Berlin Heidelberg, 12 2013, pp. 39–75.

[24] L. Breiman, J. Friedman, C. J. Stone, and R. Olshen, *Classiffication and Regression Trees*. Belmont, CA: Chapman and Hall/CRC, 1984.

[25] D. F. Specht, “A general regression neural network,” *Neural Networks, IEEE Transactions on*, vol. 2, no. 6, pp. 568–576, Nov 1991.

[26] R. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*. Otexts, 2014.

[27] E. K. T. Menzies; and E. Mendes, “Transfer learning in effort estimation, empirical software engineering,” *Empirical Software Engineering*, vol. 20, no. 3, pp. 813–843, June 2015.

[28] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit, “A simulation study of the model evaluation criterion mmre,” *IEEE Transactions on Software Engineering*, vol. 29, no. 11, pp. 985–995, Nov 2003.

[29] A.-N. N. Spiess and N. Neumeyer, “An evaluation of R2 as an inadequate measure for nonlinear models in pharmacological and biochemical research: a Monte Carlo approach.” *BMC pharmacology*, vol. 10, no. 1, pp. 6+, Jun. 2010. [Online]. Available: http://dx.doi.org/10.1186/1471-2210-10-6

[30] E. Kocaguneli, T. Menzies, and J. Keung, “On the value of ensemble effort estimation,” *Software Engineering, IEEE Transactions on*, vol. 38, no. 6, pp. 1403–1416, Nov 2012.

[31] U. Grmping, “Variable importance assessment in regression: Linear regression versus random forest,” tO FINISH.

[32] G. Louppe, L. Wehenkel, A. Sutera, and P. Geurts, “Understanding variable importances in forests of randomized trees,” in *Advances in Neural Information Processing Systems 26*, C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, Eds., 2013, pp. 431–439. [Online]. Available: http://media.nips.cc/nipsbooks/nipspapers/paper_files/nips26/281.pdf

[33] K. M. Eisenhardt, “Building theories from case study research,” *Academy of Management Review*, vol. 14, no. 4, pp. 532–550, October 1989.

[34] A. Bener, A. Misirli, B. Caglayan, E. Kocaguneli, and G. Calikli, *The Art and Science of Analyzing Software Data*, 1st ed. Waltham, USA: Elsevier, Aug. 2015, ch. Lessons Learned from Software Analytics in Practice, pp. 453–489.

[35] J. Wen, S. Li, Z. Lin, Y. Hu, and C. Huang, “Systematic literature review of machine learning based software development effort estimation models,” *Information and Software Technology*, vol. 54, no. 1, pp. 41 – 59, 2012.

[36] T. M. Therneau and E. J. Atkinson, “An introduction to recursive partitioning using the rpart routines,” Mayo Foundation for Medical Education and Research; Mayo Clinic; and Regents of the University of Minnesota, Minneapolis, USA, Tech. Rep. 61., Oct. 2011. [Online]. Available: http://r.789695.n4.nabble.com/attachment/3209029/0/zed.pdf

[37] P. Hooimeijer and W. Weimer, “Modeling bug report quality,” in *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE ’07. New York, NY, USA: ACM, 2007, pp. 34–43. [Online]. Available: http://doi.acm.org/10.1145/1321631.1321639

[38] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, “”not my bug!” and other reasons for software bug report reassignments,” in *Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work*, ser. CSCW ’11. New York, NY, USA: ACM, 2011, pp. 395–404. [Online]. Available: http://doi.acm.org/10.1145/1958824.1958887

Lotfi ben Othmane is currently the head of the Secure Software Engineering group at Fraunhofer SIT. He received his Ph.D. degree from Western Michigan University (WMU), USA, in 2010; the M.S. degree from University of Sherbrooke, Canada, in 2000; and the B.S degree from University of Sfax, Tunisia, in 1995. He worked for 13 years in the industry in Tunisia, Canada, and USA. Dr ben Othmane has about 30 peer-reviewed publications. He is currently investigating the use of data science in secure software development and the development of secure systems using the agile approach.

Golriz Chehrazi is a PhD candidate at the department of Information Science, i.e. Electronic Markets, at TU Darmstadt and works as research assistant at the Secure Software Engineering department at Fraunhofer SIT, Germany. She received her Diploma-degree in Wirtschaftsinformatik (Business Information Systems) at the Technische Universitt Darmstadt, Germany, in 2009 and her Master of Science in Computer Science at Linkping Universitet, Sweeden, in 2006. She is currently investigating the use of empirical analyses of IT security issues in open source projects and the measurement of economic impacts of security in software development.

Eric Bodden is professor for Software Engineering at the University of Paderborn and at Fraunhofer IEM. At the time this research was conducted he was cooperative professor for Secure Software Engineering at Fraunhofer SIT and Technische Universität Darmstadt. Bodden received his Ph.D. in 2009 from McGill University, Montréal, Québec, Canada. His research has been honored with numerous awards, including the Heinz Maier-Leibnitz-Price of the Deutsche Forschungsgemeinschaft (DFG) and two ACM Distinguished Paper Awards. In 2014, the magazine Capital elected him one of the top 40 researchers under 40.

Petar Tsalovski is a security expert and developer at SAP SE. He is a graduate from the University of Mannheim and is has been working in the area of Security Testing, Validation & Dependency Analysis since 2011. His interest and main area of expertise are security testing and data analysis. He is currently working on the research of security testing KPI’s and developing an S2DL-enabling service tool.

Achim D. Brucker is a Research Expert and Security Testing Strategist at SAP SE. He holds a PhD from ETH Zurich, Switzerland and his research areas are security, software engineering, and formal methods. He is interested in tools and methods for modelling, building, validating, and verifying secure and reliable systems. He also participates in the OCL standardisation process of the OMG. Further information can be found on his website: http://www.brucker.ch.