Chapter 3

An Introduction to Data Analytics For Software Security

Lotfi ben Othmane

Fraunhofer Institute for Secure Information Technology, Darmstadt, Germany

Achim D. Brucker

Department of Computer Science, The University of Sheffield, Sheffield, UK

Stanislav Dashevskyi

University of Trento, Italy

Peter Tsalovski

SAP SE, Dietmar-Hopp-Allee 16, 69190 Walldorf, Germany

CONTENTS

3.1 Introduction

Secure software development, e. g., following processes similar to Microsoft's Security Development Lifecycle (SDL) [18], is considered to be an important part of developing secure software. On the one hand, such processes require a significant effort and, on the other hand, generate (potentially) a large amount of data—both on the process level (e. g., process descriptions and regulations) where reported, as well as on the technical level (e. g., results of static code analysis tools).

The large effort put into secure software development immediately raises the question, if this investment is effective and if the effort can be invested more effectively. While, at the first sight, it looks like the generated data provides the basis for an answer, this is, in our experience, not the case: data often does not exist in the necessary quality and quantity. This can be caused by processes being constantly improved (changed), sometimes in an undocumented way, while recording data. Moreover, the large variety of security-related challenges can also work against statistical methods: if one is interested in analysing a specific vulnerability or development approach, the actual data set for this specific part of the overall picture might be rather small. Thus, the successful application of data science methods for improving the software security or for making software security processes more efficient required careful planning to record the right data in the necessary quality.

In this chapter, we report on our own experiences [8, 2] in empirical secure software research at, SAP SE, the largest European software vendor. Based on this, we derive an actionable recommendations for building the foundations of an expressive data science for software security: we focus on using *data analytics* for improving the secure software development. Data analytics is the science of examining raw data with the purpose of drawing conclusions about that information using machine learning methods or statistical learning methods. Data analytical techniques have been successfully used in both the cyber-security domain as well as the software engineering domain. For example, Jackobe and Rudis showed how to learn virus propagation and characteristics of data breaches form public data [20]. Data analytical methods are also commonly used to investigate software engineering challenges such as effort prediction [10]. Thus, applying these techniques to the intersection of both areas *to help practitioners* to develop more secure software with less effort, seems promising.

The rest of the chapter is structured as follows: in Sec. 3.2 we introduce the secure software development life-cycle used at SAP and two case studies that we worked on in collaboration with the central security team of SAP: they motivated our software security analytical process (Sec. 3.3). Afterwards, we introduce the most important learning methods (Sec. 3.4) and techniques for evaluating the performance of the generated models (Sec. 3.5)—both with a strict focus on their application in the software security field. We finish the chapter with few generic lessons that we learned and recommend for data scientist in the software security field (Sec. 3.6) and conclude the chapter (Sec. 3.7).

3.2 Secure Software Development

The case studies we report on in this chapter were done together with the product security team of SAP SE. The processes for secure software development at SAP need to support a wide range of application types (ranging from small mobile apps to large scale enterprise resource planning solutions). These applications are developed using a wide range of software development styles (ranging from traditional waterfall, to agile development to DevOps with continuous delivery). Serving such a diverse software development community is already a very challenging problem, still it gets even more complex as the cultural differences within a globally distributed organization need to be taken into account as well. To allow the flexibility required to meet the different demands, SAP follows a two-staged security expert model:

1. a central security team defines the security global processes, such as the SAP Secure Development Lifecycle (S^{2}DL) or the guidance for consuming FOSS, provides security training programs, risk identification methods, offers security testing tools, or defines and implements the security response process;

2. local security experts in each development area or team are supporting the developers, architects, and product owners in implementing the S^{2}DL and its supporting processes.

This two-staged models allows a high degree of flexibility and adaptability on

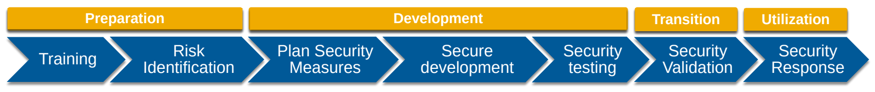

Figure 3.1: Overview of the SAP Secure Development Lifecycle (S^{2}DL).

the local level while ensuring that all products meet the level of security (and quality) SAP customers expect.

To ensure a secure software development, SAP follows the SAP Security Development Lifecycle (S^{2}DL) (which is inspired by Microsft's Security Development Lifecylce [18]). Fig. 3.1 shows the main steps in the S^{2}DL, which is split into four phases: preparation, development, transition, and utilization.

- *Preparation:* This phase comprises all activities that take place before the actual development starts. These activities can be independent of the actual product being developed (e. g., general security awareness trainings) or product specific (e. g., risk identification for a specific product).

  The results of a thorough data analytics and modeling of the following steps of the S^{2}DL contribute significantly to the success of the preparation phase: for example, it helps to identify training needs as well as gaps in the risk analysis.

- *Development:* This phase comprise the steps from planing a new product (or product version) to the actual development. In particular, it covers

  - the *Planning of Security Measures* which describes the mitigation of the previously identified security risks,
  - the *Secure Development* using defensive implementation strategies,
  - the *Security Testing* that ensures that the planned security measures (including the defensive implementation strategies) are implemented and are effective in preventing security threats.

  This phase generated a large amount of data (see, e. g., the datasets used in [3]) that is important for further analysis as well as profits a lot in effectivity and efficiency from a software-security specific data analytical approach. In particular the security testing activities are expensive and require a close monitoring to be successful. We will discuss this in our case studies (see Sec. 3.2.1 and Sec. 3.2.2) in more details.

- *Transition:* The *Security Validation* team is an independent control that acts like the first customer and executes a security analysis and security test of the final products.

- *Utilization:* The *Security Response* team handles the communication with customers and external security researchers about reported vulnerabilities as well as ensures that the development and maintenance teams fix the reported issues (including down-ports to all supported releases as required by the support agreements).

  As the development phase, this phase generated a lot of data that is important for further analysis as well as profits a lot in effectively and efficiency from a software-security specific data analytical approach, e. g., to improve the response and fix times for security vulnerabilities. Thus, it is not surprising that our case studies (see Sec. 3.2.1 and Sec. 3.2.2) also address this phase.

The S^2DL is only one example of a security development lifecycle and that the challenges and lessons learned in this paper are not specific to this particular security development process. We believe that, for example, they are similarly applicable to Microsoft's SDL [18].

In the following, we briefly introduce our two case studies with SAP, which both focus on improving the development and utilization phases of the S^2DL .

3.2.1 Fixing Vulnerabilities and Static Analysis Efforts

We worked, in the first case study [2, 3], together with the central security team of SAP to identify the factors that impact the time for fixing issues[1] (either reported by in-house security testing activities [1, 6] or reported by external security researchers.

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. But by how much? and what are the most influential factors? To answer these questions directly, one would need to trace all the effort of the different actions that the developers undertake to address a security issue: initial triage, communication, implementation, verification, porting, deployment and validation of a fix. Unfortunately, such a *direct* accountability of the individual efforts associated with these action items is impossible to achieve, last but not least due to legal constraints that forbid any monitoring of the workforce. One must therefore opt for *indirect* means to relate quantitative, measurable data, such as the vulnerability type, the channel through which it was reported, or the component in which it resides, to soft human factors that correlate with the time it takes to fix the related vulnerabilities. We described the work that we performed to identify these factors and the results that we obtained in [2, 3].

[1]Experts check each reported issue and confirm that either it is indeed a vulnerability or it cannot be exploited.

3.2.2 Secure Consumption of Third Party Components

Our second case study [8] is also a collaboration with the central security team at SAP: in this project, we focused on the impact of vulnerabilities in consumed third-party code in general and Free/Libre and Open Source Software (FLOSS) in particular.

As the security of a software offering, independently of the delivery model (e.g., cloud software or on premise delivery), depends on all components, a secure software supply chain is of utmost importance. While this is true for both proprietary and as well as FOSS components that are consumed, FLOSS components impose particular challenges as well as provide unique opportunities. For example, while FOSS licenses contain usually a very strong "no warranty" clause (and no service-level agreement), they allow users to modify the source code and, thus, to fix issues without depending on an (external) software vendor.

Thus, it is important to determine the *future* security risk (and, thus, the associated effort) of a third-party already when deciding to use a component. Thus, we worked with the central security team of SAP on validating if static analysis (which was already used successfully at SAP [1, 6]) can be used for assessing FLOSS components. Our research showed that this, while being the original motivation, is not the most urgent question to answer [8]—allowing project teams to plan the *future* security maintenance effort is much more important. Thus, we concentrated our collaboration in developing effort models and predictors for the security maintenance effort. In case of SAP, where software is used over a very long time (i. e., decades) it is very common that old FLOSS version are used that are not necessarily supported by the community: in this scenario it becomes very important to be able to estimate the required maintenance effort that is either caused by down-porting fixes to the actual consumed version or by upgrading the consumed version.

3.3 Software Security Analytical Process

Extracting knowledge from data using data analytical techniques requires (1) identifying the research goal, (2) collecting data, (3) preparing the data, (4) exploring the data, (5) developing analytical models, and (6) analyzing the developed models. Fig. 3.2 depicts this generic data analytics process, which is quite similar to the one used by Bener et al. [4]. The process is iterative. For example, the project partners may decide, after analysing developed models, to extend the datasets.[2] We describe in the following the process activities.

[2]A decision to change the research goal implies starting a new project, as the scope changes.

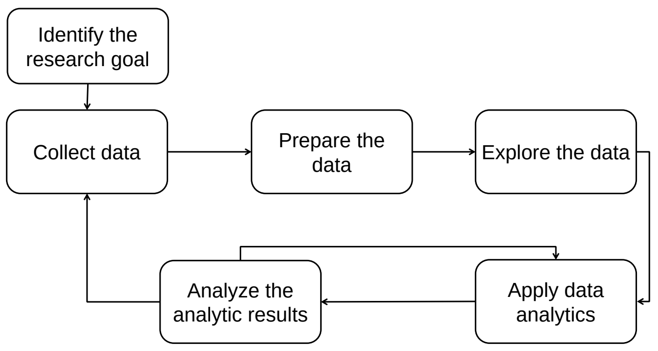

Figure 3.2: The software security analytical process.

3.3.1 Identify the Research Goal

Secure software engineering involves software, people, tools, and processes. Each of these aspects has a complex structure. For example, people involved in secure software engineering include developers, managers, security experts, operation administrators, and incident response engineers, all should collaborate even though their locations and cultures may be diverse. Decisions needs often to be made regarding aspects of secure software engineering, such as resources allocation. These decisions either use knowledge or make assumptions about specific phenomenon. The knowledge could be acquired by testing theoretical models using empirical research methods such as data analytics.

Testing theoretical models using data analytical techniques requires expressing the problem to be investigated as a question, which may be divided into sub-questions. The question could be of common interest or specific to a given organization. Questions of common interest are often raised by scholars curious about specific aspects related to software security. Questions of interest to a given organization are often raised by the organization and need to consider the organization context.

Research questions formulated early in the projects are often vague, imprecise, and cannot be answered/evaluated. Workshops and discussions among the interested parties including the researchers allow to nail down them to precise ones that could be assessed [4]. For example, in our fixing effort project (recall Sec. 3.2.1), the initial goal was: estimate the budget required to fix security issues in a new project. The questions is vague; it does not indicate, for example, whether we need to consider time spent to design generic solutions to vulnerabilities types or not. The project participants, including the researchers, had a workshop to discuss the goal and nail it down to questions that could be assessed using data. We agreed that the practical goal would be: predict the time to fix security issues.

Similarly, in our secure consumption project (recall Sec. 3.2.2) we started with the initial goal to validate if static application security testing is an effective means for ensuring the security of third party components. After several iterations with the product teams, we ended up with the goal of developing effort models that help product teams to actually plan for fixing vulnerabilities in consumed components (respectively, for the effort of upgrading products to later version of a consumed component). Thus, the final research goal was to validate different effort models.

A good question should be precise enough to be formulated mathematically as follows: let y be the response variable, x_i are the set of independent variables (i is the index of the variables), and their relationships could be formulated as: $y = f(x_1, x_2, \ldots, x_n)$ where f represents the systematic information that the variables x_i provide about y [21]. The goal is then to find data that measure x_i and y and to apply data analytics to identify the function f and measure its performance. For instance, the question we identified for the generic question described above is: what is relationship between the time to fix security issues and the characteristics of security issues.

3.3.2 Collect Data

The main challenge in data analytics is the availability of data. Often, the researcher needs to collect artifacts (e. g., source code and documents) that could be used to derive data that can be used to test theoretical models. The sources of the artifacts and datasets could be private and public. Public artifacts could be, for example, a repository of the comments on code changes related to a set of open source software. Such artifacts could be changed to derive data related to fixing security issues in open source software [5]. *Public datasets* could be repositories such as the Promise data repository [31] or the Common Vulnerabilities and Exposures (CVE) database.[3] Public data sources played, e. g., an important role in our secure consumption project (Sec. 3.2.2). *Private datasets* could be, for example, the database of the security code analysis tool, such as Coverity[4] or Fortify[5] of a given organization. These datasets played an important role in our fixing effort project (Sec. 3.2.1). Nevertheless, used datasets must represent the population of the study goal. For example, programs developed by students cannot be used to derive results about characteristics of vulnerable software.

Useful datasets need to contribute to addressing the research goal. This implies that the attributes of the datasets need to be in accordance with the independent and dependent variables of the theoretical models. The researcher needs

[3] `https://cve.mitre.org/`
[4] `http://www.coverity.com/`
[5] `http://www.fortify.com/`

to understand the attributes of the data and the used codification schema that may be used by some attributes, e. g., codification of vulnerabilities types. This should include, for example, the semantic of the data, the scale of the data, the relationships between the data attributes, and the process of collecting the data (including the dates of process changes if possible). For instance, the names of data attributes are often misleading and need clear definitions. For example, the attribute "`open date`" in dataset fixing security issues may imply the date when the code analysis tool reported the issue or the date when the developers started working on the issue.[6] The data attribute definitions impact the interpretations of the results obtained from the data analytical techniques. In addition, the researcher needs to understand the implication of missing data. They need to determine whether they should replace missing data with default values or computed values, or to exclude the rows that have missing data.

Data collection is iterative. At the beginning, the researcher may start with an initial dataset that has a limited set of attributes. Such dataset may be provided by the entity that has interest in the research results or was identified by the researcher. First, the researcher may derive new variables from the data attributes. For instance, they may compute development life-cycle duration by computing the difference between start of a release and end of a release. This was on of the approaches that we took in our fixing effort project (Sec. 3.2.1), see [2] for details. Second, the dataset may be extended with variables that are commonly used for the given research problem. For example, data related to code size, cohesion, and coherence should be collected and used for research concerning predicting whether a software is vulnerable or not as they are commonly used [7].

The researcher may use the initial data to develop initial models that address the research goal. They should present and discuss the initial models they derive from the data with the stakeholders interested in or familiar with the research questions. The findings may spark discussions of usability of the derived models or ways to improve them. Such discussions may lead to ideas to integrate other existing datasets—not discussed before—or collect new data that allow to get better insights. For example, in our fixing effort project (Sec. 3.2.1), the main factor that impact the issue fix time is the projects where the issues are found [2]. The team realized that they have a dataset that describes a subset of the projects—i. e., not all projects have records in the dataset. The dataset was used to develop extended models using the subset of the records of the main dataset related to the projects described in the secondary dataset.

[6] This uncertainty in the time frames could be more complicated if issues could be reopened to address inefficacy of implemented solution: Is the open date the date of first discovery or the date of discovery of the inefficacy of the solution?

3.3.3 Prepare the Data

Collected data are, often, not suited as-is to address the research goal. They may include records that are not related to the research goal and thus should be excluded from the datasets. For example, in our fixing effort project (Sec. 3.2.1), the initial datasets that we received in our study on issue fix time include records of security issues that are still open (not fixed yet). The records were excluded because they do not have issue fix time [2]. In addition, the datasets may include invalid data. Recognizing invalid data requires understanding the semantics of the data attributes. For example, the values "`?`" and "`&novuln`" are not valid vulnerability types. Though, sometimes datasets may include valid values that are poorly-expressed, which should be retained. The researcher should plot the data to analyse the distribution of the values, which allows to identify such problems.

Moreover, not all data is structured. In our secure consumption project (Sec. 3.2.2), CVEs play an important role. In a CVE, a lot of important information is part of the semi-structured or even unstructured part of an CVE entry. Thus, the data preparation phase required a manual translation of unstructured data into a structured form that can be analyzed automatically [8].

Data analytics is performed to validate theoretical models, which relate independent variables to dependent variables. The variables of the theoretical models may not have equivalent in the datasets. In this case, new variables maybe derived (e. g., computed) from data attributes of the dataset. For example, the variable issue fix time could be derived from the issue closing date and issue discovery date such as in [2]. In other cases, the datasets contain attributes, where the values could not be used as-is to address the investigated research problem; derived values are very useful though. For example, comments on code-changes are too detailed and cannot be used by statistical methods; thus, useful information may be derived from these comments and be used to address the research problem [5]. In other cases the data attributes contain detailed information that need to be abstracted to be useful for analytics. For example, in our fixing effort project, we had to group the 511 vulnerability types in vulnerability categories and to group the 2300 components into component families. Then, we used the derived data in the prediction models [2].

Collected datasets have often data attributes where their values are frequently missing. Missing values impact the results of the statistical algorithms because those algorithms may incorrectly assume default values for the missing ones. The researcher should visualize the missing values in their datasets. Fig. 3.3 shows the missing values of one of the datasets we worked with in our fixing effort project (recall Sec. 3.2.1 and [2]). Statistical algorithms have strategies to address them, such as ignore the related record, replace with default values, or extrapolate the values, using e. g., average. The researcher should investigate the causes of the missing values and have a strategy for addressing them.

Data attributes sometimes contain row values collected using diverse meth-

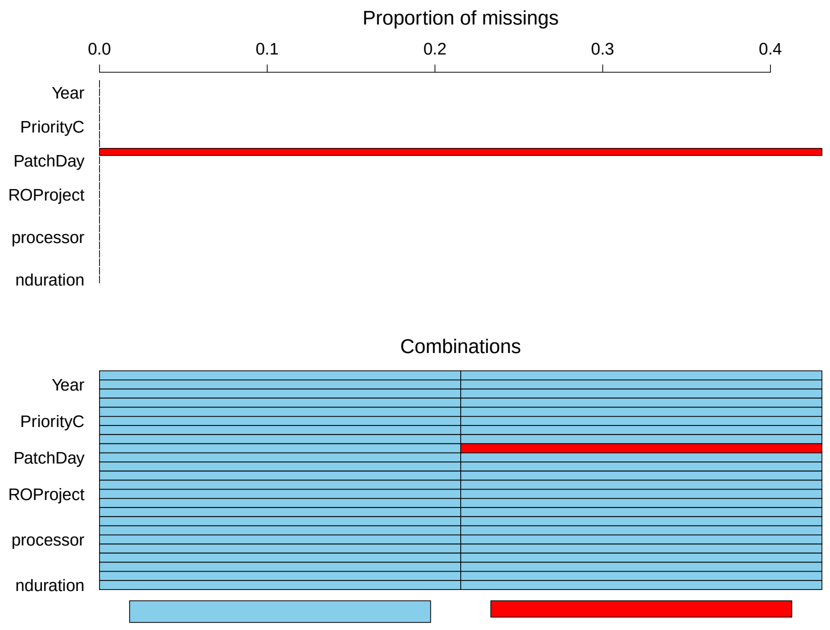

Figure 3.3: Plot that visualizes missing data.

ods and tools. The researcher has two options: (1) consider the diversity when interpreting the results of the data analytics or (2) transform the values such that they become uniform, e. g., use the same measurement metric or have the same format. For example, the Common Vulnerability Scoring System (CVSS) [7] score attribute shall include numeric values computed using only one version of the metric and be within a specific range. In addition, data values are descriptive and may not capture processes changes (e. g., issue fixing process). The researcher may not be able to address such problems, but they should report them in the validity of the analytics' results.

3.3.4 *Explore the Data*

Data describe a phenomenon at a given abstraction level. Deriving statistical models and knowledge from data requires understating the patterns and hidden facts that they embed. In our fixing effort project (recall Sec. 3.2.1), we initially anonymized the vulnerability types [2]. We realized when discussing the use of the data that the anonymization algorithms included bugs—e. g., processing special characters. Thus, we were making wrong statements from the data and it was difficult to detect that.

[7] `https://www.first.org/cvss`

There are three main techniques to explore the data and identify patterns, which are data visualization, correlation analysis, and hypothesis testing. Data visualization concerns the use of plots (box plots, line charts, and basic charts) to visualize the distributional characteristics of the data, such as frequencies and variability. The plots can visualize the basic descriptive statistics, such as the min, max, mean, and variance for numeric data attributes and levels for factors data attributes. They can also visualize the basic relationships and patterns in the data. In addition, they could be used to detect outliers, i. e., data values that are outside the expected range. For example, we developed in the fixing effort project a plot relating the issue fix time to vulnerability type, shown in Fig. 3.4. The figure shows that vulnerability type impacts moderately the issue fix time.

Correlation analysis concerns using statistics to identify the relationships between variables represented by data attributes. The two commonly used approaches to measure correlation are Pearson's correlation coefficient and Spearman's rank correlation coefficient [47]. The Pearson's correlation coefficient measures the linear relationship between two variables. The coefficient values range between 1 and -1. In software engineering, a coefficient whose absolute value is above ± 0.75 implies a strong relationship and a coefficient whose absolute value is less than ± 0.3 implies the correlation is weak. Spearman's rank correlation coefficient measures the relationship between the ranks of the data values of the variables. The coefficient values range also between 1 and -1 [47]. The information is critical as it shows the dependencies between the variables, which allows to choose the (independent) variables to use in the predictive models. For example, the correlation between issue fix time and CVSS score was found to be weak, it has a score of -0.051 [2].

The data visualization and correlation analysis may show the researcher patterns, such as distribution of values of specific variables or the equality of two samples. The researcher may use hypothesis testing techniques, such as t-test or Mann-Whitney U-test to check such hypothesis.

3.3.5 Apply Data Analytics

Recall that a data analytics problem should be formulated as: $y = f(x_1, x_2, \ldots, x_n)$ where y is the dependent variable (also called response variable), x_i are the set of independent variables (also called features and predictors) where i is the variable index, and f represents the systematic information that the variables x_i provide about y. The goal of this step is to learn (i. e., infer) the dependency relationship between the variables x_i and variable y, which is represented by function f, from the datasets using a machine learning algorithm.

The researcher needs first to identify the nature of the function that they need to identify; that is, whether f is a regression, classification, or forecasting function. Then, they may select one of the standard data analytical methods, such as the ones described in Sec. 3.4. The researcher may apply initially the commonly

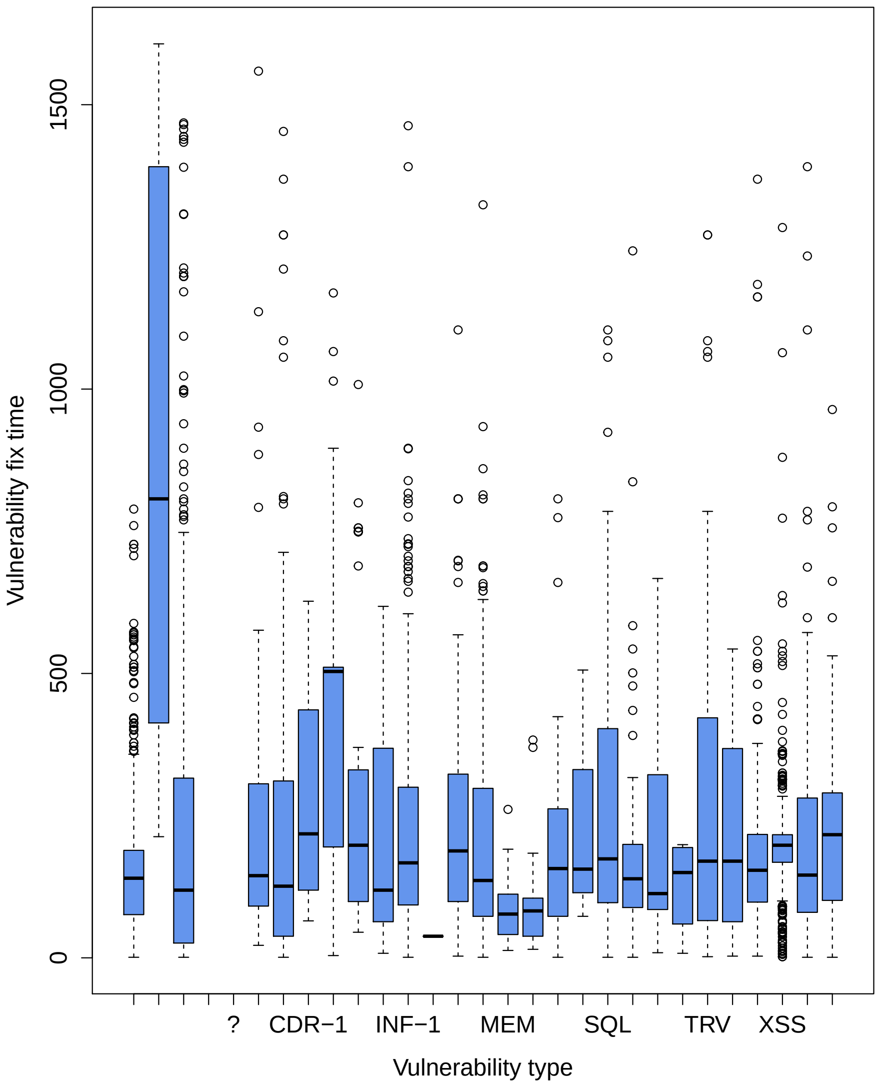

Figure 3.4: Duration by vulnerability.

used analytical methods and explore the possibilities of improving the results through, e. g., using other analytical methods or customizing the used algorithms. The commonly used methods are linear regression for regression problems and support vector machine for classification problems.

The analytical algorithms derive/infer analytical models from given datasets, where the values for both the response variable and the dependent variables are given. An analytical algorithm produces a function that computes the response variable from the dependent variables such that the applied metric (e.g., R^2 for the linear regression method) for the deviation of the computed response values from the real response values is minimized. The developed analytical models are expected to be applied on unknown datasets and thus need to be validated. The simple validation technique, called one round validation, splits the given dataset into a training set and test set. The commonly used ratios of training set to test set are: 3 to 1 and 4 to 1. The analytical model is inferred from the training set and then applied on the test set to compute expected response values. The performance of developed models is measured as the deviation of the expected response values computed using the validation/test data from the related real response values. Sec. 3.5 discusses the commonly used performance metrics.

Validation could be done using a more complicated technique: a k-fold cross-validation. The method requires partitioning the dataset randomly into k equal sized shares. The cross-validation process iterates k times—10 is commonly used for k. In each iteration, $k-1$ shares are used to train the model and the remaining one share is retained for validating the trained model—each of the k shares is used only once as the validation data. The k results of the k iterations can then be averaged to produce a single estimation.

3.3.6 Analyze the Data Analytics Results

Developed analytical models have performance (aka goodness-of-fit). The researcher needs to compare the performance of the generated models to the performance of models developed in similar topics. They need to discuss the performance of their models with the project stakeholders to evaluate the trust on the results and the possibilities to improve them. For instance, the performance metric $PRED(25)$ (see Eq. 3.10 in Sec. 3.5 for its formal definition) that we obtained in a study that we performed on issue fix time estimation was about 35% [2] which is below a $PRED(30)$ of 51 reports by Kultur [27] for project effort estimation in commercial products. Nevertheless, a ratio of 35% (or even 51%) does not encourage companies to use the models (and the techniques) for business purposes.

Analytical models are usually developed to address specific research questions. The results of the secure software analytical projects should be discussed with the stakeholders to interpret them and get their semantics, that is, what do the results mean in practice. The stakeholders may get new insights while discussing the results. For example, when discussing the initial results of the issue fix time project [2], the participants asked about the coefficients of the independent variables used in the linear regression model. The coefficients indicate the contributions of the independent variables to the issue fix time. The analysis of the coefficients showed that the factors components, projects, and development teams have high impact on the issue fix time. The stakeholders have observed the impact of identified factors but did not have evidence, as the one presented, to justify using the information to take decisions. Thus, the discussion allowed to identify new uses of the results.

Analytical models are often developed iteratively. Sharing the results with the stakeholders and discussing them may allow to identify the weaknesses in the work, which could be related to the data or to the used analytical methods. Analysis of the causes of these weaknesses should reveal ways to improve the results. For example, we found when developing a model predicting the issue fix time, as stated above, that the components, projects, and development teams are the main factors that impact the issue fix time [2]. The results encouraged investigating the aspects of these specific factors. The stakeholders provided new datasets related to these factors, which we used to extend the used datasets. The analytical models developed from the extended datasets had sometimes better performance than the basic ones. This activity allowed to identify the aspects that made the factors highly contribute to the issue fix time.

Analysis of the results of the data analytical projects often sparks ideas for new directions in the research. For example, we observed in our fixing effort project (Sec. 3.2.1) that the results are not stable because extending the datasets with newly collected data changes the models (e. g., the coefficients of linear models change). We plotted the tendency of the average issue fix time by month and we observed that the metric is not constant; it has a changing tendency over time. The change is explained by frequent changes to the issue fixing process and by the use of push-sprints. This analysis suggests that a prediction model of issue fix time that consider time evolution should have better performance that the basic prediction model. The observed sequence of publications on vulnerability prediction models [35, 46, 42, 37] is also a result of reflection and analysis of the obtained results.

3.4 Learning Methods Used in Software Security

Many research questions[8] are related to software security study dependencies between all variables of interest using correlation analysis. However, often the need is to predict or forecast response variables (the variables of interest) and not

[8]For example, "How software security metrics obtained from a software component are relevant with the time it takes to compromise the component?" [17].

just to explain them. In these cases learning algorithms of the response variables are used.

Response variables are either quantitative or qualitative. Quantitative variables take numerical values and qualitative variables take values from a finite unordered set. For example, The variable number of days is quantitative while the variable vulnerability type is qualitative. We refer to prediction problems that have quantitative response variables as regression problems and problems that have qualitative response variables as classification problems [22]. When the independent variable is time, the prediction problems is considered as a special case and is called time series forecasting [19].

This section gives an overview of set of regression, classification, and forecasting learning methods commonly used in software security.[9] Each of the method is based on the optimization of a specific metric for measuring errors and may require satisfying specific assumptions. Tab 3.1 summarizes these methods and gives examples of studies that applied each of them.

Table 3.1: Selected set of machine learning methods.

Response variable	**Learn. Type**	**Algorithm**	**Example**
Categorical	Classification	Logistic regression	[41, 42, 7]
		Bayes	[45, 28, 37, 7]
		Support vector machine (SVM)	[34, 45, 28, 37]
		Decision-tree classification	[45, 37, 7]
Continuous	Regression	Linear regression	[35, 2, 8, 50]
		Tree-based regression	[11, 2, 12]
		Neural-networks regression	[2]
Continuous	Forecasting	Exponential smoothing	[36]
		Autoreg. integrated moving avg.	[36, 23]

3.4.1 Classification Methods

Classifying observations is assigning the observations to categories (aka classes) based on prediction of the values of the response variable [22]. There are several methods that could be used for classification. We give in the following an overview of some of the commonly used methods.

[9]Readers interested in machine learning methods and techniques may consult for example [15, 22].

Naïve Bayes. This classifier is a probabilistic classifiersbased on the Bayes theorem:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{3.1}$$

where $P(A)$ and $P(B)$ are the probabilities of observing the event A and B independently; $P(A|B)$ (respectively $P(B|A)$) is the conditional probability of observing event A (respectively B) given that B (respectively A) is true. Naïve Bayes classifiers are particularly successful when applied to text classification problems, including spam detection.

Logistic regression. This method, called also linear classifier, models the probability that the values of the response variable belong to a given category. This probability is modeled using the logistic function [22]. The method is commonly used, for example, to classify the factors that indicate vulnerable code.

Decision-tree classification. The method is based on segmenting the training observations into a set of partitions using a set of rules expressed based on the independent variables. Each of the partitions is assigned a response value, which is the value that is the most commonly occurring in the partition. The splitting algorithms use metrics that are based on the proportions of observations that are classified in the wrong partitions [22]. Bagging, random forest, and boosting methods extend the decision tree prediction methods by building a set of decisions trees from the training datasets such that together they predict better the response variables.

Support vector machines. The logistic regression and decision-tree classification methods assume a linear decision boundaries on the features, that is, the boundaries between two classes is linear. These methods do not work well for the case of non-linear class boundaries. The SVM idea is to enlarge the features space using specific ways (called kernels [22]) and map the points from the original features space to the enlarged features space. The SVM algorithm learns a hyperplane that can separate the classes in the enlarged features space.

3.4.2 Regression Methods

Regression methods predict quantitative response variables. The most used one is the linear regression method [22]. We give in the following an overview of three of the commonly regression used methods in software security.

Linear regression. This method assumes that the regression function is linear to the input [15]. The method allows for an easy interpretation of the dependencies between input and output variables, as well as predictions of potential future values of the output variables. Most modern regression methods can be perceived as modifications of the linear regression method, being relatively simple [22] and transparent as opposed to its successors. Moreover, understanding and successful usage of such methods as neural networks is nearly impossible without good grasp of the linear regression method [48].

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 prediction error, e. g., the *Residual Sum of Squares* of the partitions [22, 30]. Each partition is assigned a response value, which is the mean of the response values of that partition [22].

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 [43].

3.4.3 Graph Mining

Graph mining methods try to identify graph patterns in complicated structures. In the security domain, they are, for example, popular to analyze social networks [33] or computer programs [49].

Particularly important are techniques for identifying *frequent subgraphs*, i. e., reoccurring sub-patterns within a large graph as well as techniques for identifying *constrained subgraphs*, i. e., sub-patterns that fulfill a specified constraints. A good overview of the various techniques and their implementation is given in [13, Chapter 9].

3.4.4 Forecasting Methods

A time series is a set of numbers that measures a fact over time. Their analysis accounts for the fact that the data points may have an internal structure (such as autocorrelation, trend or seasonal variation) that should be accounted for [24].

Time series data are useful to forecast phenomenon that change over time. The aim of forecasting time series data is to estimate how the sequence of observations will continue into the future. The generated forecasting model uses only information on the variable to be forecasted, and makes no attempt to discover the factors that affect its behavior [19]. We describe in the following the two main forecasting methods: the exponential smoothing method and the Autoregressive Integrated Moving Average (ARIMA) method.

Exponential smoothing. This method uses weighted average of past observations, with the weights decaying exponentially as the observations get older [19], that is, recent observations are given relatively more weight than the older observations. The commonly used method, the Holt-Winters, uses three smoothing parameters [36]:

1. Level - the relative magnitude of the fact,
2. trend - the gradual upward or downward long term movement, and
3. seasonality - short-term variation at the regular intervals.

Autoregressive Integrated Moving Average (ARIMA). The ARIMA[10] method aims to describe the autocorrelations in the data. Unlike the exponential smoothing method, which aims to model time series data that have trend and seasonality, ARIMA models stationary time series data [19]. The ARIMA method uses the following three parameters:

1. the number of Autoregressive (AR) terms - the number of preceding (in time) data points),
2. the differencing - the type of adjustment to make the data stationary, e. g., remove trend or seasonality, and
3. the number of Moving Average (MA) terms - the number of preceding prediction errors.

3.5 Evaluation of Models Performance

The measures of evaluating the performance of analytical models (aka accuracy of models) are different from categorical and continuous response-variables. We discuss in the following the metrics commonly used for both response-variable categories.

3.5.1 Performance Metrics: Categorical-Response-Variables

This subsection provides the common metrics used to evaluate the performance of analytical models when the response variable is categorical. All the performance measures can be calculated from the confusion matrix. The confusion matrix, as seen in Tab 3.2, provides four basic metrics, which are: true positives (TP), false positives (FN), true negatives (TN), and false negatives (FN). The description of the metrics follow.

[10]This method is also called Box-Jenkins method.

Table 3.2: Confusion matrix for two-class outcome variables.

		Predicted	
		Positive	Negative
Actual	True	*TP*	*FN*
	False	*FP*	*TN*

Accuracy. This metric measures the fraction of correctly classified cases [32]. A perfect prediction model has accuracy 1. The formula for computing the metric is:

$$ACC = \frac{TP + TN}{TP + TN + FN + FP} \tag{3.2}$$

This metric should be interpreted carefully, as it could be misleading [32].

Hit rate (aka Recall). This metric measures the success rate to predict correctly positive cases. A perfect prediction model has Recall of 1. The formula for the metric is:

$$REC = \frac{TP}{TP + FN} \tag{3.3}$$

Precision. This metric measures the rate of success to predict positive cases. A perfect prediction model has Precision of 1. The formula for the metric is:

$$PREC = \frac{TP}{TP + FP} \tag{3.4}$$

False alarm rate. This metric measures the rate of incorrect prediction of positive cases. A perfect prediction model has false alarm rate of 0. The formula for the metric is:

$$FPR = \frac{FP}{TP + FP} \tag{3.5}$$

F-measure. This metric measures the weighted harmonic mean of recall and precision. The formula for the metric is:

$$F\text{-}measure = 2 \cdot \frac{PREC \cdot REC}{PREC + REC} \tag{3.6}$$

Receiver Operating Characteristics (ROC) Curve. The ROC Curve [32] plots the hit rate on the *y* axis against the false alarm rate on the *x* axis. A good classification model has a high hit rate and a low false alarm rate, which would be visualized with an ROC curve closer to the upper left corner. The Curve allows to compare generated prediction models. In addition, it allows to compare prediction models to the random prediction model–i.e., where the hit rate is equal to the false alarm rate [9].

3.5.2 Performance Metrics: Continuous Response-Variables

Several metrics have been developed to compare the performance of the prediction and forecast models. These metrics indicate how well the models predict/forecast accurate responses for future inputs. We describe in the following 4 metrics that are commonly used in software security.

Coefficient of determination (R^2). This metric "summarizes" how well the generated regression model fits the data. 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 [19]. A value such as 0.5 indicates that about half of the variation in the data can be predicted or explained using the model [19]. A value 1 of this metric indicates that the model perfectly fits the data and value 0 indicates that the model does not explain the data. The following equation formulates the metric.

$$R^2 = 1 - \frac{\sum\limits_{i=0}^{n} (x_i - \hat{x}_i)^2}{\sum\limits_{i=0}^{n} (x_i - \bar{x})^2} \tag{3.7}$$

where n is the number of observations, x_i is the actual value for observation i, $\hat{x}_i$ is the estimated value for observation i, and $\bar{x}$ is the mean of x_i values.

The Linear Regression method focuses on minimizing R^2. The metric is used to evaluate the performance of linear regression models. Thus, it may not be a good metric to evaluate non-linear models [44] since they do not attempt to optimize it too.

Mean Magnitude of relative Error (MMRE). This metric measures the mean of the errors ratio between the predicted/forecasted values and their corresponding actual values [25, 26]. The following equation formulates the metric.

$$MMRE = \frac{\sum\limits_{k=0}^{n} \frac{|\hat{x}_i - x_i|}{x_i}}{n} \tag{3.8}$$

where n is the number of observations, x_i is the actual value for observation i, $\hat{x}_i$ is the estimated value for observation i.

Akaike's Information Criterion. This metric estimates the information loss when approximating reality. The following equation formulates the metric [19].

$$AIC = n \cdot \log\left(\sum_{k=0}^{n} \frac{(x_i - \hat{x}_i)^2}{n}\right) + 2 \cdot (k+2) \tag{3.9}$$

where n is the number of observations, x_i is the actual value for observation i, $\hat{x}_i$ is the estimated value for observation i, and k is the number of variables.

A smaller AIC value indicates a better model.

PRED. This metric computes the percentage of prediction falling within a threshold h [25, 26]. The following equation formulates the metric

$$PRED(h) = \frac{100}{n} \cdot \sum_{i=1}^{n} \begin{cases} 1 & \text{if } \frac{x_i - \hat{x}_i}{x_i} \leq h, \\ 0 & \text{otherwise.} \end{cases} \tag{3.10}$$

Here n is the number of observations, x_i is the actual value for observation i, $\hat{x}_i$ is the estimated value for observation i, and h is the threshold, e. g., 25%.

The perfect value for the *PRED* metric is 100%.

3.6 More Lessons Learned

Our experience shows that one of the biggest challenges of empirical research applied to software security is the availability and quality of the data that can be used for hypothesis evaluation. This *ground truth* data is often incomplete. For example, when we tried to understand, how the number of published security vulnerabilities for an open source project is related to its other characteristics, we understood that for many similar projects there is not enough information about vulnerabilities (sometimes, it does not even exist). That means, for example, that a time-series model for predicting the disclosure time of security vulnerabilities trained on the set of Apache projects may not be adequate for other projects, because Apache Foundation may be more dedicated to systematic vulnerability disclosure than other projects.

Therefore, choosing the right source of information is critical. For instance, Massacci and Nguyen [29] addressed the question of selecting adequate sources of *ground truth* data for vulnerability discovery models, showing problems of various vulnerability data sources, and that the set of features that can be used for vulnerability prediction is scattered over all of them, discouraging researchers in relying on a single source of such information.

Another example of data deficiency in empirical security research is the question "Is open source software more/less secure than proprietary software?" While there exist numerous studies on the matter [14, 16, 40, 38, 39], we believe it is unlikely that this question will get an exhaustive answer. The reason for that is that independent security researchers will unlikely get full access to the data that correspond to the whole population of proprietary software, or (at least) to the similar extent with open source software.

Working with large data sets can be also very challenging for empirical researchers. However, while tasks such as data exploration/cleaning of analysis cannot be completely automated, we fully appreciate the value of automation in repetitive tasks such as data collection, which allows to relocate significant amounts of time to the actual analysis.

3.7 Conclusion

While applying data analytics for improving software security has already proven to be useful, it is a very young discipline and we expect much more improvements and success stories. We identified, as a particular challenges that needs to be addressed, the lack of data of the necessary quality. Thus, we can only repeat our call for collecting high-quality data in any security engineering project and to use data analytics to monitor and improve secure software engineering activities.

References

[1] Ruediger Bachmann and Achim D. Brucker. Developing secure software: A holistic approach to security testing. *Datenschutz und Datensicherheit (DuD)*, 38(4):257–261, apr 2014.

[2] Lotfi ben Othmane, Golriz Chehrazi, Eric Bodden, Petar Tsalovski, and Achim D. Brucker. Time for addressing software security issues: Prediction models and impacting factors. *Data Science and Engineering (DSEJ)*, 2016.

[3] Lotfi ben Othmane, Golriz Chehrazi, Eric Bodden, Petar Tsalovski, Achim D. Brucker, and Philip Miseldine. Factors impacting the effort required to fix security vulnerabilities: An industrial case study. In Colin Boyd and Danilo Gligoriski, editors, *Information Security Conference (ISC) 2015)*, Lecture Notes in Computer Science. Springer-Verlag, 2015.

[4] Ayse Bener, Ayse Tosun Misirli, Bora Caglayan, Ekrem Kocaguneli, and Gul Calikli. Lessons learned from software analytics in practice. In Christian Bird, Tim Menzies, and Thomas Zimmermann, editors, *The Art and Science of Analyzing Software Data*, pages 453–489, Waltham, USA, Aug. 2015. Elsevier.

[5] Amiangshu Bosu, Jeffrey C. Carver, Munawar Hafiz, Patrick Hilley, and Derek Janni. Identifying the characteristics of vulnerable code changes: An empirical study. In *Proc. of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 257–268, 2014.

[6] Achim D. Brucker and Uwe Sodan. Deploying static application security testing on a large scale. In Stefan Katzenbeisser, Volkmar Lotz, and Edgar Weippl, editors, *GI*

Sicherheit 2014, volume 228 of *Lecture Notes in Informatics*, pages 91–101. GI, March 2014.

[7] Istehad Chowdhury and Mohammad Zulkernine. Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. *Journal of Systems Architecture*, 57(3):294–313, 2011. Special Issue on Security and Dependability Assurance of Software Architectures.

[8] Stanislav Dashevskyi, Achim D. Brucker, and Fabio Massacci. On the security cost of using a free and open source component in a proprietary product. In Juan Caballero and Eric Bodden, editors, *International Symposium on Engineering Secure Software and Systems (ESSoS)*, number 9639 in Lecture Notes in Computer Science, pages 190–206. Springer-Verlag, 2016.

[9] Tom Fawcett. An introduction to roc analysis. *Pattern Recogn. Lett.*, 27(8):861–874, June 2006.

[10] Harald Gall, Tim Menzies, Laurie Williams, and Thomas Zimmermann. Software Development Analytics (Dagstuhl Seminar 14261). *Dagstuhl Reports*, 4(6):64–83, 2014.

[11] Michael Gegick, Pete Rotella, and Laurie Williams. Toward non-security failures as a predictor of security faults and failures. In *International Symposium on Engineering Secure Software and Systems*, pages 135–149. Springer, 2009.

[12] Michael Gegick, Laurie Williams, Jason Osborne, and Mladen Vouk. Prioritizing software security fortification throughcode-level metrics. In *Proceedings of the 4th ACM workshop on Quality of protection*, pages 31–38. ACM, 2008.

[13] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

[14] Marit Hansen, Kristian Köhntopp, and Andreas Pfitzmann. The Open Source approach: opportunities and limitations with respect to security and privacy. *Computers & Security Journal*, 21(5):461–471, 2002.

[15] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2013.

[16] Jaap-Henk Hoepman and Bart Jacobs. Increased security through open source. *Communications of the ACM*, 50(1):79–83, 2007.

[17] Hannes Holm, Mathias Ekstedt, and Dennis Andersson. Empirical analysis of system-level vulnerability metrics through actual attacks. *IEEE Transactions on dependable and secure computing*, 9(6):825–837, 2012.

[18] Michael Howard and Steve Lipner. *The Security Development Lifecycle*. Microsoft Press, Redmond, WA, USA, 2006.

[19] Rob J. Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. Otexts, 2014.

[20] Jay Jacobs and Bob Rudis. *Data-Driven Security: Analysis, Visualization and Dashboards*. Wiley Publishing, 2014.

[21] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning with Applications in R*. Springer-Verlag, New York, US, 2013.

[22] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning with Applications in R*. Springer, 2014. Springer Texts in Statistics.

[23] Pontus Johnson, Dan Gorton, Robert Lagerström, and Mathias Ekstedt. Time between vulnerability disclosures: A measure of software product vulnerability. *Computers & Security*, 62:278–295, 2016.

[24] Myungsook Klassen. Investigation of some technical indexes instock forecasting using neural networks. *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 1(5):1438–1442, 2007.

[25] Ekrem Kocaguneli, Tim Menzies, and Jacky Keung. On the value of ensemble effort estimation. *IEEE Transactions on Software Engineering*, 38(6):1403–1416, Nov 2012.

[26] Ekrem Kocaguneli, Tim Menzies, and Emilia Mendes. Transfer learning in effort estimation. *Empirical Software Engineering*, 20(3):813–843, June 2015.

[27] Yigit Kultur, Burak Turhan, and Ayse Bener. Ensemble of neural networks with associative memory (enna) for estimating software development costs. *Knowledge-Based Systems*, 22(6):395–402, 2009.

[28] Zhenmin Li, Lin Tan, Xuanhui Wang, Shan Lu, Yuanyuan Zhou, and Chengxiang Zhai. Have things changed now? an empirical study of bug characteristics in modern open source software. In *Proceedings of the 1st workshop on Architectural and system support for improving software dependability*, pages 25–33. ACM, 2006.

[29] Fabio Massacci and Viet Hung Nguyen. An empirical methodology to evaluate vulnerability discovery models. *Software Engineering, IEEE Transactions on*, 40(12):1147–1162, 2014.

[30] Tim Menzies. Data mining: A tutorial. In Martin P. Robillard, Walid Maalej, Robert J. Walker, and Thomas Zimmermann, editors, *Recommendation Systems in Software Engineering*, pages 39–75. Springer Berlin Heidelberg, 12 2013.

[31] Tim Menzies, Rahul Krishna, and David Pryor. The promise repository of empirical software engineering data. `http://openscience.us/repo`, 2015. North Carolina State University, Department of Computer Science.

[32] Charles E. Metz. Basic principles of ROC analysis. *Seminars in Nuclear Medicine*, 8(4):283–298, 1978.

[33] Sudip Mittal, Prajit Kumar Das, Varish Mulwad, Anupam Joshi, and Tim Finin. CyberTwitter: Using twitter to generate alerts for cybersecurity threats and vulnerabilities. In *International Symposium on Foundations of Open Source Intelligence and Security Informatics*. IEEE Computer Society, August 2016.

[34] Stephan Neuhaus, Thomas Zimmermann, Christian Holler, and Andreas Zeller. Predicting vulnerable software components. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 529–540, 2007.

[35] Viet Hung Nguyen and Le Minh Sang Tran. Predicting vulnerable software components with dependency graphs. In *Proceedings of the 6th International Workshop on Security Measurements and Metrics*, page 3. ACM, 2010.

[36] Yaman Roumania, Joseph K. Nwankpab, and Yazan F. Roumani. Time series modeling of vulnerabilities. *Computers & Security*, 51:32–40, June 2015.

[37] Riccardo Scandariato, James Walden, Aram Hovsepyan, and Wouter Joosen. Predicting vulnerable software components via text mining. *Software Engineering, IEEE Transactions on*, 40(10):993–1006, 2014.

[38] Guido Schryen. Security of open source and closed source software: An empirical comparison of published vulnerabilities. In *Americas Conference on Information Systems (AMCIS)*, 2009.

[39] Guido Schryen. Is open source security a myth? *Communications of the ACM*, 54(5):130–140, 2011.

[40] Guido Schryen and Rouven Kadura. Open source vs. closed source software: towards measuring security. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 2016–2023. ACM, 2009.

[41] Yonghee Shin, A. Meneely, L. Williams, and J.A. Osborne. Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. *Software Engineering, IEEE Transactions on*, 37(6):772–787, Nov 2011.

[42] Yonghee Shin and Laurie Williams. An empirical model to predict security vulnerabilities using code complexity metrics. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 315–317. ACM, 2008.

[43] Donald F. Specht. A general regression neural network. *Neural Networks, IEEE Transactions on*, 2(6):568–576, Nov 1991.

[44] Andrej-Nikolai N. Spiess and Natalie Neumeyer. An evaluation of R2 as an inadequate measure for nonlinear models in pharmacological and biochemical research: a Monte Carlo approach. *BMC pharmacology*, 10(1):6+, June 2010.

[45] Lin Tan, Chen Liu, Zhenmin Li, Xuanhui Wang, Yuanyuan Zhou, and Chengxiang Zhai. Bug characteristics in open source software. *Empirical Software Engineering*, 19(6):1665–1705, 2014.

[46] James Walden and Maureen Doyle. Savi: Static-analysis vulnerability indicator. *Security & Privacy Journal, IEEE*, 10(3):32–39, 2012.

[47] Ronald E. Walpole, Raymond H. Myers, Sharon L. Myers, and Keying Ye. *Probability & statistics for engineers and scientists*. Pearson Education, Upper Saddle River, 2007.

[48] Sanford Weisberg. *Applied linear regression*, volume 528. John Wiley & Sons, 2005.

[49] Fabian Yamaguchi, Nico Golde, Daniel Arp, and Konrad Rieck. Modeling and discovering vulnerabilities with code property graphs. In *Symposium on Security and Privacy (SP)*, pages 590–604. IEEE Computer Society, 2014.

[50] Su Zhang, Doina Caragea, and Xinming Ou. An empirical study on using the national vulnerability database to predict software vulnerabilities. In *International Conference on Database and Expert Systems Applications*, pages 217–231. Springer, 2011.