Autom Softw Eng (2015) 22:3–46
DOI 10.1007/s10515-014-0155-1

AutoODC: Automated generation of orthogonal defect classifications

LiGuo Huang · Vincent Ng · Isaac Persing · Mingrui Chen · Zeheng Li · Ruili Geng · Jeff Tian

Received: 8 October 2013 / Accepted: 3 May 2014 / Published online: 3 June 2014

Abstract Orthogonal defect classification (ODC), the most influential framework for software defect classification and analysis, provides valuable in-process feedback to system development and maintenance. Conducting ODC classification on existing organizational defect reports is human-intensive and requires experts' knowledge of both ODC and system domains. This paper presents AutoODC, an approach for automating ODC classification by casting it as a supervised text classification problem. Rather than merely applying the standard machine learning framework to this task, we seek to acquire a better ODC classification system by integrating experts' ODC experience and domain knowledge into the learning process via proposing a novel relevance annotation framework. We have trained AutoODC using two state-of-the-art machine learning algorithms for text classification, Naive Bayes (NB) and support vec-

L. Huang · M. Chen · Z. Li (✉) · R. Geng · J. Tian
Southern Methodist University, Dallas, TX, USA
e-mail: zehengl@smu.edu

L. Huang
e-mail: lghuang@smu.edu

M. Chen
e-mail: mingruic@smu.edu

R. Geng
e-mail: rgeng@smu.edu

J. Tian
e-mail: tian@smu.edu

V. Ng · I. Persing
University of Texas at Dallas, Richardson, TX, USA
e-mail: vince@hlt.utdallas.edu

I. Persing
e-mail: persingq@hlt.utdallas.edu

tor machine (SVM), and evaluated it on both an industrial defect report from the social network domain and a larger defect list extracted from a publicly accessible defect tracker of the open source system FileZilla. AutoODC is a promising approach: not only does it leverage minimal human effort beyond the human annotations typically required by standard machine learning approaches, but it achieves overall accuracies of 82.9 % (NB) and 80.7 % (SVM) on the industrial defect report, and accuracies of 77.5 % (NB) and 75.2 % (SVM) on the larger, more diversified open source defect list.

Keywords Orthogonal defect classification (ODC) · Machine learning · Natural language processing · Text classification

1 Introduction

The systematic classification and analysis of defect data bridge the gap between causal analysis and statistical quality control, provide valuable in-process feedback to the system development or maintenance, as well as help assure and improve system and software quality. The analysis results based on systematic defect classifications enable us to understand the major impact types of system defects and to pinpoint specific problematic areas for focused problem resolution and quality improvement. Orthogonal defect classification (ODC), developed initially at IBM, is the most influential among general frameworks for software defect classification and analysis. Ram Chillarege and his colleagues proposed and elaborated the taxonomy of ODC framework for IBM systems Chillarege (1995) and have used ODC to classify defects to support in-process feedback to developers by extracting signatures on the development process from defects (Chillarege et al. 1992). ODC has been successfully used in various types of industrial and government organizations, to identify problematic areas, and to improve software product quality. ODC has been used by both small and large organizations including NASA, IBM, Motorola, and Cisco. It has also been engaged with a broad swath of industrial and consumer applications such as embedded systems, hand-held devices, and business applications. In recent years, this ODC taxonomy has been adapted and customized to various software system domains. Lutz and Mikulski (2004a,b) presented a more fine-grained ODC taxonomy (particularly for the "activity" attribute) adapted to safety-critical operation anomalies of NASA spacecraft systems. Bridge and Miller reported their experience of using a subset of ODC "impact" categories to classify all the post-release software defects since 1996 in Motorola GSM Product Divisions Base Station Systems software development group (Bridge and Miller 1998). Zheng et al. (2006) adapted ODC for the static analysis of software fault detection by associating ODC defect types with the development process elements.

Existing defect classification approaches and processes are human intensive, requiring significant knowledge of both historical projects and ODC taxonomy from domain experts. At IBM, ODC has been used to drive Software Process Improvement initiatives. Once defects are found, developers directly classify them into the ODC taxonomy for further defect analysis (Mays et al. 1990). Existing approaches require the human analyst(s) to assimilate each defect description in order to classify it into the ODC taxonomy. AutoODC is developed to automatically generate ODC defect classifications from existing defect repositories with a defect's summary and description in a specific software domain. In most cases, defect data are reported by users or developers during system development, operation, and maintenance and are stored in the defect repositories with a customized classification scheme for an industrial organization or without being classified. These defect repositories are often called defect (bug) reports, defect (bug) trackers, or issue lists. Analysts then need to read and understand the textual description of each reported defect in order to label it with *exactly one* defect category defined in the ODC taxonomy. The ODC defect classifications discussed in Bridge and Miller (1998), Lutz and Mikulski (2004a,b), and Zheng et al. (2006) were all performed based on existing defect repositories. Manual ODC defect classification based on existing defect repositories is at best a mundane, mind numbing activity, as anyone who has spent any time performing this activity will tell you. Due to the huge amount of manual effort required for ODC generation, the types of follow-on ODC-based defect analyses are often restricted by the limited defect classification results. For instance, the limited analyst resources in an organization may restrict the number of ODC attributes on which the defect classification can be performed so that multi-way defect analyses may not be supported.

To overcome these challenges in software system defect classification, we have developed AutoODC, an approach for automatically generating Orthogonal Defect Classifications (ODCs) from defect reports (tracker). This paper presents our research results on improving the state of the art in ODC generation. We cast the problem as a *supervised text classification* problem, where the goal is to train a classifier via machine learning techniques for automatically classifying a defect *record* in a defect report. While one may expect that classification accuracy consistently increases with the amount of available training data, somewhat surprisingly we observed that this was not the case for our classification problem. We hypothesize that the reason could be attributed to *the learning algorithm's inability to learn from the relevant portions of a defect record*. Specifically, only a subset of the words appearing in a record may be relevant for classification, and hence learning from all of the words in the report may introduce noise into the learning process and eventually deteriorate the performance of the resulting classification system. Consequently, we propose a new framework in which the learning algorithm learns from a richer kind of training data. Rather than simply determining the category to which a defect record belongs, a human annotator additionally annotates the segments in the report (e.g., words, phrases) that indicate *why* she believes the report should belong to a particular category. To further our attempt to enable a learning algorithm to learn from only the relevant materials, we complement our annotation relevance framework with a *discourse analysis component*, where we perform shallow discourse analysis to *heuristically* identify and remove text segments from a defect record that are *unlikely* to be relevant for classification. We hypothesize that employing both the annotation relevance framework and the discourse analysis component can offer complementary benefits: while the annotation relevance framework employs *machine learning* to identify the portions of a defect record that are *relevant* to classification, the discourse analysis component aims to *heuristically* identify the portions of a defect record that are likely to be *irrelevant* to classification. Our experiments demonstrate that employing our proposed *annotation relevance* framework in combination with our shallow *discourse analysis* component can improve the performance of our defect classifier in comparison to the standard learning-based text classification framework, with relative error reductions of 14–29 %. Note that having a human annotator to provide these additional relevant annotations does not significantly increase her annotation burden: since she has to read the entire defect record to identify its category in order to develop the training set, the only additional step she has to take is to mark up those text segments responsible for her classification.

We have applied AutoODC to two different defect datasets. One contains 403 defect records (reported issues) in an industrial defect report from Company P[1]; the other contains 1,250 defect records (reported issues) from the publicly available defect tracker (issue list) of the open source system FileZilla. For the experiments on both datasets, we have compared the performance of AutoODC with manual ODC classifications on the "impact" attribute.[2] Defect impact is an ODC attribute used to show the impact type (detailed in Sect. 2) the defect would have had upon the customer and end users if it had escaped to the field. Our evaluation was performed on the "impact" attribute because Company P was more concerned about customer satisfaction with its social networking products and similarly the defects in the open source system FileZilla were mostly reported by users. This allows quality improvement efforts to be geared towards reducing the defects that most significantly impact customer/user satisfaction as opposed to blindly reducing the total number of defects. We evaluated AutoODC by calculating the overall accuracy. Moreover, to measure the performance on each defect type, we employed two commonly-used metrics: the percentage of correct defect classifications that are identified (recall) and the percentage of correct classifications as a ratio to the total number of examined defects (precision). In summary, we make the following contributions:

- We introduce the *annotation relevance framework*, a novel framework that enables the acquisition of a defect classifier from a richer kind of training data by injecting experts' domain knowledge into the classifier so as to robustly analyze natural language defect descriptions.
- To further our attempt to enable a learning algorithm to learn from the relevant portions of a defect record, we complement our annotation relevance framework with a *discourse analysis component*, where we employ shallow linguistic analysis to *heuristically* identify and remove materials that are *unlikely* to be relevant for classification.
- We evaluate AutoODC using two state-of-the-art machine learning algorithms for text classification, Naive Bayes (NB) and support vector machine (SVM) on two evaluation datasets, one containing 403 defect records and the other containing 1,250 defect records. Not only do the datasets differ in size, but they also differ in terms of class distributions, with the larger dataset having a more balanced class distribution than the smaller one. AutoODC produces satisfactory classification

[1] Due to proprietary rules, we anonymize the industrial company by referring to it as "Company P" throughout this paper.

[2] The definitions and taxonomy of ODC v5.2 attributes are accessible at http://researcher.watson.ibm.com/researcher/files/us-pasanth/ODC-5-2.pdf.

results with accuracies of 80.7–82.9 % on the small dataset and 75.2–77.5 % on the large dataset when using manual defect classification as a basis of evaluation, where accuracy is computed as the percentage of defect records correctly classified by our classification system. These results suggest that AutoODC offers robust performance regardless of whether the underlying class distribution of the evaluation dataset is skewed or relatively balanced.
- To gain insights into the weaknesses of AutoODC and provide directions for future work, we perform an analysis of the major types of errors it makes.
- As a complement to the manual ODC classification, AutoODC improves the confidence of defect classification results by reducing the investigation set that a human analyst has to examine when performing ODC classification.

AutoODC has several implications for the software engineering research and practice community. First, AutoODC launches an initial step for applying and tailoring the natural language processing (NLP) text classification methodology into a practical software engineering problem faced by both academia and industry (i.e., ODC generation). Second, a key advantage of our framework is its generality: it is by no means specific to defect classification and can be readily applied to any supervised text classification task in the software engineering community with a bit tailoring.

The rest of this paper is organized as follows. Section 2 presents the motivating examples for AutoODC defect classification. Section 3 discusses related work. Section 4 presents our approach and Sect. 5 describes our evaluation methods and results. Section 6 discusses the current limitations and threats to validity of AutoODC. Section 7 concludes with a discussion of future work.

2 Motivating example

To illustrate the key features of a defect report or issue list, Table 1 provides two simple examples extracted from the defect report (issue list) from an open source defect (issue) tracker of Elgg.[3] Although an industrial defect report or open source issue list may contain a set of fields such as “ID”, “summary”, “description”, “priority”, “severity”, and “status”, analysts primarily rely on the “summary” and ”description” for ODC classifications. Table 1 shows an abridged defect record (issue) reported in Elgg’s defect tracker. A row in the table is a *defect record* to be classified. The “ID” field is only used for uniquely identifying a defect but does not contribute to defect understanding, and the “summary” field could be optional, in which case the classification is entirely based on the “description”.

The analyst is asked to classify each defect record into an ODC category under the “impact” attribute. Based on the generic ODC classification taxonomy initially proposed by IBM, the “impact” of a defect to customers and users can be further clas-

[3] Elgg is an open source social networking engine. The defect (issue) tracker of Elgg can be accessed at https://github.com/Elgg/Elgg/issues.

sified into one of the following 14 categories including *Installability, Integrity/Security, Capability, Requirements, Reliability, Usability, Performance, Maintenance, Serviceability, Migration, Documentation, Standards, Accessibility, Others*.

Here is an example scenario of ODC defect classification from Motorola's GSM Products Division. In their GSM base station systems (GSMBSS) software development group, five experienced development engineers who are familiar with both project development and ODC were assigned to classify all post-release defects since 1996 by defect "impact" attribute. It took them approximately 6 min to classify each defect (Bridge and Miller 1998). Considering a 1,000 defect dataset, it would take approximately 2.5 weeks for 5 experienced engineers to complete the classification only for one single ODC attribute. Imagine the extra learning curve it would incur for the novice who just takes over the ODC task and does not have sufficient project background. The huge amount of manual effort and human resources demanded by ODC classification prevents the organization from performing the comprehensive ODC classifications in order to enable ODC-based one-way, two-way and multi-way defect analyses as also mentioned earlier.

One major innovation of AutoODC is its use of relevant annotations to improve automated defect classification. Informally, a relevant annotation is a text segment (typically a word or phrase) that an ODC expert analyst relies on to determine the ODC classification of a defect record. To better understand why relevant annotations can potentially help improve classification accuracy, let us consider two motivating examples, both of which are defect records (issues) taken from the defect (issue) tracker of Elgg.

Example 1 The *Requirements* category indicates a customer expectation, with regard to capability, which was not known, understood, or prioritized as a requirement for the current product or release. This value should be chosen during development for additions to the plan of record. It should also be selected, after a product is made generally available, when customers report that the capability of the function or product does not meet their expectation. On the other hand, the *Capability* category defines the ability of the software to perform its intended functions, and satisfy *known* requirements, where the customer is not impacted in any of other ODC impact categories. Defect 1 in Table 1 is a report which belongs to the *Requirements* category. The phrase "would be" in the last sentence is a reasonably strong indicator of the *Requirements* category (e.g., "would be lovely", "would be great", "would be cool"), which implies a desired requirement currently missing from the system. However, a defect dataset typically has a skewed class distribution, where approximately 70 % of the defect records belong to the *Capability* category. Because of this imbalanced class distribution, a learning algorithm would tend to assign documents to this most frequent category if the evidence it is provided is insufficient to assign it elsewhere (Chawla et al. 2004). In this example, even though the phrase "would be" is a piece of useful evidence, the fact that this defect description is long lessens the impact of this evidence, and can potentially cause this report to be misclassified. On the other hand, employing "would be" as a relevant annotation highlights its importance, potentially allowing the learning algorithm to focus on this piece of evidence in spite of the long report.

Table 1 An example of an abridged defect report

ID	Summary	Description	Relevant words/Phrases
1	Prevent (last) Admin User from self-destruction	My client somehow managed to delete himself, meaning that no admin users continued to exist, so no new accounts—admin or otherwise—could be created, and no-one could log into the site! Fortunatley the site was in its infancy, so a simple database restore got it up and running again. But imagine the situtaion if a larger/active site had not been backed up and all the admins had been deleted :-(In principle this shouldn't be possible, but is in fact permitted by the current Elgg core implementation, which provides no "guard code" to ensure that an admin user cannot self-delete, self-ban or self-remove admin. The attached code diffs provide that necessary functionality, and it would be very useful to see these appear in the forthcoming Elgg 1.7 release, so that the update won't overwrite the code changes I have made to my client sites.	Provide, necessary functionality, would be useful
2		Getting 404 after "system settings" this is a fresh install of elgg and after I hit the "save" button on the "system settings" page I get 404 with the message "Oops! This link appears broken." The link in the address bar points to "http://cricmate.com/action/systemsettings/install" Can someone please help me out there? Thanks	Getting 404, link appears broken

Example 2 However, not all mistakes made by a classifier result from mistakenly predicting a class more frequent than a defect's actual class. Consider Defect 2 in Table 1. Though the presence of the word "install" in this example is strong evidence that it may belong to the *Installability* category, it is actually an example of *Reliability* which occurs more frequently than the *Installability* class in our defect records. *Installability* is defined as the ability of the customer to prepare and place the software in position for use which does not include *Usability*, while *Reliability* defines the ability of the software to consistently perform its intended function without unplanned interruption. Severe interruptions, such as ABEND and WAIT would always be considered reliability.

Again, we may be able to exploit relevant annotations to train a classifier that avoids this mistake. For example, our human annotator marked the phrase "link appears broken" as evidence that this defect is an example of *Reliability*. In fact, the human annotator identified five times as many unique phrases for the *Reliability* category containing the word "link" than she did for the *Installability* category. Hence, a learning algorithm may profitably exploit this piece of information to reduce, if not eliminate, the confusion between *Installability* and *Reliability*.

3 Related work

3.1 ODC defect classification

At IBM, ODC has been used to drive Software Process Improvement initiatives. Once defects are found, developers directly classify them into the ODC taxonomy for further defect analysis (Mays et al. 1990). The ODC defect classifications discussed in Bridge and Miller (1998), Lutz and Mikulski (2004a,b), and Zheng et al. (2006) were all performed based on existing defect repositories described in Sect. 1. Existing approaches require the human analyst(s) to assimilate each defect description in order to classify it into the ODC taxonomy. AutoODC is developed to automatically generate ODC defect classifications from the existing defect repositories with the defect summary and description in a specific software domain.

3.2 ODC-based defect analysis

After defects are classified into the ODC taxonomy, statistical analysis can be performed to predict the reliability of a software product by estimating the number of defects remaining and to help identify risks for software process improvement (Chillarege and Biyani 1994). Ma and Tian (2003, 2007) used ODC defect classifications adapted from web server logs and web errors to identify problematic areas of web application systems to improve reliability. ODC is also widely used as a causal analysis method to identify the root cause of defects and initiate actions to prevent, control or eliminate their sources (Lutz and Mikulski 2004b). All these analyses are done manually.

Menzies et al. (2003) used association rule learners to automatically identify patterns among frequently co-occurring anomalies that have already been classified into Trigger, Activity and Target under ODC framework. This simplified part of the manual ODC analysis process and improves the productivity.

IBM patented the automatic calculation of orthogonal defect classification (ODC) fields (Bellucci and Portaluri 2012) in July 2008 (US 8214798 B2), which was published in 2012. This invention provides a method and a system for ODC analysis in a computing system enabling objective calculation of ODC fields including ODC Target, ODC Defect Type, ODC Qualifier and ODC Source History. ODC Target represents the high level identity of the entity that was fixed. ODC Defect Type represents the actual correction that was made (the type captures the size, scope, and complexity of the fix). ODC Defect Qualifier captures the element of a non-existent, wrong or irrelevant implementation. ODC Source History is the selection which identifies the history of the program code or information which was fixed. One embodiment involves determining a defect in a software application, providing a defect fix to the software application, linking the source code fix to the defect, and automatically performing ODC analysis and calculating ODC information based on calculations against the source code linked to the defect fixed. It automatically calculated ODC fields such as Number of lines inserted (CI), Number of lines modified (CM), Number of lines deleted (CD), Number of changed files (F), Number of components impacted by the changes (NC), File extensions (FE), and Number of lines changed (CC), and subsequently calculated ODC fields based on these measures. For instance, the determination of a defect type is based on the total number of code lines changed (i.e., Assignment: CI = 1; Checking: $CC < 5$; Algorithm: $5 < CC < 25$; Function: $25 < CC < 100$; Interface: $NC > 1$). Instead of calculating the code change metrics, AutoODC automatically interprets the semantics of submitted defect (bug) reports leveraging machine learning and natural language processing techniques in order to classify the reported defects into ODC "Impact" defect categories.

Regardless of whether they use manual or automated approaches, all the above ODC-based defect analyses are conducted posterior to the ODC defect classification. In contrast, AutoODC aims at automating ODC classification.

Thung et al. (2012) applied the SVM multi-class classification algorithm to classify 500 defects to three super-categories of defect types, namely, control and data flow, structural, and non-functional, leveraging both textual and code features. Their approach achieved an average accuracy of 77.8 %. In contrast, AutoODC aims at to automate defect classification based on IBM's ODC Impact attribute, so that we have a different perspective of study. Additionally, AutoODC only uses the textual defect summary and description as the basis of classification. We have evaluated AutoODC in combination with two widely used learning techniques (SVM and NB). Most importantly, we have investigated four extensions on both learning techniques to leverage the the human annotations to improve the accuracy of classification. Finally, we have systematically evaluated AutoODC on an industrial defect report as well as on a larger issue list from the defect tracker of an open source system.

3.3 Text classifications in software engineering

3.3.1 Text classification methods in NLP

The most successful text classification methods developed in the NLP community to date have involved supervised learning. Supervised methods typically involve employing an off-the-shelf learning algorithm, such as NB (Rennie et al. 2003; Romano and Pinzger 1998) and SVMs (Joachims 1998; Tong and Koller 2001; Vapnik 1995) to train a classifier on a set of annotated documents. A document is typically represented as a bag of words that appear in the document. Words can be stemmed, and/or analyzed on different linguistic levels: lexical, morphological, syntactic or semantic (Sebastiani 2005). More complex feature representations (e.g., bigrams (Caropreso et al. 2001), part-of-speech tags (Aizawa 2001; Ko and Myers 2006; Pandita et al. 2013), extracted keywords (Kiekel et al. 2002) can also be used.

One may wonder what the difference between the automatically extracted keywords used in previous work and our manually extracted relevant annotations is as far as text classification is concerned. The difference is that automatically extracted keywords are typically computed based on statistical measures (e.g., how frequently a word appears in a text), and hence the resulting keywords may not correlate at all with the categories we are trying to predict. On the other hand, our relevant annotations are selected based on their relevance to the categories involved in our prediction task, and unlike the automatically extracted keywords, relevant annotations may occur infrequently.

3.3.2 Applications in software engineering

Swigger et al. (2010) applied content analysis and semi-automated text classification methods to the communication data from a global software student project in order to predict team performance based on online transcripts. Ormandjieva et al. (2007) measured text quality, specifically the quality of text specifying software requirements. To do so, they proposed a hierarchical quality model that decomposed text quality into measurable features that could be collected directly from text, and developed a text classification system to automatically assess text quality and ambiguity. Hussain et al. (2007) and Polpinij and Ghose (2008) also studied automating quality assessment process of requirement specifications using text classification. Gegick et al. (2010) differentiated security bug reports (SBRs) from non-security bug reports (NSBRs) using a statistical text classification model trained on natural-language descriptions of labeled bug reports. Other applications of text classification on software engineering problems include assigning bugs and change requests to the right developers (Ahsan et al. 2009; Cubranic and Murphy 2004; Tamrawi et al. 2011), recovering traceability links (Asuncion et al. 2010; Huang et al. 2010) and predicting bug severity (Lamkanf et al. 2010; Menzies and Marcus 2008; Yang et al. 2012). AutoODC, on the other hand, employs a new framework that facilitates the automatic acquisition of a defect impact type classifier from a richer kind of training data, specifically data that is annotated with relevance annotations.

4 Approach

Section 4.1 presents an overview of AutoODC, Sect. 4.2 describes a basic learning-based defect classification system. Section 4.3 shows how to integrate our annotation relevance framework into the basic system to exploit the relevant annotations in order to improve classification accuracy.

4.1 Overview

AutoODC accepts a semi-structured text defect report as shown in Table 1, where each defect record contains a defect “summary” (optional) and a “description” (required) and outputs the ODC classification and confidence of classification for each record. It uses a text classification system based on SVM and NB to be integrated with our annotation relevance framework to improve the accuracy of automated ODC classification. Figure 1 depicts an overview of the AutoODC approach which consists of two major components: (1) basic defect classification system, and (2) annotation relevance framework.

The **basic defect classification system** is developed in three steps: (1) Preprocessing defect report; (2) Learning ODC classifications; and (3) classification. Aiming to improve classification accuracy, an **annotation relevance framework** extends the basic classification system by exploiting relevant annotations in four ways (1) gen-

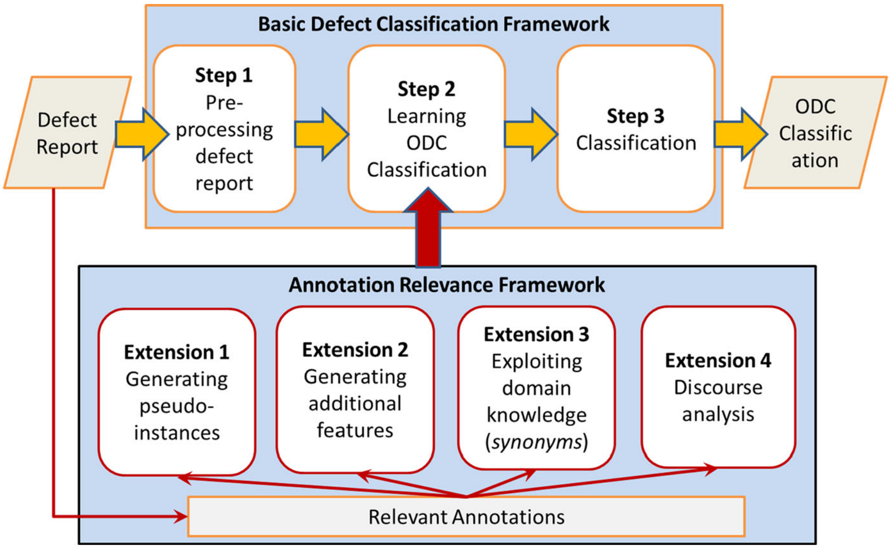

Basic Defect Classification Framework
Defect Report
Step 1 Pre-processing defect report
Step 2 Learning ODC Classification
Step 3 Classification
ODC Classification
Annotation Relevance Framework
Extension 1 Generating pseudo-instances
Extension 2 Generating additional features
Extension 3 Exploiting domain knowledge (synonyms)
Extension 4 Discourse analysis
Relevant Annotations

Fig. 1 An overview of the AutoODC architecture

erating pseudo-instances for training the SVM/NB classifier; (2) generating additional features; (3) adding domain knowledge; and (4) performing discourse analysis.

4.2 The basic defect classification system

Our basic defect classification system has three steps.

Step 1: Pre-processing defect report

As shown in the first three columns of Table 1, the input of AutoODC is a defect report consisting of defect records, each of which contains a textual "description" and an optional "summary". We construct a training set using defect records as follows. First, we preprocess each defect record by tokenizing the record and stemming each word using the stemmer that comes with the WordNet (Fellbaum 1998) lexical knowledge base.[4] We then represent each defect record as a binary-valued vector, where each element corresponds to a distinct word that appears in the training set. Specifically, if the element corresponds to a word that appears in the record, its value is 1; otherwise, its value is 0. In other words, the vector indicates the presence or absence of a word in the corresponding record. Also note that we did not remove stopwords from the vector, as preliminary experiments indicated that results deteriorated slightly with their removal. Also associated with the vector is a class value, which is simply the analyst-assigned ODC category of the corresponding record. Finally, to prevent long defect records, which presumably contain many 1's in their vectors, from having an undesirably large influence on the learning process, we normalize each vector, divid-

[4] Other stemmers, such as the Porter stemmer (Porter 1980), can be used, but we found that the WordNet stemmer yields slightly better accuracy.

Table 2 An Example ODC classification of AutoODC

ID	ODC impact category	Confidence
1	Requirements	0.83
2	Reliability	0.9

ing the value of each feature by a constant so that the length of the resulting vector as measured by its 2-norm is 1.

Step 2: Learning ODC classifications

Now that we have a pre-processed training set consisting of binary-valued vectors, we can apply an off-the-shelf learning algorithm to train a classifier that can be used to classify an unseen record as belonging to one of the pre-defined defect categories. In our experiments, we use a SVM and a NB classifier as the underlying learning algorithms for classifier training. Our choice of SVM and NB is motivated in part by the fact that they have offered impressive performance on a variety of text classification tasks.

We adopt a one-versus-others training scheme, where we train one SVM/NB classifier for each of the classes. For instance, we train one classifier for determining whether a defect record belongs to *Reliability*, another classifier for determining whether a record belongs to *Usability*, etc. In essence, each classifier represents exactly one ODC class, and the number of SVM/NB classifiers we train is equal to the number of defect classes. The training set for training each of these classifiers, of course, is different from each other. Specifically, to train a classifier for determining whether a record belongs to class i, we take the pre-processed training set described above, and assign the class value of each training instance as follows. If the training instance belongs to class i, we assign its class value to +1; otherwise, we change its class value to -1.

Step 3: Classification

After training, we can apply the resulting classifiers to classify each defect record in the test set, where the test instances are created in the same way as the training instances (see Step 1). Specifically, given a test instance (a defect record to be classified), we apply each of the resulting classifiers separately to classify the instance. Each classifier will return a real value. The higher the value is, the more confident the classifier is in its classification of the instance as belonging to the class it represents. As a result, we assign to a defect record the class whose classifier returns the highest value among the set of values returned by all the classifiers. Finally, AutoODC will output the ODC defect category with a confidence measure as shown in Table 2. When SVM classifiers are used, confidence is measured by the data point's distance from the SVM hyperplane either from the "+" or "−" side. When NB classifiers are used, confidence is measured by the probability that the data point belongs to the "+" class.

4.3 Exploiting relevant annotations

Next, we present four extensions to the basic classification framework described above, all of which involve exploiting relevant annotations. Importantly, note that only the defect records in the training set contain the relevant annotations. Our framework does *not* assume any annotations for a record in the test set.

Extension 1: Generating pseudo-instances

Recall that in the basic defect classification system, for each ODC category c we train a classifier that identifies defect records belonging to c and describe how to create the training set for training this classifier. Our first extension involves augmenting this training set with additional positive and negative training instances generated from the relevant annotations. We will refer to these additional training instances as *pseudo-instances*.

Below we describe two methods to generate and exploit the pseudo-instances. The first method involves modifying SVM's learning procedure and therefore is applicable only when SVM is the underlying learning algorithm. The second method, on the other hand, treats a learning algorithm as a black-box procedure and therefore can be applied in combination of any machine learning algorithm.

Method 1: SVM-based pseudo-instance generation

To understand how to modify SVM's learning procedure to take advantage of the relevant annotations, we need to first review SVM's learning procedure.

Assume that we are given a dataset consisting of positively labeled points, having a class value of +1, and negatively labeled points, having a class value of -1. An SVM learning algorithm aims to learn a hyperplane (i.e., a linear classifier) that separates the positive points from the negative points. If there is more than one hyperplane that achieves zero training error (i.e., the hyperplane perfectly separates the points from the two classes), the learning algorithm will choose the hyperplane that maximizes the margin of separation (i.e., the distance between the hyperplane and the training example closest to it), as a larger margin can be proven to provide better generalization (i.e., accuracy) on unseen data (Vapnik 1995). More formally, a maximum margin hyperplane is defined by $w \cdot x - b = 0$, where x is a feature vector representing an arbitrary data point, and w (a weight vector) and b (a scalar) are parameters that are learned by solving the following constrained optimization problem:

Optimization problem 1: Hard-margin SVM

$$\min \frac{1}{2} ||w||^2 \tag{1}$$

subject to

$$y_i(w \cdot x_i - b) \geq 1, \quad 1 \leq i \leq n,$$

where $y_i \in \{+1, -1\}$ is the class of the ith training point x_i. Note that for each data point x_i, there is exactly one linear constraint in this optimization problem that ensures x_i is correctly classified. In particular, using a value of 1 on the right side of each inequality constraint ensures a certain distance (i.e., margin) between each x_i and the hyperplane. It can be shown that the margin is inversely proportional to the length of the weight vector. Hence, minimizing the length of the weight vector is equivalent to maximizing the margin. The resulting SVM classifier is known as a hard-margin SVM: the margin is "hard" because each data point has to be on the correct side of the hyperplane.

However, in cases where the dataset is not linearly separable, there is no hyperplane that can perfectly separate the positives from the negatives, and as a result, the above constrained optimization problem does not have a solution. Instead a relaxed version of the problem is typically solved, where we also consider hyperplanes that produce non-zero training errors (misclassifications) as potential solutions. That is, we have to modify the linear constraints associated with each data point to allow training errors. However, if we only modify the linear constraints but leave the objective function as it is, then the learning algorithm will only search for a maximum-margin hyperplane regardless of the training error it produces. Since training error correlates positively with test error, it is crucial for the objective function to also take into consideration the training error so that a hyperplane with a large margin and a low training error can be found. However, it is non-trivial to maximize the margin and minimize the training error simultaneously, as training error tends to increase when we maximize the margin. To find a trade-off between them, the following constrained optimization problem gives an objective function that is a linear combination of margin size and training error.

Optimization problem 2: Soft-Margin SVM

$$\min \frac{1}{2}\,||w||^2 + C\sum \xi_i \tag{2}$$

subject to

$$y_i(w \cdot x_i - b) \geq 1 - \xi_i, \quad 1 \leq i \leq n. \tag{3}$$

As before, $y_i \in \{+1, -1\}$ is the class of the i th training point x_i. C is a regularization parameter that balances training error and margin size. Finally, ξ_i is a non-negative slack variable that represents the degree of misclassification of x_i; in particular, if $\xi_i > 1$, then data point i is on the wrong side of the hyperplane. Because this SVM allows data points to appear on the wrong side of the hyperplane, it is also known as a soft-margin SVM. Publicly available SVM software packages, such as SVM^{light}, train soft-margin SVMs when given a training set. AutoODC is implemented on top of SVM^{light}.

Now that we understand SVM's learning procedure, we describe how relevant annotations can be used to generate pseudo-instances. Let us begin by defining some notation. Let x_i be the vector representation of a training defect record R_i. Without loss of generality, assume that R_i belongs to class c_i. As mentioned before, since we adopt a one-versus-others training scheme for generating training instances, x_i is a positive instance for training the classifier representing class c_i. Given x_i, we construct one or more not-so-positive training instances v_{ij} (pseudo-instances). Specifically, we create v_{ij} by removing relevant annotation r_{ij} (the relevant words/phrases extracted from the original defect summary and description in the report as shown in the rightmost column in Table 1) from R_i. In other words, the number of pseudo-instances v_{ij} we create from each original instance x_i is equal to the number of relevant annotations present in R_i, and each v_{ij} is created by deleting exactly one relevant annotation substring from R_i. Since v_{ij} lacks a relevant annotation and therefore contains less evidence that a human annotator found relevant for classification than x_i, the correct SVM classifier should be less confident of a positive classification on v_{ij}.

This idea can be implemented by imposing additional constraints on the correct SVM classifier. Recall from the above discussion that the usual SVM constraint on positive instance x_i is $(w \cdot x_i - b) \geq 1$, which ensures that x_i is on the positive side of the hyperplane. In addition to these usual constraints, we desire that for each j, $(w \cdot x_i - w \cdot v_{ij}) \geq \mu$, where $\mu \geq 0$. Intuitively, this constraint specifies that x_i should be classified as more positive than v_{ij} by a margin of at least μ. The method AutoODC uses to adjust parameters such as μ and C is described in Sect. 5.2.

While we have thus far described how to generate positive pseudo-instances, we also create negative pseudo-instances similarly from each negative training instance.

Method 2: Learning algorithm-independent pseudo-instance generation

As described above, Method 1 is SVM-specific because it involves modifying SVM's learning procedure. Method 2, on the other hand, can be applied in combination with *any* machine learning algorithm because it treats a learning algorithm as a black box. In other words, Method 2 does *not* involve modifying any learning procedure.

The generation of pseudo-instances in Method 2 is motivated by a simple observation: since the relevant annotations associated with each training instance supposedly contains materials that are relevant for classification, we should create pseudo-instances solely from the relevant annotation associated with each training instances. Following the notation introduced in Method 1, for each training defect record R_i, we construct one or more pseudo-instances, where pseudo-instance v_{ij} contains exactly one relevant annotation, r_{ij}, in R_i. The class value of v_{ij} is the same as that of R_i. Intuitively, Method 2 seeks to strengthen the influence of the relevant annotations on the learning process via the introduction of pseudo-instances that contain all and only the relevant annotations.

Extension 2: Generating additional features

In our second extension to the basic defect classification system, we exploit the relevant annotations to generate additional features for training an SVM/NB classifier.

Recall that each defect record in the basic system is represented as a vector of words, where each single word in a defect record is a feature for the SVM/NB classifier. Since the relevant annotations (relevant words/phrases) used in Extension 1 are supposed to be relevant for classification, we hypothesize that they can also serve as useful features for the SVM/NB classifier. One way to exploit these relevant annotations as features is as follows. First, we collect all the relevant annotations from the defect records in the training set. Then we create one feature from each relevant annotation and augment the word-based feature set with these new features. In other words, each training/test instance that is not a pseudo-instance is now represented by a binary feature vector consisting of both the word-based features and the relevant annotation-based features.

Given a defect record, the value of a relevant annotation-based feature is 1 if and only if the corresponding relevant word/phrase appears in the defect record.

However, some of these relevant annotation-based features, especially those key phrases (e.g., “I would like to repair”) composed of many words, may not appear at all in the test set, a problem commonly known as data sparseness. This limits the usefulness of the relevant annotation-based features, as we may not be able to exploit them to classify the test instances if they are not present in the test set. To combat the data sparseness, we increase the likelihood of seeing a relevant annotation-based feature in the test set as follows. Instead of using a relevant annotation directly as a feature, we create all possible bigrams (i.e., consecutive words of length two) from each relevant annotation and use them as additional features instead. In this example, the created bigram features are “I would”, “would like”, “like to”, “to repair”. Another example key phrase would be “none of my directories are listed”, which can be formulated into a set of bigrams as “none of”, “of my”, “my directories”, “directories are”, “are listed”. These bigrams are created to represent the relevant annotation-based features composed of multiple words.

Extension 3: Exploiting domain knowledge

Another way to combat data sparseness is to apply domain knowledge to identify the relevant annotations that are synonymous. Specifically, we (1) collect all the relevant annotations from the training defect records, (2) have a human analyst partition them so that each cluster contains all and only synonymous relevant annotations, and (3) assign a unique id to each cluster. Given a training/test instance that is not a pseudo-instance, we exploit this domain knowledge as follows. We check whether a relevant annotation is present in the corresponding defect record. If so, we create an additional feature that corresponds to the id of the cluster containing the relevant annotation. An example of a synonym cluster from our defect report is {appear truncated, text cutoff}.

A natural question is: would the manual detection of synonyms be scalable for a large set of defect records? The answer is yes *provided that the synonyms are identified on the fly with the manual annotation and assignment of defect records to ODC categories*. More specifically, for a human analyst to annotate a defect record with its ODC category, she has to read its summary and description, looking for lexical cues (i.e., relevant annotations) that would enable her to correctly annotate it. For each lexical cue identified, she can simply assign it to one of the existing clusters of lexical cues (if it is synonymous with those in the cluster) or create a new cluster for it (if it is not synonymous with any of the existing cues). Hence, this on-the-fly synonym detection process creates the synonym clusters incrementally. Importantly, compared to a manual synonym detection process where the synonym clusters are created only after all the defect records are manually labeled, this incremental process adds comparatively less overhead to the labeling of defect records, making it scalable for a large set of defect records. Let us use an example to explain the process of on-the-fly intervention. For instance, as a human expert annotates the defect records, she has incrementally constructed a cluster that contains the following 3 relevant annotation synonyms (hangup, dies, hangs). Later when she is classifying and annotating another defect record that reads “when i was uploading files to my hosting sometimes it crashes like not responding on windows vista sp1”, she annotates “crashes” and “not responding" as new relevant annotations indicating the defect should be classified as *Reliability*. Meanwhile, she determines that both "crashes" and "not responding" can be added into the aforementioned existing synonym cluster. The cluster now contains 5 synonymous relevant annotations (hangup, dies, hangs, crashes, not responding).

Extension 4: Identifying not-so-relevant materials using shallow discourse analysis

While a relevant annotation can provide strong evidence of the correct defect category, there are many cases where using the relevant annotation *alone* is not sufficient for determining the correct defect category. This is especially true when the relevant annotation appears in certain linguistic constructs. More specifically, recall that in a defect record, not only will the user report the defect, but she will also describe the a *condition* or the *context* under which the defect occurs. If a relevant annotation appears in such a condition or context, it could provide a misleading signal for AutoODC, as reflected in the following examples.

Example 1 "It seems that *if* a *Timeout* is detected during a transfer of a file the console still thinks the file is tranfering and does not allow you to reset the transfer. The only way I can see to fix this is to exit and reopen the program." is an example of *Capability* defect. However, the word "*Timeout*" in the *if* clause is a strong indicator that this defect should be classified as a *Reliability* defect, which would be considered a misclassification.

Example 2 "*After install* version 1.7, FileZilla can't start, return error." is also a *Capability* defect. Unfortunately, the presence of "install" is a strong indicator of an *Installability* defect. This would be a misclassification, as "install" only appears in the *After* clause, indicating that it only describes the context where "FillZilla cannot start, return error", which is the problem the user intended to report in this defect record.

Example 3 "It then takes more than 30 s *before* it actually *displays* the application. Again, this may be just because it is beta, but I wanted to mention it. It doesn't seem to cause excessive CPU use or anything, it just takes 30 s to display." This instance belongs to *Performance* category but it contains a strong indicator of *Usability*, "*display*", in the *before* clause.

As we can see from the above examples, failure to analyze the context in which a relevant annotation appears can provide misleading signals for AutoODC. However, none of the extensions we have described thus far takes into account the context in which a relevant annotation appears. Extension 4 seeks to fill this gap. More specifically, we aim to *identify* such conditions and context and *remove* them from a defect record *before* it is even sent to AutoODC for analysis.

We identify such conditions and context, which might contain not-so-relevant materials as far as classification is concerned, via shallow *discourse analysis*. In NLP, discourse analysis refers to the analysis of *how two text segments are related to each other*. Recall that human writers typically express the relation between text segments via the use of *discourse connectives* such as *before*, *after*, *but*, *if*, etc. For instance, two segments connected by *before* or *after* indicate that there is a *temporal* relation between them, whereas two segments connected by *but* signals a *contrast* relation.

Returning to the question of how we can identify the aforementioned conditions and context, we employ the PDTB-styled end-to-end discourse parser (Lin et al. 2014) to perform shallow discourse analysis. More specifically, given a sentence, the parser determines whether a discourse connective exists. If so, it identifies the two text segments participating in the discourse relation and assigns a type to the relation (e.g., contrast, temporal). For our purposes, we are not concerned about the relation type. Rather, we will only make use of the text segments identified by the parser. In particular, a preliminary analysis of the defect records suggests that discourse connectives such as *if*, *before*, and *after* are likely to introduce a condition or context that contains materials irrelevant to classification.

Motivated in part by this preliminary analysis, we preprocess a defect record *before* it is sent to AutoODC for processing as follows. If a sentence in a training defect record contains an *if*, *before*, or *after*, we remove from the record the text segment *s* that immediately follows the discourse connective as well as the connective itself if *s* contains a relevant annotation. For instance, in Example 1, we remove *if a Timeout is detected during the transfer of a file*; in Example 2, we remove *After install version 1.7*; and in Example 3, we remove *before it actually displays the application*. However, since relevant annotations are *not* present in a test defect record, we have to preprocess a test record slightly differently. Specifically, we remove from a test record each text segment that immediately follows an *if*, *before*, or *after* as well as the connective itself.

We conclude this section by summarizing the difference between Extension 4 and the first three extensions: while the first three extensions seek to strengthen the influence of the *relevant materials* present in the relevant annotations on the learning process. Extension 4 aims to eliminate the influence of the materials that are *unlikely* to be relevant for classification from a defect record.

5 Evaluation

Our evaluation addresses six research questions:

RQ1: To what extent does our annotation relevance framework, which comprises the four aforementioned extensions to the basic defect classification system, help improve ODC defect classification, and are the four extensions all contributing positively to overall performance?

RQ2: How different is our annotation relevance framework from the basic defect classification system in terms of the classification errors they made?

RQ3: Since the relevant annotations are supposed to contain the information a human needs to assign an ODC class to a defect record, to what extent is the rest of a defect record (i.e., the portion of the record not belonging to any relevant annotations) contributing to overall performance?

RQ4: To what extent is the effectiveness of our annotation relevance framework dependent on the amount of training data? In other words, can our framework still improve AutoODC when the training set is small?

RQ5: Where can we further improve the classification accuracy of AutoODC?

RQ6: How is the cost-effectiveness of AutoODC in terms of effort reduction and analysts confidence improvement?

5.1 Experimental setup

Our experiments involve using AutoODC to classify defect records by their ODC "impact" attribute. To optimize the performance of AutoODC, we have experimented with multiple variations of AutoODC, combining basic defect classification system with each of the four extensions.

5.1.1 Datasets

We tested AutoODC on two defect reports (issue lists), one from an industrial organization (henceforth Company P) and the other from a publicly available defect tracker of the open source system FileZilla.[5] Specifically, we acquired a defect report (issue list) with 403 defect records (issues) submitted for a social network project from industrial Company P. We also randomly selected a total of 1,250 defect records (issues) from a larger defect report (issue list) on 3 of FileZilla's subsystems, with a majority of the records coming from the "Other" subsystem. To provide training/testing data for AutoODC, we asked two expert analysts to independently classify the 403 Company P defects into six categories under the "impact" attribute. They initially agreed on 90 % of the records, and then cross-validated their results to resolve the disagreements. The classified defects are distributed over the categories of Capability (284), Integrity/Security (11), Performance (1), Reliability (8), Requirements (39), and Usability (60). For the 1,250 defect records (issues) reported in FileZilla, the two human annotators initially agreed on 92 % of the records, and then resolved the disagreements via discussion. The classified defects are more evenly distributed over the eight categories of Capability (492), Reliability (269), Requirements (145), Usability (140), Integrity/Security (98), Performance (74), Installability (26), and Documentation (6), compared to the above mentioned industrial defect report. The two analysts, who are co-authors of the paper, had 6 and 3 years of ODC classification and analysis experience in industry, respectively. One of them worked as an ODC expert at the IBM quality assurance group for 6 years. They both marked the relevant words/phrases in each defect record indicating which ODC category the record belonged to. Moreover, they were asked to identify the synonyms among these relevant words/phrases.

5.1.2 Evaluation metrics

We employ two evaluation metrics. First, we report overall accuracy, which is the percentage of records in the test set correctly classified by a defect classification system. Second, we compute the precision and recall for each ODC category. Given an ODC category c, its recall is the percentage of defect records belonging to c that are correctly classified by the system; and its precision is the percentage of records classified by the system as c that indeed belong to c.

[5] FileZilla is a free FTP solution composed of three subsystems: FileZilla Client, FileZilla Server, and Other. The defect tracker for the three subsystems of FileZilla is accessible at http://trac.filezilla-project.org/wiki/Queries.

5.1.3 Evaluation methodology

All performance numbers we report in this section are obtained via fivefold cross-validation experiments. Specifically, we first randomly partition the defect records in each of the two evaluation datasets into five equal-sized subsets (or *folds*). Then, in each fold experiment, we reserve one of the five folds for testing and use the remaining four folds as the training set. We repeat this five times, each time using a different fold as a test set. Finally, we average the result obtained over the fivefolds.

5.2 Experimental results on the basic system and annotation relevance framework (RQ1)

5.2.1 The basic defect classification system

We first report the performance of the basic defect classification system. Each record is represented by a binary vector with the 1,089 unique words appearing in the data set as features. In the SVM experiments, we use SVM^{light} to train a soft-margin SVM classifier on the training set. Recall from OPTIMIZATION PROBLEM 2 that C is a tunable parameter, and AutoODC computes C that optimizes performance as follows. First, it partitions the available training data into two subsets following a 75–25 ratio. Second, it trains SVM classifiers with different values of C on the larger subset and measures their performance on the smaller subset. Finally, it selects the C value that offers the best performance on the smaller subset. After selecting C, an SVM classifier is retrained on the entire training set with this optimal C value. In the NB experiments, we use the Weka (Hall et al. 2009) implementation of the NB classifiers. Unlike in SVM, there are no tunable parameters in the NB classifier. Results of the basic defect classification system on Company P and FileZilla are shown in row 1 of Table 3. For Company P, the basic system achieves accuracies of 79.7 % (NB) and 74.2 % (SVM). For FileZilla, it achieves accuracies of 68.2 % (NB) and 71.2 % (SVM).

While in our basic defect classification system we trained a set of *binary* SVM classifiers where each defect record is represented as a *binary* vector of words *without* removing stopwords, one may wonder whether there are alternative ways that we can employ to train a stronger basic system. In particular, can we obtain a better-performing basic system by (1) training a single multi-class SVM classifier that can classify a record as belonging to one of the six categories, rather than a set of binary SVM classifiers; (2) employing frequency of occurrence for the bag-of-words representation of each defect record rather than using the presence/absence of a word; or (3) representing each record without using stopwords?

To answer this question, we conducted four experiments. In Experiment 1, we trained the basic defect classification system in the same way as before except that we trained a single multi-class SVM/NB classifier[6] rather than a set of binary SVM/NB classifiers. In Experiment 2, we trained the basic defect classification system in the same way as before except that we employed frequency of occurrence for the bag-

[6] To train a multi-class SVM classifier, we use $SVM^{multiclass}$ (Tsochantaridis et al. 2004). To train a multi-class NB classifier, we use the implementation in Weka.

Table 3 Fivefold cross-validation accuracy for variants

	Company P		FileZilla	
	NB	SVM	NB	SVM
Basic	79.7	74.2	68.2	71.2
Multi-class	80.6	73.5	66.6	70.8
Freq. of occurrence	73.9	72.5	58.2	70.7
Stopwords removed	77.7	70.5	68.8	70.3
Multi-class w/ Freq. of occurrence and stopwords removed	71.0	68.7	56.6	70.2

Table 4 Paired *t* test results on accuracy for variants

Approach A	Approach B	Company P		FileZilla	
		NB	SVM	NB	SVM
Basic	Multi-class	0.327	0.5614	0.03934	0.08901
Basic	Freq	0.06003	0.26	0.00185	0.2835
Basic	Stopwords removed	0.3977	0.08332	0.4814	0.4177
Basic	Multi w./ Freq-Stopwords	0.02443	0.1225	0.0004895	0.3675

of-words representation of a defect record rather than using the presence/absence of a word. In Experiment 3, we trained the basic defect classification system in the same way as before except that we represented each record without using stopwords. Finally, in Experiment 4, we trained the basic defect classification system in the same way as before except that we trained a single multi-class SVM/NB classifier (as in Experiment 1), employed frequency of occurrence for the bag-of-words representation of a defect record (as in Experiment 2), *and* represented each record without using stopwords. In other words, in Experiment 4, we applied the three modifications in Experiments 1, 2, and 3 in combination with the training of the basic defect classification system.

Results of Experiments 1, 2, 3, and 4 on Company P and FileZilla are shown in the rows 2, 3, 4, and 5 of Table 3 respectively. To determine whether the difference in accuracy between our basic defect classification system (row 1) and each of these four system variants is statistically significant, we employ the two-tailed paired t-test. In each significance test, we employ the null hypothesis that there is no performance difference between the two systems under comparison.

Next, we describe the significance test results. For ease of exposition, when describing the result of a significance test, we will use the term (1) *highly significant* if the performance difference between the two systems under consideration is significant at the $p < 0.01$ level; (2) *significant* if their difference is significant at the $p < 0.05$ level; (3) *moderately significant* if their difference is significant at the $p < 0.1$ level; and (4) *statistically indistinguishable* if their difference is *not* significant at the $p < 0.1$ level.

The significance test results are shown in Table 4. Each row shows the pair of systems to which the test is applied, as well as the p value at which the two systems' difference in accuracy is statistically significant for each of the two datasets (Company P and FileZilla) and each of the two learning algorithms (NB and SVM). From row 1, we can see that on Company P, the basic system, which trains binary SVM/NB classifiers, is statistically indistinguishable from its multi-class counterpart. On the other hand, on FileZilla, the basic system is significantly better than its multi-class counterpart when NB is used, and their difference is moderately significant when SVM is used. From row 2, we can see that on Company P, the basic system, which employs binary-valued features, is moderately significantly better than its frequency-based counterpart when NB is used, but the two systems are statistically indistinguishable when SVM is used. Similar trends can be observed on FileZilla: the basic system performs highly significantly better than its frequency-based counterpart when NB is used, but no significant difference is observed when SVM is used. From row 3, we can see that in general, removing stopwords neither improves nor hurts performance, although the basic system is moderately significantly better than the one without stopwords on Company P when SVM is used. Finally, from row 4, we can see that when all three modifications are made, our basic system is significantly or highly significantly better than its counterpart when NB is used, but the two systems are statistically indistinguishable when SVM is used.

Overall, these statistical significance test results suggest that while the results are different depending on which learning algorithm and which dataset are used, under no circumstances is our basic system statistically worse than any of the system variants. In other words, these results justify our use of the basic defect classification system as our baseline against which we evaluate the effectiveness of our four extensions.

5.2.2 Effectiveness of our four extensions to the basic system

We begin by incorporating Extension 1 (generating pseudo-instances for training) into the basic system. Recall that in the SVM-based version of this extension, both C and μ are tunable. AutoODC selects optimal values for these two parameters using essentially the same method that we described above for tuning C. On the other hand, in the learning algorithm-independent version of this version, we employ NB, and as a result, there are no tunable parameters. Results of this extension on Company P and FileZilla are shown in row 2 of Tables 5 and 6, respectively. To facilitate comparison, we show again the performance of the basic defect classification system in row 1 of Tables 5 and 6, but this time performance is reported not only in terms of accuracy but also precision, recall and F-score (the unweighted harmonic mean of precision and recall) for each ODC class. Note that Table 5 contains results for all but the *Performance* class. We omitted the results for *Performance* on Company P for the obvious reason: Company P has only one instance of *Performance*, and since this instance can appear in either the training set or the test set for a given fold experiment, the F score achieves for this ODC class will always be zero. As we can see in row 2, on Company P, Extension 1 gives improvements of 2.7 % (SVM) and 1.4 % (NB) in accuracy over the basic system. On FileZilla, this extension gives improvements of 2.6 % (SVM) and 3.4 % (NB) in accuracy over the basic system.

Next, we incorporate Extension 2 (employing the relevant annotations to generate additional features). Row 3 of Tables 5 and 6 show the results when we apply it in

Table 5 Fivefold cross-validation results on Company P

Experiment	Accuracy	Reliability			Capability			Integrity/Security			Usability			Requirements			Performance		
		R	*P*	*F*	*R*	*P*	*F*	*R*	*P*	*F*	*R*	*P*	*F*	*R*	*P*	*F*	*R*	*P*	*F*
NB																			
Basic	79.7	0.0	0.0	0.0	94.7	82.5	88.2	9.1	100.0	16.7	61.7	63.8	62.7	35.9	77.8	49.1	0.0	0.0	0.0
Basic + Ext 1	81.1	12.5	100.0	22.2	94.4	84.0	88.9	45.5	100.0	62.5	56.7	68.0	61.8	48.7	67.9	56.7	0.0	0.0	0.0
Basic + Ext 1,2	82.1	12.5	100.0	22.2	93.0	86.8	89.8	54.6	100.0	70.6	65.0	63.9	64.5	53.9	67.7	60.0	0.0	0.0	0.0
Basic + Ext 1,2,3	82.4	12.5	100.0	22.2	92.6	86.5	89.5	45.5	100.0	62.5	66.7	69.0	67.8	59.0	65.7	62.2	0.0	0.0	0.0
Basic + Ext 1,2,3,4	82.9	12.5	100.0	22.2	93.3	86.9	90.0	45.5	100.0	62.5	66.7	69.0	67.8	59.0	67.7	63.0	0.0	0.0	0.0
SVM																			
Basic	74.2	0.0	0.0	0.0	93.3	78.2	85.1	18.2	66.7	28.6	38.3	50.0	43.4	23.1	60.0	33.3	0.0	0.0	0.0
Basic + Ext 1	76.9	0.0	0.0	0.0	93.3	80.5	86.5	45.5	71.4	55.6	46.7	60.9	52.8	30.8	57.1	40.0	0.0	0.0	0.0
Basic + Ext 1,2	78.9	12.5	100.0	22.2	95.1	81.1	87.5	45.5	71.4	55.6	48.3	70.7	57.4	33.3	61.9	43.3	0.0	0.0	0.0
Basic + Ext 1,2,3	80.6	0.0	0.0	0.0	94.4	83.0	88.3	54.5	85.7	66.7	65.0	72.2	68.4	30.8	63.2	41.4	0.0	0.0	0.0
Basic + Ext 1,2,3,4	80.7	0.0	0.0	0.0	94.4	83.5	88.6	63.6	70.0	66.7	63.3	71.7	67.3	30.8	63.2	41.4	0.0	0.0	0.0

P, *R*, *F* stand for Precision, Recall, *F* score respectively

Table 6 Fivefold cross-validation results on FileZilla

Experiment	Accuracy	Reliability			Capability			Integrity/Security			Usability			Requirements			Performance			Documentation			Installability		
		R	P	F	R	P	F	R	P	F	R	P	F	R	P	F	R	P	F	R	P	F	R	P	F
NB																									
Basic	68.2	77.7	73.6	75.6	81.1	66.4	73.0	59.2	71.6	64.8	43.6	52.6	47.7	57.9	70.6	63.6	55.4	87.2	67.8	0.0	0.0	0.0	7.7	50.0	13.3
Basic + Ext 1	71.6	81.4	75.3	78.2	78.0	72.8	75.3	69.4	70.1	69.7	46.4	58.0	51.6	67.6	68.5	68.1	74.3	80.9	77.5	16.7	100.0	28.6	23.1	50.0	31.6
Basic + Ext 1,2	73.6	80.7	79.8	80.2	78.0	75.4	76.7	73.5	73.5	73.5	48.6	59.1	53.3	73.1	69.3	71.1	87.8	74.7	80.8	16.7	100.0	28.6	30.8	50.0	38.1
Basic + Ext 1,2,3	75.4	82.9	82.0	82.4	79.8	77.0	78.4	72.5	74.0	73.2	52.9	60.2	56.3	73.8	72.3	73.0	90.5	80.7	85.4	16.7	100.0	28.6	42.3	61.1	50.0
Basic + Ext 1,2,3,4	77.5	82.2	83.4	82.8	80.5	78.6	79.5	75.5	74.8	75.1	56.4	63.2	59.6	80.7	77.5	79.1	91.9	79.1	85.0	16.7	100.0	28.6	50.0	68.4	57.8
SVM																									
Basic	71.2	75.8	82.6	79.1	87.6	66.7	75.8	61.2	74.1	67.0	19.3	52.9	28.3	66.2	70.6	68.3	81.1	83.3	82.2	33.3	50.0	40.0	38.5	76.9	51.3
Basic + Ext 1	73.8	78.1	86.1	81.9	85.4	68.3	75.9	70.4	74.2	72.3	34.3	63.2	44.4	68.3	75.6	71.7	86.5	87.7	87.1	16.7	50.0	25.0	46.2	75.0	57.1
Basic + Ext 1,2	73.4	78.4	85.1	81.6	86.8	68.9	76.8	64.3	74.1	68.9	32.9	60.5	42.6	63.5	73.6	68.2	86.5	84.2	85.3	33.3	66.7	44.4	50.0	76.5	60.5
Basic + Ext 1,2,3	73.9	79.6	86.3	82.8	85.0	69.9	76.7	67.4	76.7	71.7	37.9	58.2	45.9	66.2	73.3	69.6	85.1	82.9	84.0	50.0	100.0	66.7	42.3	64.7	51.2
Basic + Ext 1,2,3,4	75.2	81.0	87.9	84.3	85.8	71.0	77.7	64.3	76.8	70.0	37.9	53.5	44.4	69.0	80.0	74.1	86.5	86.5	86.5	50.0	100.0	66.7	65.4	68.0	66.7

P, *R*, *F* stand for precision, recall, *F* score respectively

combination with Extension 1 (row 2 of Tables 5 and 6) on Company P and FileZilla, respectively. Adding Extension 2 on top of Extension 1 yields accuracy improvements of 2.0 % (SVM) and 1.0 % (NB) on Company P but produces mixed results on FileZilla, where accuracy improves by 2.0 % when NB is used but drops by 0.4 % when SVM is used. On the other hand, the combined improvements of Extensions 1 and 2 over the basic system are 4.7 % (SVM) and 2.4 % (NB) on Company P and 2.2 % (SVM) and 5.4 % (NB) on FileZilla.

Next, we incorporate Extension 3 into the system (classification with domain knowledge). Specifically, the results in row 4 of Tables 5 and 6 are obtained by applying the first three extensions to the basic system. Adding Extension 3 on top of Extensions 1 and 2 yields accuracy improvements of 1.7 % (SVM) and 0.3 % (NB) on Company P and 0.5 % (SVM) and 1.8 % (NB) on FileZilla. On the other hand, the combined improvements of Extensions 1, 2 and 3 over the basic system are 6.4 % (SVM) and 2.7 % (NB) on Company P and 2.7 % (SVM) and 7.2 % (NB) on FileZilla.

Finally, we incorporate Extension 4 into the system (removing not-so-relevant materials via discourse analysis). Results are shown in row 5 of Tables 5 and 6. Specifically, the results in row 5 of Tables 5 and 6 are obtained by applying all four extensions to the basic system. Adding Extension 4 on top of Extensions 1, 2 and 3 yields accuracy improvements of 0.1 % (SVM) and 0.5 % (NB) on Company P and 1.3 % (SVM) and 2.1 % (NB) on FileZilla.

In comparison to the basic system, the four extensions together improve accuracy by 6.5 % (SVM) and 3.2 % (NB) on Company P, and by 4.0 % (SVM) and 9.3 % (NB) on FileZilla. These results translate to a relative error reduction of 23 % (SVM) and 12 % (NB) on Company P, and a relative error reduction of 14 % (SVM) and 25 % (NB) on FileZilla.

Next, we employ the paired t-test to determine statistical significance. We conduct two sets of significance tests. The first set aims to determine whether the addition of an extension on top of the previous extensions produces significant improvements or not. This set of tests will enable us to determine the usefulness of each extension. The second set aims to determine whether the extensions, when applied in combination, produce significant improvements over the basic system or not.

Results of the first set of significance tests are shown in the first four rows of Table 7. As we can see, on Company P, none of these extensions, when added on top of the previous extensions, yields any significant improvements. Different results are obtained on FileZilla, however. When NB is used, adding Extension 1 to the basic system yields a significant improvement, adding Extension 2 on top of Extension 1 yields a highly significant improvement, adding Extension 3 on top of the first two extensions yields a moderately significant improvement; and adding Extension 4 on top of the first three extensions yields a significant improvement. When SVM is used, adding Extensions 1 and 4 yields a moderately significant improvement, but adding Extensions 2 and 3 does not.

Results of the second set of significance tests are shown in the last four rows of Table 7. On Company P, when NB is used, we see significant improvements over the basic system after applying the first three extensions. On the other hand, when SVM is used, we can already see significant improvements over the basic system after applying the first two extensions, and the improvements become highly significant

Table 7 Paired *t* test results on accuracy

Approach A	Approach B	Company P		FileZilla	
		NB	SVM	NB	SVM
Basic	Basic + Ext 1	0.3267	0.3682	0.02697	0.07136
Basic + Ext 1	Basic + Ext 1,2	0.4511	0.1203	0.00394	0.6968
Basic + Ext 1,2	Basic + Ext 1,2,3	0.7509	0.2371	0.08451	0.1098
Basic + Ext 1,2,3	Basic + Ext 1,2,3,4	0.1778	0.6208	0.02913	0.05994
Basic	Basic + Ext 1	0.3267	0.3682	0.02697	0.07136
Basic	Basic + Ext 1,2	0.2302	0.0136	0.01365	0.05161
Basic	Basic + Ext 1,2,3	0.04	0.01453	0.01236	0.03192
Basic	Basic + Ext 1,2,3,4	0.03323	0.003995	0.002881	0.008146

when all four extensions are applied. On FileZilla, when NB is used, we can already see significant improvements over the basic system after applying the first extension, and the improvements become highly significant when all four extensions are applied. On the other hand, when SVM is used, we see moderately significant improvements over the basic system after applying the first extension, and the improvements become significant and highly significant after applying the first three extensions and all four extensions, respectively.

Overall, these significance test results show that while a particular extension may not always yield significant improvement when added incrementally to the system, their cumulative benefits are significant.

5.3 Differences between the basic system and the full system (RQ2)

To determine how different the basic system and the full system (i.e., the basic system augmented with our four extensions) are in terms of the classification errors they made, we show their confusion matrices. Specifically, Tables 8 and 9 show the confusion matrices for the basic system and the full system on Company P when SVM and NB are used, and Tables 10 and 11 show the confusion matrices for the two systems on FileZilla when SVM and NB are used. Note that the rows and columns in these tables correspond to the correct categories and the predicted categories, respectively.

From these confusion matrices, it may not be easy to see whether the full system is indeed significantly better than the basic system with respect to each ODC category. As a result, we conduct paired *t*-test tests again to determine whether the difference in F-measure between the basic system and the full system with respect to each ODC category is indeed statistically significant or not. The null hypothesis is that there is no difference in the F-measure score between the two systems with respect to the ODC category under consideration.

Statistical significance test results are shown in Table 12. On Company P, when NB is used, the full system is significantly better than the basic system with respect to *Integrity/Security*. When SVM is used, in comparison to the basic system, the full

Table 8 Company P defect report with 403 defect records

	Reliability	Capability	Integrity/Security	Performance	Usability	Requirements
Confusion matrix for the basic defect classification system (SVM)						
Reliability	0	8	0	0	0	0
Capability	0	265	0	0	15	4
Integrity/security	0	9	2	0	0	0
Performance	0	0	0	0	1	0
Usability	0	35	0	0	23	2
Requirements	0	22	1	0	7	9
Confusion matrix for the Basic + Ext 1,2,3,4 defect classification system (SVM)						
Reliability	0	8	0	0	0	0
Capability	0	268	0	0	11	5
Integrity/security	0	3	7	0	0	1
Performance	0	0	0	0	1	0
Usability	0	21	0	0	38	1
Requirements	0	21	3	0	3	12

Confusion matrices for the basic defect classification system and Basic + Ext 1,2,3,4 classification system on SVM. The rows and columns correspond to the correct and predicted ODC categories, respectively

Table 9 Company P defect report with 403 defect records

	Reliability	Capability	Integrity/Security	Performance	Usability	Requirements
Confusion matrix for the basic defect classification system (NB)						
Reliability	0	8	0	0	0	0
Capability	0	269	0	0	12	3
Integrity/security	0	10	1	0	0	0
Performance	0	0	0	0	1	0
Usability	0	22	0	0	37	1
Requirements	0	17	0	0	8	14
Confusion matrix for the Basic + Ext 1,2,3,4 defect classification system (NB)						
Reliability	1	7	0	0	0	0
Capability	0	265	0	0	14	5
Integrity/security	0	5	5	0	0	1
Performance	0	0	0	0	1	0
Usability	0	15	0	0	40	5
Requirements	0	13	0	0	3	23

Confusion matrices for the basic defect classification system and Basic + Ext 1,2,3,4 classification system on NB. The rows and columns correspond to the correct and predicted ODC categories, respectively

system is significantly better with respect to *Usability* and highly significantly better with respect to *Capability*.

On FileZilla, when NB is used, in comparison to the basic system, the full system performs highly significantly better with respect to *Reliability* and *Capability*, signif-

Table 10 FileZilla defect (issue) list with 1,250 issues (defect records)

	Documentation	Installability	Reliability	Capability	Integrity/Security	Performance	Usability	Requirements
Confusion matrix for the basic defect classification system (SVM)								
Documentation	2	0	0	2	0	0	0	2
Installability	0	10	0	13	0	0	1	2
Reliability	0	0	204	55	1	4	2	3
Capability	0	2	21	431	7	4	11	16
Integrity/security	0	0	2	22	60	0	2	12
Performance	0	0	6	6	0	60	2	0
Usability	1	0	8	88	7	4	27	5
Requirements	1	1	6	29	6	0	6	96
Confusion matrix for the Basic + Ext 1,2,3,4 defect classification system (SVM)								
Documentation	3	0	0	1	1	0	0	1
Installability	0	17	0	7	0	0	2	0
Reliability	0	0	218	47	1	2	1	0
Capability	0	6	23	422	5	1	28	7
Integrity/security	0	0	1	19	63	1	3	11
Performance	0	0	1	6	0	64	2	1
Usability	0	1	4	67	6	4	53	5
Requirements	0	1	1	25	6	2	10	100

Confusion matrices for the basic defect classification system and Basic + Ext 1,2,3,4 classification system on SVM. The rows and columns correspond to the correct and predicted ODC categories, respectively

Table 11 FileZilla defect (issue) list with 1,250 issues (defect records)

	Documentation	Installability	Reliability	Capability	Integrity/Security	Performance	Usability	Requirements
Confusion matrix for the basic defect classification system (NB)								
Documentation	0	0	0	3	0	0	0	3
Installability	0	2	3	16	2	0	0	3
Reliability	0	0	209	52	1	5	2	0
Capability	0	2	39	398	6	0	35	11
Integrity/security	0	0	6	19	58	0	4	11
Performance	0	0	19	8	0	41	2	4
Usability	0	0	5	67	4	0	61	3
Requirements	0	0	3	35	10	1	12	84
Confusion matrix for the Basic + Ext 1,2,3,4 defect classification system (NB)								
Documentation	1	0	0	1	0	0	2	2
Installability	0	13	2	7	3	0	0	1
Reliability	0	0	221	38	2	6	1	1
Capability	0	4	29	396	10	4	34	15
Integrity/security	0	0	3	10	74	0	2	9
Performance	0	0	4	0	0	68	0	2
Usability	0	2	6	40	3	6	79	4
Requirements	0	0	0	12	7	2	7	117

Confusion matrices for the basic defect classification system and Basic + Ext 1,2,3,4 classification system on NB. The rows and columns correspond to the correct and predicted ODC categories, respectively

Table 12 Paired *t* test results on per-category *F* measure

	Company P			FileZilla	
	NB Basic versus Basic + Ext 1,2,3,4	SVM Basic versus Basic + Ext 1,2,3,4		NB Basic versus Basic + Ext 1,2,3,4	SVM Basic versus Basic + Ext 1,2,3,4
Reliability	NA	NA	Documentation	0.7039	0.242
Capability	0.7145	0.002874	Installability	0.0794	0.1043
Integrity/security	0.02992	0.2502	Reliability	0.008655	0.04332
Usability	0.4136	0.01243	Capability	0.008826	0.09051
Requirements	0.5902	0.4983	Integrity/security	0.06493	0.4262
			Performance	0.04341	0.1741
			Usability	0.01359	0.1693
			Requirements	0.03254	0.2159

icantly better with respect to *Performance*, *Usability*, and *Requirements*, moderately significantly better with respect to *Installability* and *Integrity/Security*, and at the same level with respect to *Documentation*.

When SVM is used, in comparison to the basic system, the full system performs significantly better in *Reliability*, and moderately significantly better in *Capability*.

5.4 The value of relevant annotations (RQ3)

Recall that the motivation behind employing relevant annotations is that they allow the learning algorithm to focus on learning from the relevant portions of a defect record. A few questions naturally arise. First, if we remove the relevant annotations from each training defect record and train a classifier on the remaining portions of the records, how well will the classifier perform? In other words, will learning completely fail after removing the relevant materials from the training defect records? Second, is learning still necessary given these relevant annotations? In other words, can we exploit these relevant annotations in a non-learning framework and still achieve a good accuracy?

To answer the first question, we conducted an experiment, which we will refer to as Experiment A. We ask the learning algorithm to learn from the portion of a defect record that does not belong to a relevant annotation. More specifically, we (1) removed all the words that are part of a relevant annotation from each defect record in the training set, (2) generated a binary feature vector from each training record as in the basic system, and (3) trained a SVM/NB classifier on the resulting training set. If the resulting classifier performed poorly, it would imply that the materials outside the relevant annotations are largely not useful as far as classification is concerned.

Results of this experiment on Company P and FileZilla are shown in row 2 of Tables 13 and 14 respectively. Comparing rows 1 and 2, we can see that on Company

P, the classifier from Experiment A underperforms the basic system by 0.0 % (NB) and 0.2 % (SVM); and on FileZilla, the classifier from Experiment A underperforms the basic system by 0.9 % (NB) and 0.4 % (SVM). While it is not surprising that the system in Experiment B underperforms the basic system, what is perhaps somewhat surprising is that even without relevant annotations, the SVM/NB classifier is still able to train a system that performs comparably to the basic system. More precisely, according to the paired *t* test shown in Table 15, the difference in accuracy between the basic system and the classifier from Experiment A is statistically indistinguishable in all but one case (when NB is used on Company P). Taken together, these results suggest that the portion of a defect record that is outside its relevant annotation may still contain patterns that cannot be detected by a human and yet can be exploited to perform a decent job on defect classification.

To answer the second question, we examined a simple heuristic approach to defect classification using the relevant annotations. In this approach, we first create a list of relevant annotations for each ODC class based on the defect records in the training set. More specifically, if we encounter relevant annotation *r* in a defect record *d* with ODC class *c* in the training set, we add *r* to the list for *c*. We can then classify a defect record e from the test set using these lists as follows. For each ODC class *c*, we count the number of times the relevant annotations in its list appears in *e*, and assign *e* the class whose relevant annotations appear most frequently in it. Two implementation details deserve mention. First, if the same relevant annotation appears in *e* more than once, we count it as many times as it appears. Second, if a relevant annotation belonging to more than one ODC class appears in *e*, we give credit to all these classes. We will refer to this experiment as Experiment B.

Results of Experiment B on Company P and FileZilla are shown in row 3 of Tables 13 and 14, respectively. As we can see, this heuristic approach yields accuracies of 71.5 % on Company P and 68.0 % on FileZilla. The system for Experiment B underperforms that for the basic system by accuracies of 1.7 % (SVM) and 7.2 % (NB) on Company P and by 3.2 % (SVM) and 0.2 % (NB) on FileZilla. According to the statistical test results in Table 15, on Company P, the differences between these two systems are highly significant when NB is used and significant when SVM is used, but the two systems are statistically indistinguishable on FileZilla, regardless of which learning algorithm is used.

While it is interesting to see that the heuristic approach is statistically indistinguishable from the machine learning approach on FileZilla, we believe that the performance of the heuristic approach depends to a large extent on the relevant annotations that can be extracted from a dataset. Specifically, if the relevant annotations manually extracted for different ODC categories have little overlap (i.e., each ODC category is characterized by a fairly unique set of relevant annotations), then even the simplistic heuristic approach can potentially perform well. On the other hand, in cases where the ODC categories are not easily separable given the relevant annotations, the simplistic heuristic approach may not perform as well as a machine learning approach. Overall, these results suggest that machine learning has its own value, as the learned classifiers perform as least as well as and possibly better than their heuristic counterparts.

Table 13 Fivefold cross-validation results for variants on Company P

Experiment	Accuracy	Reliability			Capability			Integrity/Security			Usability			Requirements			Performance		
		R	P	F	R	P	F	R	P	F	R	P	F	R	P	F	R	P	F
NB																			
Basic	79.7	0.0	0.0	0.0	94.7	82.5	88.2	9.1	100.0	16.7	61.7	63.8	62.7	35.9	77.8	49.1	0.0	0.0	0.0
Rel. removed	79.7	0.0	0.0	0.0	95.1	81.8	88.0	9.1	100.0	16.7	61.7	66.1	63.8	33.3	81.3	42.3	0.0	0.0	0.0
Heuristic	72.5	50.0	18.2	26.7	90.5	76.9	83.2	81.8	60.0	69.2	28.3	77.3	41.5	12.8	50.0	20.4	0.0	0.0	0.0
SVM																			
Basic	74.2	0.0	0.0	0.0	93.3	78.2	85.1	18.2	66.7	28.6	38.3	50	43.4	23.1	60	33.3	0.0	0.0	0.0
Rel. removed	74.0	0.0	0.0	0.0	93.3	77.9	84.9	0.0	0.0	0.0	36.7	52.4	43.1	28.2	52.4	36.7	0.0	0.0	0.0
Heuristic	72.5	50.0	18.2	26.7	90.5	76.9	83.2	81.8	60.0	69.2	28.3	77.3	41.5	12.8	50.0	20.4	0.0	0.0	0.0

P, *R*, *F* stand for precision, recall, *F* score respectively

Table 14 Fivefold cross-validation results for variants on FileZilla

Experiment	Accuracy	Reliability			Capability			Integrity/Security			Usability			Requirements			Performance			Documentation			Installability		
		R	P	F	R	P	F	R	P	F	R	P	F	R	P	F	R	P	F	R	P	F	R	P	F
NB																									
Basic	68.2	77.7	73.6	75.6	81.1	66.4	73.0	59.2	71.6	64.8	43.6	52.6	47.7	57.9	70.6	63.6	55.4	87.2	67.8	0.0	0.0	0.0	7.7	50.0	13.3
Rel. removed	67.3	73.6	68.5	71.0	78.9	66.3	72.1	63.3	69.7	66.3	45.0	52.9	48.7	56.6	68.3	61.9	51.4	86.4	64.4	0.0	0.0	0.0	7.7	50.0	13.3
Heuristic	68.0	86.4	59.9	70.7	82.5	82.5	82.5	76.5	68.2	72.1	23.4	75.6	35.8	0.0	0.0	0.0	17.1	72.7	27.7	66.7	66.7	66.7	50.0	81.2	61.9
SVM																									
Basic	71.2	75.8	82.6	79.1	87.6	66.7	75.8	61.2	74.1	67.0	19.3	52.9	28.3	66.2	70.6	68.3	81.1	83.3	82.2	33.3	50.0	40.0	38.5	76.9	51.3
Rel. removed	70.8	76.2	81.7	78.9	86.0	66.9	75.3	66.3	73.9	69.9	16.4	50.0	24.7	67.6	68.1	67.8	82.4	82.4	82.4	0.0	0.0	0.0	38.5	76.9	51.3
Heuristic	68.0	86.4	59.9	70.7	82.5	82.5	82.5	76.5	68.2	72.1	23.4	75.6	35.8	0.0	0.0	0.0	17.1	72.7	27.7	66.7	66.7	66.7	50.0	81.2	61.9

P, *R*, *F* stand for precision, recall, *F* score respectively

Table 15 Paired *t* test results on accuracy for variants

		Company P		FileZilla	
Approach A	Approach B	NB	SVM	NB	SVM
Basic	Rel. removed	0.9932	0.3739	0.0003578	0.2663
Basic	Heuristic	0.006585	0.02137	0.6621	0.1493

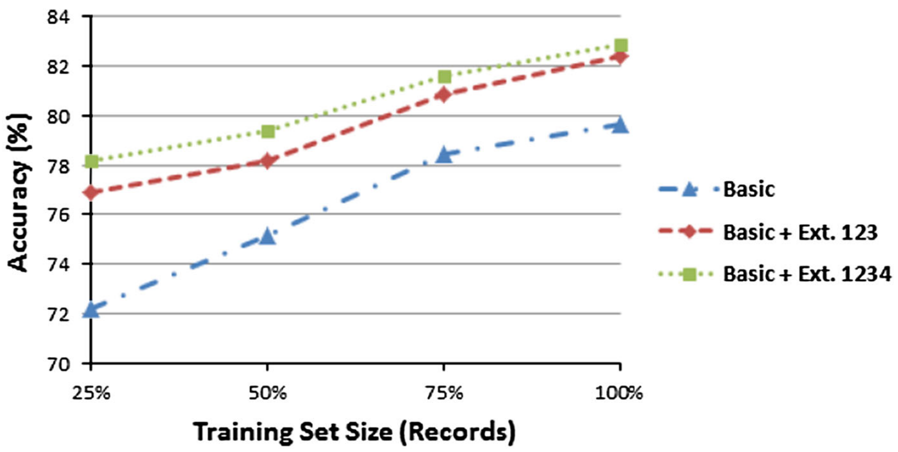
5.5 Learning curves (RQ4)

An interesting question is: will our annotation relevance framework still be able to improve ODC classification when only a small amount of training data is available? To answer this question, we plot in Figs. 2 and 3 the learning curves for the three experiments (Basic, Basic + Ext.1,2,3, Basic + Ext.1,2,3,4) discussed in Sect. 5.2. Figure 2 shows the learning curves for both NB and SVM on the Company P dataset. Figure 3 presents the learning curves for NB and SVM on the FileZilla dataset. Each learning curve is drawn based on four data points that correspond to the performance of a classification system when it is trained on 25, 50, 75, and 100 % of the available training data. As before, each data point is obtained via fivefold cross-validation experiments.

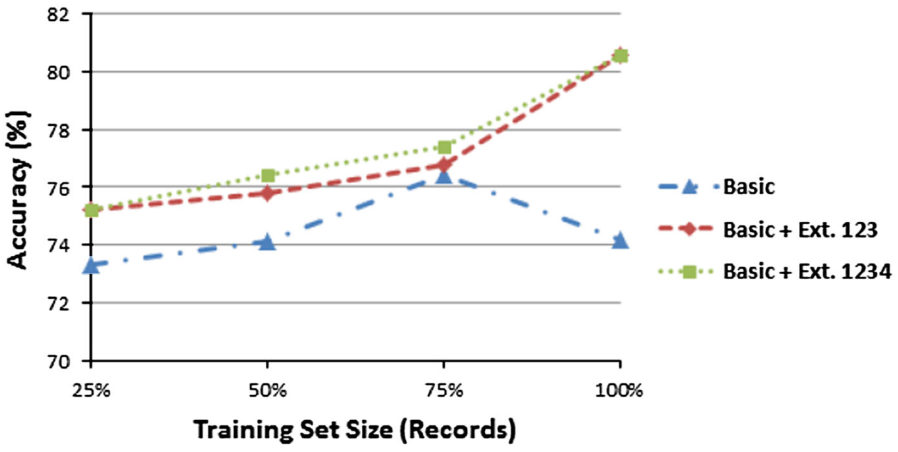
Figures 2 and 3 reveal several interesting observations about our four extensions to the basic defect classification system on both datasets (Company P and FileZilla). First, Basic is unstable on SVM in particular for the smaller dataset (Company P's defect report): its performance does not always increase with the amount of training data. Second, Basic + Ext.1,2,3 performs much better than Basic when only a small amount of training data is available. Basic + Ext.1,2,3,4 outperforms even Basic+Ext.1,2,3 in the presence of a small training dataset. Third, Basic + Ext.1,2,3 always outperforms Basic, suggesting the robustness of the three extensions when applied in combination. Further, the full system with Ext.1,2,3,4 always outperforms Basic + Ext.1,2,3. These indicate that the effectiveness of a combination of Ext.1,2,3 as well as Extension 4 (Identifying not-so-relevant materials using shallow discourse analysis) does not depend on the amount of available training data. Finally, the addition of Extension 4 to Basic+Ext.1,2,3 has a greater positive effect under NB than SVM.

5.6 Error analysis (RQ5)

To further improve the accuracy of AutoODC and identify the prospective extensions which we could explore in the future, we have performed a comprehensive error analysis to understand the distribution of misclassified defects in SVM and NB as well as the root causes why a defect is still misclassified. Table 16 lists the overall percentage of misclassifications as well as the number of misclassified defects in the FileZilla dataset for AutoODC with all four extensions based on NB and SVM, respectively. Table 17 lists the overall percentage of misclassifications as well as the number of misclassified defects in the Company P dataset for NB and SVM. We have also analyzed the number of defects misclassified by NB and SVM on both datasets.



(a) NB



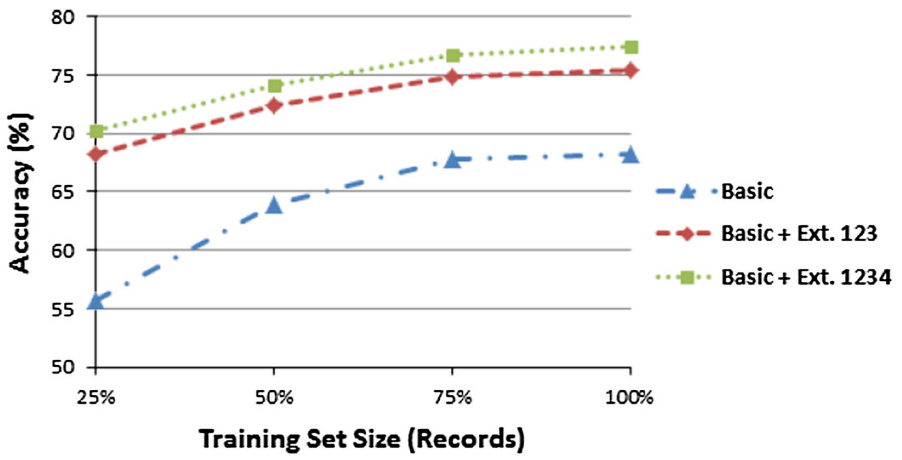
(b) SVM

Fig. 2 Company P defect report: learning curves for 3 variants of AutoODC based on NB and SVM

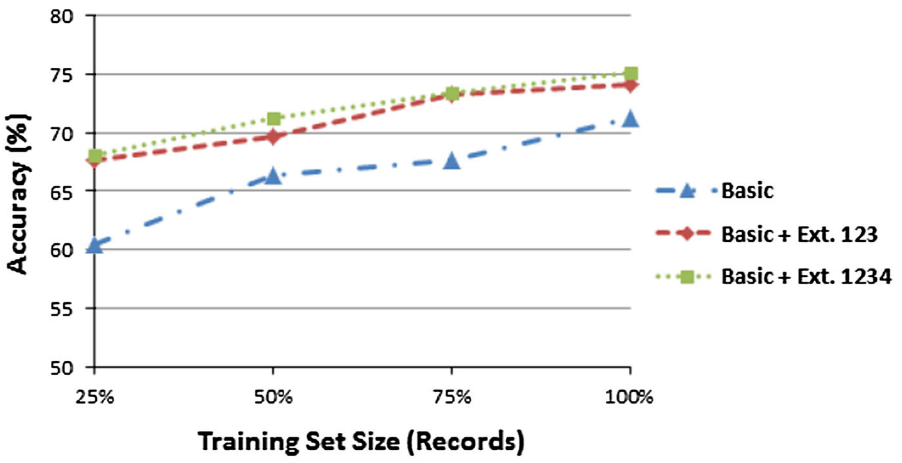
Table 18 shows on both datasets the number of defects correctly classified by both NB and SVM, the number of defects misclassified by both NB and SVM, the number of defects correctly classified by NB but not SVM, and the number of defects correctly classified by SVM but not NB.

Based on our analysis, we have observed two typical types of errors for a majority of the misclassified defects by NB and SVM.

– **Type 1 Errors** *The misclassified defect instance contains terms that overlap with terms from another defect class, which results in an ambiguous feature vector. These terms are not relevant annotations for any of the defect classes but are recognized by the learning algorithms to have strong association with a specific defect class.* In FileZilla, approximately 53 % of the erroneous classifications



(a) NB



(b) SVM

Fig. 3 FileZilla defect list: learning curves for 3 variants of AutoODC based on NB and SVM

by SVM and 52 % of the erroneous classifications by NB belong to this type. In Company P, approximately 68 % of the erroneous classifications by SVM and 69 % of the erroneous classifications by NB belong to this type. Here is an example from FileZilla: "When clicking on View Full *Profile*, a lot of the information from the regular *profile page* vanishes. See the attached picture." This defect instance should belong to the "*Reliability*" class. However, its description contains the terms "profile" and "page" which happened to occur frequently in the defect descriptions belonging to the "*Capability*" class so that the classifier misclassified it as "*Capability*".

– **Type 2 Errors** *The misclassified defect instance contains a relevant annotation for a defect class other than the class it belongs to. This results in a false indicator* *of another class.* In FileZilla, approximately 22 % of the erroneous classifications by both SVM and NB belong to this type. In Company P, approximately 9 % of the erroneous classifications by SVM and 10 % of the erroneous classifications by NB belong to this type. Here is an example from FileZilla. "Currently, public key authentication thru Pageant (Putty Key Agent) works fine. But FZ still pops up a password dialog, even though the value from that is not used. It would be nice to first check Pageant, and see if key authorization works with the remote, before this dialog is brought up. Putty's current method for this user interface is a good sample to follow." This instance should belong to the *Requirements* class. However, the description contains several terms "password", "authentication" and "key" which are relevant annotations for *Integrity/Security*. Although "It would be nice to" is a relevant annotation of the *Requirement* defect class, the classifier still mislabeled it as *Integrity/Security* because more relevant annotations in *Integrity/Security* class were included in the instance.

Type 1 and 2 errors are also the main causes for the misclassifications on minor defect classes. If a minor class defect instance contains relevant annotations from a major class, the learning algorithm will be most likely to misclassify it. For example, "Checking the "always trust this certificate" is not working—I am prompted every time whether or not to accept the certificate." should belong to the *Integrity/Security* class, but it was actually misclassified as *Capability* because it contains "not working", a relevant annotation of *Capability* class. Although it contains a relevant annotation ("trust") for the *Integrity/Security* class, it still led to the misclassification. On the other hand, when a minor class defect instance is described very similarly to a major class, the learning algorithm is also likely to mislabel it. For example, "When the queue has many small files that are being downloaded, it is difficult to choose a particular file in the queue because the selection keeps moving to another file." is a defect instance belonging to the *Usability* class. But it is classified as *Capability* because "queue" is a prevailing expression existing in a great number of the *Capability* defects in the FileZilla defect list.

5.7 Effort saving and confidence building (RQ6)

5.7.1 Effort

We have calculated the approximate effort taken by the traditional manual ODC classification process and AutoODC on both industrial and larger open source defect datasets. It took approximately 18 h on average for an expert analyst with multi-year ODC experience to manually classify all 403 defect records in the defect report from Company P, not to say a person without any ODC experience. The effort needed to train a novice to perform ODC classification is huge. In contrast, the effort needed to train an ODC classifier is comparatively much smaller. Specifically, it took AutoODC (SVM) approximately 66 min to preprocess the defect report with 403 records for training and testing, and another 60 min to train all six SVM classifiers based on the training set (80 % records from the report) when executing AutoODC on Intel Core 2 Duo 3.3 GHz with 4 GB of RAM running Linux version 2.6.33.3. Once the training is

Table 16 Distribution of misclassified defect classes in FileZilla (AutoODC: Basic + Ext. 1, 2, 3, 4)

		Number of misclassified defects in each class							
	Percentage of overall misclassifications	Documentation	Installability	Reliability	Capability	Integrity/Security	Usability	Requirement	Performance
NB	22.5 %	0	6	44	105	25	18	46	34
SVM	24.8 %	0	8	30	172	19	10	46	25

Table 17 Distribution of misclassified defect classes in Company P (AutoODC: Basic + Ext. 1, 2, 3, 4)

		Number of misclassified defects in each class					
	Percentage of overall misclassifications	Reliability	Capability	Integrity/Security	Usability	Requirement	Performance
NB	17.1	0	40	0	0	14	8
SVM	19.3	0	53	3	0	15	7

Table 18 Analysis of misclassified defects by NB and SVM on Company P and FileZilla (AutoODC: Basic + Ext. 1, 2, 3, 4)

	NB and SVM Both correct	NB and SVM both wrong	NB correct SVM wrong	NB wrong SVM correct
Company P	307	49	27	18
FileZilla	855	164	114	85

done, AutoODC took 0.00014 s to classify one defect record on average. It is obvious that the cost reduction is significant using our automated tool. As far as the relevant annotations are concerned, the effort needed to manually mark up the relevant annotations is not as big as one may expect since this can be done in parallel with manual ODC classifications. On average, it only took the experts 0.79 min per record (5.3 h in total) to mark up the relevant words/phrases and another 4 h to identify which of these words/phrases are synonymous. Furthermore, the manual effort involved in preparing the training data is a one-time effort that cumulatively injects the experts' domain knowledge into the AutoODC learning component. Note that AutoODC (NB) only took approximately 1 min to train the classifier on 80 % of the 403 defect records.

As for the larger defect (issue) list extracted from the open source system FileZilla, it took approximately 1 week (40 h) for the expert analyst to manually classify 1,250 defects reported in FileZilla. Nevertheless, AutoODC (SVM) took approximately 3 h to preprocess the issue list with 1,250 reported defects, and another 3.5 h to train all eight AutoODC (SVM) classifiers based on the training set (80 % defect records from the issue list) on the same computer system. Once the training is done, AutoODC took 0.001 s to classify one defect record on average. On average, it took the experts 0.8 min per record (13.3 h in total) to mark up the relevant words/phrases and another 8 h to identify which of these words/phrases are synonymous. Again, the manual effort involved in preparing the training data is considered a one-time effort. Further, it only took AutoODC (NB) about 3 min to train the classifier on 80 % of the 1,250 defect records, which was even more efficient than training the SVM classifiers.

5.7.2 Confidence building

Because Precision and Recall are defined relative to human (manual) effort, higher values in these measures do not necessary translate into higher confidence. Note that our objective is not to replace human ODC generation, but to complement it by providing a comprehensive check against historical ODC classifications in a specific domain and experts' domain knowledge, which are provided as relevant annotations to AutoODC. Because AutoODC can perform automatically and independently, the resulting classifications serve as a second "independent" review that increases confidence when cross-checked with manual results. That is, AutoODC can help guide the manual ODC process by partitioning the effort into investigation sets. Analysts will only need to focus their further inspection effort on where AutoODC and analysts disagree. In addition, analysts only need to verify the AutoODCs classifications instead of generating them from scratch.

6 Discussion

This section discusses the limitations of applying AutoODC. Due to the characteristics of the defect report we were provided for training and testing, our evaluation was limited to the ODC "impact" attribute because the defects were mostly submitted by customers and end users who are more concerned with the defect's impact. Our results on precision, recall, F-measure and distribution of defects' impact types apply to the defect report of Company P and the defect (issue) list extracted from the defect tracker of the open source system FileZilla, but do not necessarily generalize to other data sets. In addition, the manual ODC classifications, relevant words/phrases and synonym identification are subject to experimenter bias as they are done by two co-authors of this paper.

7 Conclusion and future work

AutoODC when used in tandem with traditional manual approaches can largely reduce human effort and improve an analyst's confidence in ODC classification. It will have a significant impact on systematic defect analysis for software quality assurance. This paper seeks to improve ODC classification by presenting a novel framework for acquiring a defect classifier from relevant annotations to robustly analyze natural language defect descriptions.

Based on this framework, we developed AutoODC, a text classification approach for automated ODC classification that can be applied to other text classification problems in the software engineering domain. Our evaluation shows that AutoODC is accurate in classifying defects by the "impact" attribute. We have tested AutoODC using both SVM and NB. Rather than merely applying the standard machine learning framework to this task, we seek to acquire a better ODC classification system by integrating experts' ODC experience and domain knowledge into the learning process via proposing a novel Relevance Annotation Framework. We have evaluated AutoODC on both an industrial defect report from the social network domain and a larger defect list extracted from a publicly accessible defect tracker of an open source system FileZilla. AutoODC is a promising approach: not only does it leverage minimal human effort beyond the human annotations typically required by standard machine learning approaches, but it achieves an overall accuracies of 82.9 % by NB and 80.7 % by SVM on the industrial defect report and accuracies of 77.5 % by NB and 75.2 % by SVM on the larger, more diversified open source FileZilla defect list when using manual classifications as a basis of comparison.

7.1 Generalization of our approach to other SE tasks

We conclude by pointing out that none of the extensions we proposed to our basic defect classification system is specific to the ODC task, and hence our automated text classification approach based on SVM and NB provides a general framework that is potentially applicable to any text classification problem arising within and outside the Software Engineering domain. In particular, the usefulness of our proposed framework when applied to text classification tasks in software engineering should not be under-estimated. A variety of textual software engineering artifacts (i.e., documents, specifications) are produced daily in the software system development and evolution process. These textual artifacts contain valuable experience from system development and evolution history. Unfortunately, a large portion of the artifacts are unstructured and they usually contain a huge number of features determining what a specific artifact is concerned with. Manually assimilating and categorizing these artifacts based on their intentions would be extremely effort consuming as well as requiring substantial domain knowledge from human experts. Such real-world application examples include bug triage to decide what to do with an incoming bug report, risk report classification based on risk taxonomy, recovering missing tags for software engineering artifacts to facilitate artifact traceability management, categorizing software identifiers (e.g., packages, classes, methods, variables) to enhance program comprehension, etc. Our approach, when used in tandem with our novel Relevance Annotation Framework, helps bridge the gap between the human and the machine learner by harvesting human analysts' domain knowledge and eliciting their reasoning process to enhance the learner's accuracy on a text classification task.

7.2 Future work

We envisage future research in multiple directions: (1) We will investigate the potential of various combinations of learning algorithms (e.g., a combination of NB and SVM) to further improve the performance of AutoODC. (2) We will experiment with different matching algorithms including approximate matching and test their effects on the classification accuracy of AutoODC. (3) We would like to try different feature representations as learning algorithm inputs and compare their impacts on classification accuracy. (4) We will further investigate whether additional discourse connectives can help us discover defect categories under the Trigger attribute of the ODC framework. Finally, we plan to extend AutoODC to perform defect classifications on other ODC attributes and test it on other industrial and/or open source defect repositories.

References

Ahsan, S.N., Ferzund, J., Wotawa, F.: Automatic classification of software change request using multi-label machine learning methods. In: Proceedings of the 33rd IEEE Software Engineering, Workshop, pp. 79–86 (2009)

Aizawa, A.: Linguistic techniques to improve the performance of automatic text categorization. In: Proceedings of NLPRS-01, 6th Natural Language Processing Pacific Rim Symposium, pp. 307–314 (2001)

Asuncion, H.U., Asuncion, A.U., Taylor, R.N.: Software traceability with topic modeling. In: Proceedings of the 32nd International Conference on Software Engineering, pp. 95–104 (2010)

Bellucci, S., Portaluri, B.: Automatic calculation of orthogonal defect classification (odc) fields (2012). https://www.google.com/patents/US8214798. US Patent 8,214,798

Bridge, N., Miller, C.: Orthogonal defect classification: using defect data to improve software development. Softw. Qual. **3**(1), 1–8 (1998)

Caropreso, M., Matwin, S., Sebastiani, F.: A learner independent evaluation of the usefulness of statistical phrases for automated text categorization. In: Chin, A.G. (ed.) Text Databases and Document Management, Theory and Practice, pp. 78–102. Idea Group Publishing, Hershey (2001)

Chawla, N.V., Japkowicz, N., Kotcz, A.: Editorial: special issue on learning from imbalanced data sets. In: SIGKDD Exploration Newsletter, pp. 1–6 (2004)

Chillarege, R.: Orthogonal defect classification. In: Lyu, M. (ed.) Handbook of Software Reliability Engineering, pp. 359–400. McGraw-Hill, New York (1995)

Chillarege, R., Bhandari, I.S., Chaar, J.K., Halliday, M.J., Moebus, D.S., Ray, B.K., Wong, M.Y.: Orthogonal defect classification-a concept for in-process measurements. IEEE Trans. Softw. Eng. **18**(11), 943–956 (1992)

Chillarege, R., Biyani, S.: Identifying risk using odc based growth models. In: Proceedings of the 5th International Symposium on Software, Reliability Engineering, pp. 282–288 (1994)

Cubranic, D., Murphy, G.C.: Automatic bug triage using text categorization. In: Proceedings of the 6th International Conference on Software Engineering and Knowledge, Engineering, pp. 92–97 (2004)

Fellbaum, C.: WordNet: An Electronic Lexical Database. MIT Press, Cambridge (1998)

Gegick, M., Rotella, P., Xie, T.: Identifying security bug reports via text mining: an industrial case study. In: Proceedings of the 7th IEEE Working Conference on Mining Software Repositories, pp. 11–20 (2010)

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: an update. ACM SIGKDD Explor. Newslett. **11**(1), 10–18 (2009)

Huang, J., Czauderna, A., Gibiec, M., Emenecker, J.: A machine learning approach for tracing regulatory codes to product specific requirements. In: Proceedings of the 32nd International Conference on Software Engineering, pp. 155–164 (2010)

Hussain, I., Ormandjieva, O., Kosseim, L.: Automatic quality assessment of srs text by means of a decision-tree-based text classifier. In: Proceedings of the 7th International Conference on Quality Software, pp. 209–218 (2007)

Joachims, T.: Text categorization with support vector machines: learning with many relevant features. In: Proceedings of the 10th European Conference on Machine Learning, pp. 137–142. Springer, Berlin (1998)

Kiekel, P., Cooke, N., Foltz, P., Gorman, J., Martin, M.: Some promising results of communication-based automatic measures of team cognition. In: Proceedings of Human Factors and Ergonomics Society: 46th Annual Meeting, pp. 298–302 (2002)

Ko, A., Myers, B.: A linguistic analysis of how people describe software problems. In: IEEE Symposium on Visual Languages and Human-Centric, Computing, pp. 127–134 (2006)

Lamkanfi, A., Demeyer, S., Giger, E., Goethals, B.: Predicting the severity of a reported bug. In: Proceedings of the 7th IEEE Working Conference on Mining Software Repositories, pp. 1–10 (2010)

Lin, Z., Ng, H.T., Kan, M.Y.: A pdtb-styled end-to-end discourse parser. Nat. Lang. Eng. **20**, 151–184 (2014)

Lutz, R., Mikulski, C.: Empirical analysis of safety-critical anomalies during operations. IEEE Trans. Softw. Eng. **30**(3), 172–180 (2004)

Lutz, R., Mikulski, C.: Ongoing requirements discovery in high integrity systems. IEEE Softw. **21**(2), 19–25 (2004)

Ma, L., Tian, J.: Analyzing errors and referral pairs to characterize common problems and improve web reliability. In: Proceedings of the 3rd International Conference on Web, Engineering, pp. 314–323 (2003)

Ma, L., Tian, J.: Web error classification and analysis for reliability improvement. J. Syst. Softw. **80**(6), 795–804 (2007)

Mays, R., Jones, C., Holloway, G., Stundisky, D.: Experiences with defects prevention process. IBM Syst. J. **29**(1), 4–32 (1990)

Menzies, T., Lutz, R., Mikulski, C.: Better analysis of defect data at NASA. In: Proceedings of the 5th International Conference on Software Engineering and Knowledge, Engineering, pp. 607–611 (2003)

Menzies, T., Marcus, A.: Automated severity assessment of software defect reports. In: Proceedings of the International Conference on Software, Maintenance, pp. 346–355 (2008)

Ormandjieva, O., Kosseim, L., Hussain, I.: Toward a text classification system for the quality assessment of software requirements written in natural language. In: Proceedings of the 4th International Workshop on Software Quality Assurance, pp. 39–45 (2007)

Pandita, R., Xiao, X., Yang, W., Enck, W., Xie, T.: Whyper: towards automating risk assessment of mobile application. In: Proceedings of 22nd USENIX Security Symposium, pp. 527–542 (2013)

Polpinij, J., Ghose, A.: An automatic elaborate requirement specification by using hierarchical text classification. In: Proceedings of the 2008 International Conference on Computer Science and Software Engineering, pp. 706–709 (2008)

Porter, M.F.: An algorithm for suffix stripping. Program **14**(3), 130–137 (1980)

Rennie, J.D., Shih, L., Teevan, J., Karger, D.R.: Tackling the poor assumption of naive bayes text classifiers. In: Proceedings of International Conference on Machine Learning, pp. 616–623 (2003)

Romano, D., Pinzger, M.: A comparison of event models for naive bayes text classification. In: Proceedings of AAAI Workshop on Learning for Text Categorization, pp. 41–48 (1998)

Sebastiani, F.: Text categorization. In: Zanasi, A. (ed.) Texting Mining and Its Applications, pp. 109–129. MIT Press, Cambridge (2005)

Swigger, K., Brazile, R., Dafoulas, G., Serce, F.C., Alpaslan, F.N., Lopez, V.: Using content and text classification methods to characterize team performance. In: Proceedings of the 5th International Conference on Global, Software Engineering, pp. 192–200 (2010)

Tamrawi, A., Nguyen, T.T., AI-Kofahi, J., Nguyen, T.N.: Fuzzy set-based automatic bug triaging. In: Proceedings of the 33rd International Conference on Software Engineering, pp. 884–887 (2011)

Thung, F., Lo, D., Jiang, L.: Automatic defect categorization. In: Proceedings of 19th Working Conference on Reverse Engineering, pp. 205–214 (2012)

Tong, S., Koller, D.: Support vector machine active learning with applications to text classification. J. Mach. Learn. Res. **2**, 45–66 (2001)

Tsochantaridis, I., Hofmann, T., Joachims, T., Altun, Y.: Support vector machine learning for interdependent and structured output spaces. In: Proceedings of the 21st International Conference on Machine Learning, pp. 104–112 (2004)

Vapnik, V.: The Nature of Statistical Learning. Springer, Berlin (1995)

Yang, C., Hou, C., Kao, W., Chen, I.: An empirical study on improving severity prediction of defect reports using feature selection. In: Proceedings of the 19th Asia-Pacific, Software Engineering Conference, pp. 240–249 (2012)

Zheng, J., Williams, L., Nagappan, N., Hudpohl, J.: On the value of static analysis tools for fault detection. IEEE Trans. Softw. Eng. **32**(44), 240–253 (2006)