Analysis of the Characteristics and Causes of Underestimated Bug Reports*

Anna Gromova

Exactpro Systems, Moscow, Russia
anna.gromova@exactprosystems.com
https://exactpro.com

Abstract. The bug-fixing process requires software maintenance resources. Usually, defects are submitted, fixed and closed, but sometimes they have to be reopened because of a change of resolution. It happens when a defect was evaluated incorrectly at the beginning. This problem can increase maintenance costs and software quality in general.
In this paper, we investigate the characteristics of such defects and their bug reports and call them "underestimated". Our research is based on general statistical indicators and text descriptions of defect reports. We propose using different methods of feature selection and ranking in order to reveal the significant terms of such defects. The top of significant terms of the underestimated bug reports can help to find the root causes of such life cycles.

Keywords: Defect management · Bug report · Feature selection

1 Introduction

Defect management is an essential part of improving the technical stability of software. Usually, software companies use bug-tracking systems (BTS) in order to manage defects. Structured information about defects is a big advantage of a BTS, where a bug is represented as a set of attributes. Gathering data from bug reports allows us to accumulate statistics, we also use these data for predictions and analysis.

The knowledge of defect statistics and the defect management strategy helps to create an effective approach to defect prevention because it can help to reduce bug migration into the later stages of development [17]. So, the accumulated statistics of bug reports can reveal possible problem aspects. For example, reopened bugs belong to a problem area like this because they take considerably longer to resolve [16]. However, for the purposes of this paper, we are investigating not just the reopened bugs, but also the defects which were once rejected, considered a non-defect or non-fixable, and now having a "Done" or a "Fixed"

resolution. It means that they could have been evaluated incorrectly, setting a precedent for future misjudgments. Thus, they need to be analysed as a potential problem of software development. In subsection 4.1, we demonstrate the calculations for several indicators, such as time to resolve, count of comments, priority, etc. that prove this assumption.

We would also like to notice that our work is devoted to revealing the reasons of underestimated defect reports more than their future classification. Our goal is to understand the nature of such bug reports via the feature selection and ranking methods. Thats why we mainly investigated the top list of terms.

We claim the following contribution in this work:

- We reveal special characteristics of underestimated bug reports.
- We propose using different methods of feature selection and ranking for determining the most significant terms of underestimated bug reports.
- We conduct an analytical study to investigate the potential causes of the initial resolution of such bug reports via the most significant terms.

The remainder of this paper is organized as follows: in Section 2, we present an overview of the related work; in Section 3, we describe the structure of a defect report; in Section 4, we outline the process of clustering. Further, in Section 5, we present the results of the experimental evaluations of this technique; and Section 6 lists our conclusions.

2 Related work

There are many researchers who deal with analysis of bug reports which have certain specifics. They investigate the nature of these specifics, the root causes of their occurrence and ways to predict them.

Zaineb and Manarvi analysed the reasons of bug rejection in order to decrease the possibility of submission of invalid bug reports [20]. They discovered some causes of bug rejection and their impact on testing efficiency. Zimmerman et al. [21] propose to predict defect reopening. They analyse comments, description, time to fix and the components describing the defects. Shihab et al. also investigate the problem of reopened bug reports [16]. They used the decision tree in order to predict whether a bug will be reopened after its closure. Karim et al [13]. investigated the key features of high-impact bug reports (HIB). HIBs are defect reports which can significantly affect the software development process and product quality. The researchers considered several types of HIBs and discovered the most frequent features provided by submitters in such bugs. These features include the observed behaviour, the expected behaviour and code examples. Similar features were detected by Chaparro et al. [3]. They analysed the observed behaviour, the expected behaviour and steps to reproduce and proposed linear Support Vector Machines to classify the description of bug reports. Their text classification is based on N-grams.

Gegick, Rotella and Xie proposed to classify bug reports into security and non-security bug reports [7]. They analyzed the description of defects and created

three lists: a start list (includes special terms from security bug reports), a stop list (the classic list of stop words) and a synonym list (includes security-related verbiage). In order to reduce the size of a term-by-document frequency matrix, they used SVD. Peters et al. [15] also investigated security bug reports. They proposed a "FARSEC" framework that is used for filtering and ranking bug reports for reducing the presence of security-related keywords. The framework is also able to identify and remove non-security bug reports with security-related keywords. They compared several machine learning algorithms such as random forest, Naive Bayes, logistic regression and multilayer perceptron. Goseva-Popstojanova and Tyo also analysed the problem of security bugs as crucial for software quality, but they used supervised and unsupervised approaches[8].

The problem of classifying issues into defects and non-defects is very popular too. Antoniol et al. [1] analyzed text attributes of bug reports. They compared the results of three classifiers: logistic regression, decision tree, and Naive Bayes. So, the research helped to classify the issues and determine the discriminating terms. Herzig et al. [11] investigated this problem as well. However, they estimated misclassification, i.e. bias, in bug prediction models confusing bugs and features. The researchers proposed manual data validation in order to improve future studies. Terdchanakul et al. [18] also identified if the description of the issue corresponds to a bug or not. They used N-grams for text classification and built classification models with the logistic regression and the random forest methods.

This review of the related work demonstrates the importance of understanding the nature of different types of defects. The aforementioned researchers investigate various specific types of bugs and their potential life cycle stages. In our work, we propose to consider the problem of underestimated defects as a special case of the reopened bugs problem. In this paper, we investigate the submitted bug reports which were reopened because, at some point, they were understood and evaluated incorrectly. We aim to reveal the causes of such situations in order to prevent them in the future and create possible recommendations.

3 Background

Each bug report has its own life cycle. A life cycle depends on the characteristics of the software development cycle, the proprietary and the domain aspects of the project. During the life cycle, a defect report's priority, status, etc. can change.

For this research, we only considered closed and resolved defects because only such bugs have the values of *Resolution, Time to resolve, Count of attachments, Count of comments,* etc., known for a fact. For the defects that have not been closed or resolved, the values of these attributes are indefinite. We selected the defects where Resolution has such values as "Done" or "Fixed" because they look as ordinary. But then, we split them into two categories. The first category — called "type 1" — includes bug reports that have never been reopened because of a change of resolution. It means that they have never been rejected or considered a non-defect or non-fixable. The second category — called "type 2" — includes

the defect reports that had an alternative resolution before they got a “Done” or a “Fixed” resolution. These alternative resolutions could have such values as “Rejected”, “Won't Fix”, “Not defect”. In this paper, we called the bug reports of type 2 “underestimated”.

We propose to compare the two types according to the following metrics:

- time to resolve as an indicator of how expensive the defect report is [12],
- count of comments and count of attachments because their abundance can be an indicator of an insufficient defect description [12],
- percentage of “Critical” and “Blocker” priority ,
- the length of description.

Thanks to this comparison, we can reveal the distinguishing characteristics of underestimated defect reports.

4 Approach

4.1 Objects

We extracted 45,341 bug reports from three different project communities on JIRA, a popular bug-tracking system. They include open-source projects of JBOSS [22], Jenkins [23] and Sakai [24]. The comparative analysis of defect reports of both types for all projects is presented in Table 1.

As Table 1 shows, defect reports of type 2 have distinctive characteristics. The bugs of type 2 require more time for fixing than the bugs of type 1. They also have a larger number of collateral comments and attachments. These characteristics may be suitable for reopened defects or may not be suitable, such as in the case of trivial reasons of reopening like addition of labels. So we prove that defects of type 2 are “expensive-to-resolve” since they require a lot of human input and time. Therefore it is important to investigate and prevent them.

According to Table 1, defect reports of type 2 have “Blocker” to “Critical” priorities, just like type 1. It means that underestimated defects are important because they can lead to a situation when a Critical or a Blocker bug can persist in the system for a long time and undermine software quality.

The comparison of description lengths shows that the description of bugs of type 2 is more complicated than “scarce text”. It means that they can have a detailed description, but, for some reasons, it was evaluated incorrectly.

We propose to analyse the description of bug reports as a source of answers as to why these defects became underestimated.

4.2 Text preprocessing

The description of defect reports is in text format, so it needs to be transformed via natural language processing methodologies. We made the following steps:

- tokenization that chops the text into words,

Table 1. Defect reports information

	JBOSS	Jenkins	Sakai
Number of considered bugs	11,965	14,430	18,946
Number of type 1	11,848	14,280	18,763
Resolution of type 2	Won't Fix, Reject	Won't Fix, Not a defect	Won't Fix, Non-issue
Number of type 2	117	150	183
Time to resolve of type 1: min / max/ mean	0 / 3086 / 52.619	0 / 3762 / 180.142	0 / 3486 / 86.898
Time to resolve of type 2: min / max/ mean	0 / 1847 / 93.394	0 / 2791 / 367.353	0 / 4708 / 378.525
Count of comments of type 1: min / max/ mean	0 / 92 / 2.772	0 / 174 / 5.665	0 / 80 / 4.426
Count of comments of type 2: min / max/ mean	0 / 35 / 4.324	1 / 127 / 14.7	1 / 66 / 8.749
Count of attachments of type 1: min / max/ mean	0 / 22 / 0.324	0 / 20 / 0.415	0 / 47 / 0.639
Count of attachments of type 2: min / max/ mean	0 / 8 / 0.451	0 / 23 / 1.247	0 / 13 / 0.836
Percentage of Blocker / Critical of type 1	7 % / 12%	9 % / 12%	10 % / 13%
Percentage of Blocker / Critical of type 2	4 % / 8%	12 % / 13%	6 % / 16%
Mean description length of type 1	4982.13	2082.092	971.113
Mean description length of type 2	6669.768	2550.617	726.328

– removal of stop-words list that was expanded with names of months, week days, the submitter and assignee's names, parts of logs, stack traces, etc.,
– stemming that maps related words to their basic form and helps to reduce the inflectional forms.

Then, we built a "Bag of words" model. This vector model takes into account the number of occurrences of each term, rather than the exact order of the terms. Every bug report is presented as a vector of n terms. A set of bugs is presented as corpora or matrix $n \times m$ where n is the number of all terms, and m is the number of all documents. If a term occurs in the bug-report, its value in the vector is non-zero. We used TF-IDF weighting for computing these values [14].

$$TFIDF = TF(t,d) \cdot IDF(t,D) \tag{1}$$

$$TF(t,d) = \frac{freq(t,d)}{max_{w \in d} freq(w,d)} \tag{2}$$

$$IDF(t,D) = log_2(\frac{|D|}{d \in D : t \in D}) \tag{3}$$

Where *freq(t,d)* is term frequency, i.e. the number of times that term t occurs in document d; $max_{w \in d} freq(w,d)$ is the maximal frequency of any term in document d; $d \in D : t \in D$ is the number of documents containing t; D is the corpus - the total document set [14].

4.3 Proposed techniques

We used the following methods of feature selection and ranking:

1) Chi-Squared is the common statistical test that measures the divergence from the expected distribution, if one assumes that feature occurrence is actually independent of the class value [5].

2) Recursive feature elimination (RFE) is a recursive process that ranks features according to some measure of their importance [10]. At each iteration, the importance of each feature is measured, and the least relevant one is removed. The recursion is needed because, for some measures, the relative importance of each feature can change substantially when it's evaluated against a different subset of features during the stepwise elimination process. In this work, the feature ranking method is based on the measure of the variables' importance given by SVM [4].

3) Random forest is a method that builds an ensemble model of decision trees from random subsets of features and bagged samples of the training data [2]. Each tree grows on an independent bootstrap sample from the training data. For each node, it is necessary to select m variables at random out of all M possible variables (independently for each node) and find the best split on the selected m variables. Random forest classifiers can reveal feature importance, determining how much each feature contributes to class prediction [19].

4) Logistic regression is a classifier where the dependent variable is dichotomous (binary). The logistic regression model is as follows:

$$\pi(x_1, \dots x_n) = \frac{e^{\beta_0+\beta_1*x_1+\dots+\beta_n*x_n}}{1+e^{\beta_0+\beta_1*x_1+\dots+\beta_n*x_n}} \quad (4)$$

Where x_i are the characteristics describing the model, $\pi \in [0; 1]$ is a value on the logistic regression curve [6]. A regression coefficient describes the size and direction of the relationship between a predictor and the response variable. Positive coefficients make the event more likely and negative coefficients make the event less likely. A coefficient with a value near 0 implies that the effect of the predictor is small.

5 Results

In the original bug report dataset, we marked bugs as 0 if they belong to type 1 and 1 if they belong to type 2. We matched each indexed defect report with its class {0, 1}. This column was used by the feature selection and ranking techniques.

We built the top of the most significant terms with the aforementioned feature selection and ranking techniques that include chi-square, recursive feature elimination, features importance of random forest and coefficients of logistic regression. The top 15 of the most significant terms of JBOSS, Jenkins and Sakai projects is presented in Tables 2, 3, and 4.

Table 2. The top of significant terms of JBOSS

Chi2	RFE	Random Forest	Logistic regression
'cast',	'busi',	'import',	'cast',
'materi',	'capabl',	'color',	'request',
'osgi',	'connector',	'normal',	'event',
'busi',	'consequ',	'classexternallink',	'jar',
'constructor',	'day',	'lineheight',	'bundl',
'bundl',	'determin',	'condit',	'error',
'vdb',	'download',	'fonttyl',	'busi',
'network',	'end',	'error',	'osgi',
'comment',	'facet',	'file',	'open',
'request',	'includ',	'event',	'materi',
'jar',	'later',	'like',	'comment',
'spec',	'long',	'comment',	'constructor',
'lookup',	'network',	'fonteight',	'connect',
'redirect',	'osgi',	'properti',	'server',
'lot'	'perform'	'consol',	'vdb'

Having analysed the received results, we propose to split them into the following groups of terms:

Table 3. The top of significant terms of Jenkins

Chi2	RFE	Random Forest	Logistic regression
'stdout',	'avoid',	'job',	'git',
'ssl',	'capac',	'build',	'server',
'dynam',	'correspond',	'error',	'password',
'testsuit',	'detect',	'configur',	'stdout',
'password',	'ensur',	'password',	'error',
'wrapper',	'general',	'run',	'setup',
'certif',	'head',	'use',	'copi',
'stderr',	'increment',	'document',	'document',
'git',	'introduc',	'need',	'perforc',
'larg',	'jdk',	'testsuit',	'dynam',
'emailtext',	'listen',	'jenkin',	'upstream',
'perforc',	'previous',	'log',	'way',
'upstream',	'provis',	'poll',	'avail',
'setup',	'servic',	'follow',	'findbug',
'gitssh'	'strang'	'long'	'log'

Table 4. The top of significant terms of Sakai

Chi2	RFE	Random Forest	Logistic regression
'recommend',	'administr',	'user',	'recommend',
'desir',	'app',	'tool',	'appear',
'idea',	'applic',	'appear',	'addit',
'addit',	'breadcrumb',	'error',	'edit',
'retract',	'exit',	'recommend',	'resourc',
'exit',	'explicit',	'question',	'call',
'pool',	'role',	'use',	'desir',
'random',	'process',	'classexternallink',	'pool',
'uniqu',	'recommend',	'make',	'mean',
'person',	'retract',	'chang',	'entri',
'app',	'situat',	'info',	'inform',
'font',	'stay',	'click',	'idea',
'portfolio',	'trunk',	'screen',	'gradebook',
'audio',	'write',	'follow',	'creat',
'edit'	'uniqu'	'list'	'requir'

1) "Prejudiced term" are the terms that are connected with biased wording in the software bug description. The text seems to be full of subjective assessment or personal impression of the submitter. So the defect report may be considered a "fancy" or just a wild guess. Examples of such terms are: *idea, recommend, desir, larg, strange, long, comment,* etc. Below are some samples of use of these terms extracted from the defect reports under analysis:
I think the idea is to ...
I recommend ...
it would be more desirable ...
unprofessional comments ...
goes through after long time ...
large logfile ...
there is something strange with ...

It is very important to isolate such terms and avoid them in the future defects because they obscure the facts with subjectivity and increase the likelihood of preserving a bug with a high or even critical and/or blocker priority in the system.

Some terms may seem as standard forms of politeness. But sometimes a bug report is overloaded with such "terms of politeness", which can divert the developers focus from the software problem itself.

2) "False friend" terms are the terms that might seem useful and look organic in a technical text, but, surprisingly, in the context of defect reports, they can decrease the transparency and make the meaning ambiguous. The examples of such terms are: *error, perform, appear, document, configur,* etc. Some samples of use of these terms extracted from the defect reports under analysis are presented in Table 5.

Table 5. The terms of the second group

Example of use of these terms	Clarification
The documentation claims that ...	Ambiguous interpretation of "documentation" leads to a misunderstanding between the developer and the submitter.
Add the name again and continue but the name does not appear ...	The lack of details about "non-appearance" makes the bug non-reproducible.
When trying to perform operation, the exception is thrown ...	The lack of details about what is being performed can confuse the developer.
Some plugins fail to startup with the following error ...	The absence of conditions and details leads to a biased assessment of the root cause.

This group of terms is especially dangerous. The submitter describing a software defect uses these terms and overlooks the details because he or she thinks that the description is comprehensive.

3) Domain-related terms. Examples of such terms are: *osgi, gitssh, retract, gradebook, bundl,* etc. Knowing them is very useful because it gives an opportunity to reveal potential areas of testing [9] where defect reports have a high probability of being underestimated.

It is important to mention that these groups of terms can overlap. It means that some areas of testing can be rather complicated, and there are more possible cases for misunderstanding and underestimating a potential problem. For example, the submitter describes user actions which are connected to one of the areas of testing. He or she notices errors or exceptions. However, due to the complexity of the area of testing and the fact that some necessary details are missing, it is difficult for a developer to understand the cause of these results: it is not obvious whether it is a possible defect or just an erroneous chain of submitter's actions.

We have compared the methods of feature selection and ranking and noticed that chi-square is prone to place the "prejudiced" terms at the top, and the feature ranking by random forests is prone to place the "false friend" terms at the top. So they can be useful for revealing the words that can create a situation where some facts or details are omitted.

We also compared the methods of feature selection and ranking in order to check their accuracy. We used the cross-validation technique for this task. The received results are presented in Table 6. We discovered that the cases without feature selection have lower accuracy values than others. According to Table 6, Random Forest has the highest accuracy.

Table 6. The accuracy comparison

Project	Without using feature selection methods	Chi2	RFE	Random Forest	Logistic Regression
JBOSS	0.8	0.82	0.81	0.96	0.88
Jenkins	0.77	0.83	0.82	0.98	0.88
Sakai	0.69	0.7	0.77	0.96	0.87

6 Conclusion

This paper is devoted to the problem of underestimated bug reports. These are reports where the resolution of the described defect was changed from a potentially incorrect one, such as "Reject", "Not a defect", etc., to an ordinary one of "Done" or "Fixed".

We have revealed the specifics of such bug reports. They are long to resolve and have a large number of collateral comments and attachments. They can be considered as potentially problematic bugs, i.e. the defects that require human and time resources.

We have proposed several methods of feature selection and ranking in order to build the top of the most significant terms. We used chi-square, recursive feature elimination, features importance of random forest and coefficients of logistic regression.

We compared different methods of feature selection. We found out that the cases without feature selection have lower accuracy values than others and Random Forest has the highest accuracy among the methods.

We have analysed the received tops of terms and proposed to split them into three groups. The first group - "prejudiced" terms - includes terms with subjective assessment. The second group - "false friend" terms - consists of terms with a potentially dangerous lack of details. The third group - domain-related terms - includes terms associated with an area of testing. The analysis of these groups can help understand the key problems of underestimated defects as well as prevent them from occurring.

In the nearest future, we plan to analyse each potentially problematic resolution separately. We also plan to compare underestimated bug reports with defects that have the final resolution of "Rejected", "Won't Fix", etc. It can provide a better picture of the specifics of such defect reports. Consequently, it can help to generate recommendations in order to reduce the occurrence of underestimated bugs.

References

1. Antoniol, G., Ayari, K., Di Penta, M., Khomh, F., Guéhéneuc, Y.-G.: Is it a bug or an enhancement?: A text-based approach to classify change requests. In Proc. 2008 Conf. Center for Adv. Studies Collaborative Res.: Meeting Minds, 2008, ser. CASCON 08, Article No. 23. New York, NY, USA: ACM, pp. 304-318
2. Breiman, L.: Random Forests. Journal Machine Learning, vol. 45(1), pp. 5–32 (2001)
3. Chaparro, O., Lu, J., Zampetti, F., Moreno, L., Di Penta, M., Marcus, A., Bavota, G. and Ng, V.: Detecting Missing Information in Bug Descriptions. Proceedings of the 11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp. 396–407, 2017
4. Durgesh, K.S., Lekha, B.: Data classification using support vector machine. Journal of Theoretical and Applied Information Technology 12 (1), 17 (2010)
5. Freund, R.J., Wilson, W.J.: Regression Analysis: statistical modeling of a response variable. San Diego: Academic Press (1998)
6. Forman, G.: An extensive empirical study of feature selection metrics for text classification. Journal of Machine Learning, pp. 1289-1305 (2003)
7. Gegick, M., Rotella, P., Xie, T.: Identifying security bug reports via text mining: An industrial case study. In Proc. 7th IEEE Working Conf. Mining Software Repositories (MSR), May 2010, IEEE Computer Society, 11-20
8. Goseva-Popstojanova, K. and Tyo, J.: Identification of Security Related Bug Reports via Text Mining Using Supervised and Unsupervised Classification. 2018 IEEE International Conference on Software Quality, Reliability and Security (QRS), Lisbon, 2018, pp. 344–355.

9. Gromova, A. : Defect Report Classification in Accordance with Areas of Testing. In: Itsykson V., Scedrov A., Zakharov V. (eds) Tools and Methods of Program Analysis. TMPA 2017. Communications in Computer and Information Science, vol 779, pp. 38–50
10. Guyon, I., Weston, J., Barnhill, S., and Vapnik, V.: Gene selection for cancer classification using support vector machines. Mach. Learn., 46(1-3), pp. 389422, 2002.
11. Herzig, K., Just, S., and Zeller, A.: It's not a bug, it's a feature: how misclassification impacts bug prediction. In Proceedings of the 2013 International Conference on Software Engineering (ICSE '13). IEEE Press, Piscataway, NJ, USA, 392-401.
12. Hooimeijer, P., Weimer, W.: Modeling bug report quality. In: ASE 07: Proceedings of the twenty-second IEEE/ACM International Conference on Automated Software Engineering , pp. 34–43 (2007).
13. Karim, M.R., Ihara, A., Yang, X., Iida, and H., Matsumoto, K.: Understanding key features of high-impact bug reports. In: 2017 8th International Workshop on Empirical Software Engineering in Practice (IWESEP), 53-58. IEEE (2017)
14. Manning, C.D., Raghavan, P., Schutze, H.: Introduction to Information retrieval. New York: Cambridge University Press (2008)
15. Peters, F., Tun, T., Yu, Y., Nuseibeh, B.: Text filtering and ranking for security bug report prediction. IEEE Transactions on Software Engineering. 2017 Dec 27.
16. Shihab, E. , Ihara, A., Kamei, Y., Ibrahim, W. M., Ohira, M., Adams, B., Hassan, A. E., and Matsumoto, K.-I.: Predicting re-opened bugs: A case study on the eclipse project. in Proc. 2010 17th Working Conf. Reverse Eng., 2010, ser. WCRE 10. Washington, DC, USA: IEEE Computer Society, pp. 249-258
17. Suma, V., Nair, TR.: Defect management strategies in software development. arXiv preprint arXiv:1209.5573. 2012 Sep 25.
18. Terdchanakul, P. , Hata, H., Phannachitta, P., and Matsumoto, K.: Bug or not? bug report classification using N-gram IDF. Proceedings of the 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 534–538, Sept 2017
19. Touw, W.G., Bayjanov, J.R., Overmars, L., Backus, L., Boekhorst, J., Wels, M. and van Hijum, S.A.: Data mining in the Life Sciences with Random Forest: a walk in the park or lost in the jungle? Briefings in bioinformatics 14, no. 3 (2012): 315–326.
20. Zaineb, G and Manarvi, I. A.: Identification and analysis of causes for software bug rejection with impact over testing efficiency. International Journal of Software Engineering & Application, 2(4), 2011, pp. 71–82
21. Zimmermann, T., Nagappan, N., Guo, P.J., and Murphy, B.: Characterizing and predicting which bugs get reopened. In Proceedings of the 34th International Conference on Software Engineering, pp. 1074–1083. IEEE Press, 2012.
22. JBossDeveloper, [Online]. Available at: https://developer.jboss.org/welcome Accessed 27 jan 2019
23. Jenkins documentation, [Online]. Available at: https://jenkins.io/doc/ Accessed 27 jan 2019
24. Sakai documentation, [Online]. Available at: https://www.sakaiproject.org/documentation Accessed 27 jan 2019