2010 34th Annual IEEE Computer Software and Applications Conference Workshops

Software Fault Prediction Models for Web Applications

Le Truong Giang
College of Information Science and Technology
Korea Advanced Institute of Science and Technology
Daejeon, Korea
Email: giang@se.kaist.ac.kr

Dongwon Kang
College of Information Science and Technology
Korea Advanced Institute of Science and Technology
Daejeon, Korea
Email: dwkang@se.kaist.ac.kr

Doo-Hwan Bae
College of Information Science and Technology
Korea Advanced Institute of Science and Technology
Daejeon, Korea
Email: bae@se.kaist.ac.kr

***Abstract*—Our daily life increasingly relies on Web applications. Web applications provide us with abundant services to support our everyday activities. As a result, quality assurance for Web applications is becoming important and has gained much attention from software engineering community. In recent years, in order to enhance software quality, many software fault prediction models have been constructed to predict which software modules are likely to be faulty during operations. Such models can be utilized to raise the effectiveness of software testing activities and reduce project risks. Although current fault prediction models can be applied to predict faulty modules of Web applications, one limitation of them is that they do not consider particular characteristics of Web applications. In this paper, we try to build fault prediction models aiming for Web applications after analyzing major characteristics which may impact on their quality. The experimental study shows that our approach achieves very promising results.**

***Keywords*-Fault prediction, software metrics, Web applications**

I. Introduction

In recent years, Web applications have grown so quickly and they are playing an extremely important role in human life. Many everyday activities of people are now happening on the Internet. As a result, assuring quality of Web applications is very important since problems which happen to them can affect a lot of users and cause a financial loss. For example, a failure with the online shopping website Amazon.com in 1998 put the site offline for several hours, with an estimated cost of $400,000 [1]. Therefore, it is necessary to build high quality Web applications. However, Web applications have their own characteristics, which are different from traditional desktop applications [2]. Consequently, researchers and developers need a new approach and methodology so that they can develop Web applications with high quality [3], [4].

To improve software quality, many fault prediction models have been constructed to predict which software modules are likely to be faulty during operations. Such models are very useful as we can apply them to raise the effectiveness of software testing activities and reduce project risks. Before testing phase, if we can predict which software modules are most likely to be faulty, then we can focus testing resources and quality improvement efforts on these parts to save time and money. As a result, this is beneficial for software projects developed under a time-to-market pressure such as Web applications [2].

Although current fault prediction models can be applied to predict faulty modules of Web applications, one limitation of them is that they do not consider particular characteristics of Web applications. With the prevalence of Web applications in human life, we see that it is necessary to construct fault prediction models for this type of application. In this paper, we will introduce fault prediction models for Web applications using fault predictors related to their characteristics. The experimental analyses show that our models achieve better results when compared with other models using only popular fault predictors.

The rest of this paper is organized as follows. In Section II, we give an overview of the related work. In Section III, we discuss some characteristics of Web applications and common types of faults in the Web application domain. In Section IV, we introduce our research methodology, including selected fault predictors for Web applications and classification techniques used for fault prediction. The case study is reported in Section V. Finally, some conclusions and future work are presented in Section VI.

II. Related work

A systematic literature review of software fault prediction models can be found in [5]. So far a wide range of fault prediction models have been proposed to investigate the relationship between software metrics and the fault-proneness of software modules. There are two common types of software metrics which can be used as fault predictors to predict faulty modules: code metrics and change metrics [6].

Code metrics are measured based on static attributes of source code of the software itself such as LOC, complexity

978-0-7695-4105-1/10 $26.00 © 2010 IEEE
DOI 10.1109/COMPSACW.2010.19

metrics, object-oriented metrics, etc. Several researchers [7], [8] chose LOC and complexity metrics as fault predictors when constructing fault prediction models. Gyimóthy *et al*. [9] examined Mozilla, a large open source software product. They constructed fault prediction models to investigate the effect of LOC and object-oriented software metrics on the fault-proneness of classes.

Change metrics are measured by using information collected during software development process such as code delta, code churn, developer information, age of a module, etc. Moser *et al.* [6] proposed some change metrics (e.g., code delta, code churn, etc.), and they tried to compare fault prediction models using these change metrics with those using code metrics. By using a data set collected from the Eclipse software project, they found that change metrics were better than code metrics. Graves *et al.* [10] also collected change history data (e.g., number of developers working for a module during coding phase, age of a module, etc.) for fault prediction.

Some researchers used both code metrics and change metrics for fault prediction [6], [11], [12]. In some cases, using a combination of metrics brought a better result while in other cases, it did not.

In our approach, we want to predict whether each module in a Web application is faulty or not (two classes: faulty and non-faulty). Although many fault prediction models have been developed, no models aims at fault prediction for Web applications considering their particular characteristics. This is the main motivation for us to do this research.

III. Web applications

Web applications are growing so large, both in size and complexity. Although many Web development methodologies and tools have been developed, existing Web applications are still generally in poor quality, a situation referred to as the Web crisis [13]. In order to improve quality of Web applications, we need to know major characteristics which may affect their quality and common types of faults in the Web application domain. Understanding the origins of faults occurring in Web applications will help us predict them more accurately.

A. Characteristics of Web applications

Based on software engineering literature, we figure out three major characteristics which may impact on the quality of Web applications as listed below:

- *Integration of many state-of-the-art technologies [2]*
  Web technologies are changing so quickly. We are now living in the era of Web 2.0, which facilitates interactive information sharing and collaboration on the World Wide Web. However, recently, public starts to pay attention to a new Web generation (i.e., Web 3.0) with more improvements. The fast-changing Web technologies pose many challenges for software developers. To satisfy customers' demands, they are required to have advanced skills in a variety of common Web technologies such as server-side Web programming languages, Web server, HTML/XHTML/DHTML, JavaScript, CSS (Cascading Style Sheets), XML, Ajax (Asynchronous JavaScript and XML), RSS (Really Simple Syndication), network protocols, Web browser, etc. If Web developers do not have a deep knowledge and experience with these technologies, it will be very difficult for them to develop high quality Web applications.
- *High-priority requirement for user interfaces*
  Web applications have users from all over the world. Each of them uses different Web browsers running on different operating systems. Therefore, Web applications need to be developed so that they can display and work well in all popular environments. However, this task is difficult and error-prone since Web browsers support Web standards (e.g., HTML, CSS, JavaScript, etc.) at different levels. For example, a Web page may run smoothly on Internet Explorer but not on Firefox and vice versa. Another difficulty is that among all users using the same Web browser, they may use different versions. Developers also have to support all common versions of one Web browser. Therefore, developing user interfaces for Web applications is really a hard problem.
- *Hyperlink-based structure*
  Web applications provide a huge source of information, where people can find almost information they want in daily life. With Web applications, users can create, view and manage many kinds of contents (e.g., text files, pictures, audio/video files, etc.). In addition, information on Web applications are added and updated very frequently. Since there are a lot of contents, at one moment, users cannot view fully all needed information. However, users can browse all contents through a hyperlink-based structure, which is commonly complex since there are a lot of links on each website. As a result, working with hyperlinks when developing Web applications is also difficult and error-prone.

B. Common types of faults in the Web application domain

To propose fault prediction models for Web applications, along with analyzing their characteristics, we also need to consider common types of faults in the Web application domain. Li *et al.* [14] examined several websites and analyzed their faults during operations. Their results indicated that faults related to links and graphical user interfaces contributed a lot to the total number of faults. Several other surveys also found that broken link is one of the most frequently cited problems by Web users [15]. We can see that these above findings are consistent with our analysis of characteristics of Web applications which may impact on their quality. Web applications require a high priority

for user interfaces and the hyperlink structures in websites are usually complex. Therefore, working with user interfaces and hyperlinks when developing Web applications is difficult and error-prone.

IV. Research methodology

In this section, we will introduce fault prediction models for Web applications by using our proposed fault predictors and common classification techniques.

A. Selected fault predictors for Web applications

Based on the results of Section III, we can propose some fault predictors related to characteristics of Web applications to predict faulty Web modules. In our study, a Web module is a page chunk which implements one small functionality in the Web system [16]. For example, the Poll module allows users to select their preferred choice and view the result of a survey. By using the Comment module, users can express their opinions about one article and read comments from other people. Another example is the FAQs module, where users can find the most frequently asked questions when using a website.

Table I gives a list of six software metrics used as fault predictors in our approach. We arranged them in three categories:

- *User interface related metric*
  We defined two metrics related to JavaScript and two metrics related to CSS. JavaScript is the scripting language, which allows the development of enhanced user interfaces and dynamic websites. CSS is a simple mechanism for adding styles (e.g., fonts, colors, spacing, etc.) to Web pages. JavaScript and CSS are two standard Web technologies used to display Web contents and interact with users [17]. We defined these four metrics related to JavaScript and CSS to reflect faults which happen to user interfaces in the Web application domain.
- *Hyperlink related metric*
  In our approach, we defined one metric related to hyperlink. As we discussed in Section III, working with hyperlinks when developing Web applications is error-prone. So we defined this metric to reflect faults which happen to hyperlinks on websites.
- *Size metric*
  LOC, a very basic and well-known size metric, is commonly suggested to be used in combination with other kinds of software metrics for fault prediction [9], [18]. Therefore, in our study, we combined our own defined metrics with LOC for fault prediction.

Table I
List of our proposed software metrics to be used as fault predictors for Web applications

Metric name	Type	Description
LOJ	User interface related metric	The number of lines of JavaScript defined in the source code of a module
NOFJ	User interface related metric	The number of functions of JavaScript defined in the source code of a module
LOCSS	User interface related metric	The number of lines of CSS defined in the source code of a module
NOIC	User interface related metric	The number of ids and classes of CSS defined in the source code of a module
NOH	Hyperlink related metric	The number of hyperlinks defined in the source code of a module
LOC	Size metric	The total number of lines of code of source code files in a module

B. Classification techniques

So far many classification techniques have been developed to help researchers build fault prediction models. However, no single technique can outperform other techniques in all cases. Therefore, we apply several classification techniques to construct fault prediction models for Web applications. In this study, we applied logistic regression and machine learning methods (NNge, random forest, neural network, Naïve Bayes), which were commonly used in other fault prediction studies such as [6], [7], [9], [19].

V. Case study

In this section, we will describe our experimental analyses. The subject of our case study is Drupal [20], a very large open source Web Content Management System written in PHP. There are several reasons that we select Drupal for evaluating our approach:

- Drupal is popular and has been developed for a long time.
- We can access the source code repository of Drupal easily. Moreover, the bug tracking system for Drupal provides enough information for validation process.

Table II
Confusion Matrix

	YES(Actual value)	NO(Actual value)
YES(Predicted value)	True Positive	False Positive
NO(Predicted value)	False Negative	True Negative

To evaluate performance of the binary classification model, researchers usually use performance indicators derived from the confusion matrix, which is shown in Table II. Faulty modules are regarded as positive (YES) and non-faulty modules are regarded as negative (NO). Based on the confusion matrix, we can evaluate the model with four common criteria: accuracy, precision, recall and F-measure. Accuracy is the percentage of correct predictions among all predictions. Precision is the percentage of modules which have been classified correctly as faulty among all modules that are predicted as faulty. Recall is the percentage of faulty

modules which have been classified correctly. Since there is a trade-off between precision and recall [21], F-measure is proposed to take into account both precision and recall indicators to make a harmonic mean. Details about these performance indicators can be found in many materials such as [22], [23].

In our experiment, we analyzed 81 modules running on Drupal 6.14, including core modules and plug-ins modules. For each module, we collected data on six proposed metrics by mining the source code repository. Since the examined software metrics were not measured with the same units, we applied min-max normalization so that their values would lie between 0 and 1 as in [24]. When evaluating performance of our fault prediction models, we used 10-fold cross-validation, which is commonly applied by many researchers due to its relatively low bias and variance [25]. Besides, to achieve reliable results, we repeated 10-fold cross-validation 10 times and calculated average results as in [6]. To build and validate fault prediction models, we applied WEKA, a very popular data mining workbench [21].

A. Performance of our proposed fault prediction models

In this section, we will present the performance of our fault prediction models by using six metrics described in Section IV as fault predictors and common classification techniques (logistic regression and machine learning methods).

1) Logistic regression model: First, before building multivariate logistic regression (MLR) model, we carried out univariate logistic regression analyses to evaluate whether each software metric we proposed is significant for fault prediction or not. In Table III, the p-value is related to statistical hypothesis which tells us the significance of a metric. The cut-off value for p-value is 0.05, which means that a software metric that has a p-value larger than 0.05 is not significant. We can see that all software metrics are significant and can be used for fault prediction. In logistic regression, R^2 is defined as the proportion of the total variation in the dependent variable that is explained by the model. The higher R^2 is, the better the dependent variable is explained by the explanatory variable. However, one important point is that the high value of R^2 is rare in logistic regression [9]. From Table III, we can see that NOH is the best predictor among all 6 metrics. This result is somehow consistent with other studies described in previous sections, which found that there are commonly a lot of faults related to links in Web applications.

Since the examined metrics are not totally independent and they may capture redundant information, we applied stepwise regression selection procedure to select the relevant metrics when constructing MLR model. This selection procedure was also used in other fault prediction studies such as [9], [19]. The performance of our fault prediction model using MLR in cross-validation is shown in Table IV. We can see that the average accuracy of our model is 82.78%, which means that in average, 82.78% of all modules could be correctly classified. Along with accuracy, other performance indicators are also good (average precision = 90.87%, average recall = 83.03%, average F-measure = 0.8561).

Table III
RESULT OF UNIVARIATE LOGISTIC REGRESSION

Metric name	p-value	R^2
LOJ	< 0.05	0.243
NOFJ	< 0.05	0.276
LOCSS	< 0.05	0.196
NOIC	< 0.05	0.194
NOH	< 0.05	0.328
LOC	< 0.05	0.299

Table IV
PERFORMANCE OF OUR FAULT PREDICTION MODELS USING PROPOSED FAULT PREDICTORS AND MULTIVARIATE LOGISTIC REGRESSION

Accuracy	Precision	Recall	F-measure
82.78%	90.87%	83.03%	0.8561

2) Machine learning models: In this section, we will present the results of fault prediction models using our proposed fault predictors and common machine learning techniques: NNge, random forest (RF), neural network (NN) and Naïve Bayes (NB). We applied Correlation-based Feature Selection technique [26] to select the best subset of metrics for fault prediction because some metrics might be highly correlated. This feature selection technique is popular in machine learning area and was also used in other fault prediction studies such as [8], [12]. The performance of our fault prediction models using four machine learning techniques in cross-validation is shown in Table V. We can see that all models achieve quite promising results. The lowest average accuracy among these models is 72.25% and other performance indicators are also good.

Table V
PERFORMANCE OF OUR FAULT PREDICTION MODELS USING PROPOSED FAULT PREDICTORS AND MACHINE LEARNING TECHNIQUES

Technique	Accuracy	Precision	Recall	F-measure
NNge	82.72%	87.41%	87.37%	0.8662
RF	79.61%	85.32%	84.03%	0.8384
NN	79.78%	89.05%	80.63%	0.8324
NB	72.25%	89.68%	66.97%	0.7487

B. Comparison with fault prediction models using only popular software metrics as fault predictors

In this section, we will compare our fault prediction models using proposed metrics with three other kinds of models using only popular metrics:

- *Model I uses only popular code metrics as fault predictors.*
- *Model II uses only popular change metrics as fault predictors.*
- *Model III uses both popular code metrics and change metrics as fault predictors.*

Table VI
LIST OF CODE METRICS USED FOR COMPARISON

Metric name	Description
LOC	The total number of lines of code of source code files in a module
NOF	The number of functions defined in a module (for server-side Web programming language)
McCabe's cyclomatic complexity	Sum of the number of available decision paths for all functions defined in a module (for server-side Web programming language)
AVCOM	Average McCabe's cyclomatic complexity per function (for server-side Web programming language)

Table VII
LIST OF CHANGE METRICS USED FOR COMPARISON

Metric name	Description
NOD	The number of distinct developers working for a module
LOC_ADDED	The total number of LOC were added to all source code files in a module over all their revisions
LOC_DELETED	The total number of LOC were deleted from all source code files in a module over all their revisions
CODE_CHURN	Sum of (added LOC − deleted LOC) for all source code files in a module over all their revisions
AGE	Age of a module (counted in weeks)

Table VIII and Table VII give the lists of popular code metrics and change metrics we used for comparison. All these metrics were commonly used to predict faulty modules for normal applications in other studies and we collected data on them by mining the source code repository. Table VIII shows the comparison results across five classification techniques. In this table, a cell with (+) means that the performance indicator related to this cell is significantly better than the corresponding performance indicator in our model using the same classification technique. A cell with (=) means that the performance indicator related to this cell is not significantly different from the corresponding performance indicator in our model using the same classification technique. A cell with (-) means that the performance indicator related to this cell is significantly worse than the corresponding performance indicator in our model using the same classification technique. Significance is computed using the Mann-Whitney U test [27] with the significance level $\alpha = 0.05$. With all common classification techniques, we can see that our fault prediction models using proposed metrics significantly achieved better results than models using only popular metrics. Therefore, fault prediction models using our proposed metrics may be applied in practice to achieve a better prediction accuracy.

Table VIII
COMPARISON RESULTS WITH FAULT PREDICTION MODELS USING ONLY POPULAR SOFTWARE METRICS AS FAULT PREDICTORS

		Our model	Model I	Model II	Model III
MLR	Accuracy	82.78%	77.58% (-)	72.51% (-)	76.19% (-)
	Precision	90.87%	83.99% (-)	82.20% (-)	85.14% (-)
	Recall	83.03%	82.70% (=)	76.23% (-)	79.00% (=)
	F-measure	0.8561	0.8223 (-)	0.7761 (-)	0.8049 (-)
NNge	Accuracy	82.72%	70.71% (-)	67.79% (-)	70.60% (-)
	Precision	87.41%	77.43% (-)	75.73% (-)	76.13% (-)
	Recall	87.37%	79.20% (-)	78.13% (-)	82.27% (-)
	F-measure	0.8662	0.7727 (-)	0.7542 (-)	0.7805 (-)
RF	Accuracy	79.61%	68.92% (-)	65.93% (-)	69.67% (-)
	Precision	85.32%	76.62% (-)	74.04% (-)	77.72% (-)
	Recall	84.03%	78.23% (-)	75.77% (-)	78.43% (-)
	F-measure	0.8384	0.7570 (-)	0.7362 (-)	0.7637 (-)
NN	Accuracy	79.78%	71.11% (-)	72.06% (-)	73.24% (-)
	Precision	89.05%	78.64% (-)	84.77% (-)	84.08% (-)
	Recall	80.63%	77.87% (=)	71.50% (-)	74.80% (-)
	F-measure	0.8324	0.7714 (-)	0.7600 (-)	0.7756 (-)
NB	Accuracy	72.25%	67.07% (-)	67.25% (-)	69.03% (-)
	Precision	89.68%	92.98% (+)	90.17% (=)	90.92% (=)
	Recall	66.97%	55.50% (-)	57.93% (-)	60.63% (-)
	F-measure	0.7487	0.6675 (-)	0.6816 (-)	0.7018 (-)

VI. CONCLUSIONS AND FUTURE WORK

Software fault prediction models can be applied to improve software quality. Furthermore, these models also help project managers allocate resources for testing activities more reasonably. As a result, we can reduce development costs and deliver our software projects within budget with minimal schedule slippage. Therefore, it is necessary to develop and apply fault prediction models during the software development process.

In this paper, we introduce fault prediction models for Web applications with consideration of their particular characteristics. The experimental results show that our fault prediction models achieve better performance than popular ones. For

future work, we will try to replicate our approach on other Web systems so that we can evaluate comprehensively our proposed fault prediction models.

ACKNOWLEDGMENT

This research was supported by the MKE(Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency) (NIPA-2010-(C1090-1031-0001))

REFERENCES

[1] Y. Wu and J. Offutt, "Modeling and testing web-based applications," George Mason University, Virginia, Tech. Rep. ISE-TR-02-08, Nov. 2002.

[2] L. S. Al-Salem and A. A. Samaha, "Eliciting web application requirements - an industrial case study," *The Journal of Systems and Software*, vol. 80, no. 3, pp. 294–313, 2007.

[3] D. M. Brandon, Ed., *Software Engineering for Modern Web Applications: Methodologies and Technologies*. NewYork: Information Science Reference, 2008.

[4] S. Casteleyn, F. Daniel, P. Dolog, and M. Matera, *Engineering Web Applications*. Heidelberg: Springer, 2009.

[5] C. Catal and B. Diri, "A systematic review of software fault prediction studies," *Expert Systems with Applications*, vol. 36, no. 4, pp. 7346–7354, 2009.

[6] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *Proc. ACM/IEEE 30th International Conference on Software Engineering (ICSE'09)*, Leipzig, Germany, May 2008, pp. 181–190.

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

[8] K. O. Elish and M. O. Elish, "Predicting defect-prone software modules using support vector machines," *The Journal of Systems and Software*, vol. 81, no. 5, pp. 649–660, 2008.

[9] T. Gyimóthy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Transactions on Software Engineering*, vol. 31, no. 10, pp. 897–910, Oct. 2005.

[10] T. L. Graves, A. F. Karr, J. Marron, and H. Siy, "Predicting fault incidence using software change history," *IEEE Transactions on Software Engineering*, vol. 26, no. 7, pp. 653–661, Jul. 2000.

[11] N. Nagappan, T. Ball, and B. Murphy, "Using historical in-process and product metrics for early estimation of software failures," in *Proc. IEEE 17th International Symposium on Software Reliability Engineering*, Raleigh, North Carolina, USA, Nov. 2006, pp. 62–74.

[12] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *The Journal of Systems and Software*, vol. 81, no. 11, pp. 2–17, 2010.

[13] F. Ricca and P. Tonella, "Detecting anomaly and failure in web applications," *IEEE Multimedia*, vol. 13, no. 2, pp. 44–51, 2006.

[14] Z. Li, N. Alaeddine, and J. Tian, "Multi-faceted quality and defect measurement for web software and source contents," *The Journal of Systems and Software*, vol. 83, no. 1, pp. 18–28, 2010.

[15] M. V. Zelkowitz, Ed., *Web Technology*, ser. Advances in Computer. Amsterdam: Elsevier Academic Press, 2006, vol. 67.

[16] N. Curtis, *Modular Web Design: Creating Reusable Components for User Experience Design and Documentation*. Berkeley, CA: Peachpi Press, 2010.

[17] W3C Standards for Web design and applications: http://www.w3.org/standards/webdesign/

[18] Y. Zhou, B. Xu, and H. Leung, "On the ability of complexity metrics to predict fault-prone classes in object-oriented systems," *The Journal of Systems and Software*, vol. 83, no. 4, pp. 660–674, 2010.

[19] Y. Zhou and H. Leung, "Empirical analysis of object-oriented design metrics for predicting high and low severity faults," *IEEE Transactions on Software Engineering*, vol. 32, no. 10, pp. 771–789, Oct. 2006.

[20] Drupal project: http://www.drupal.org/

[21] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. San Francisco, CA: Morgan Kaufmann, 2005.

[22] D. T. Larose, *Data Mining Methods and Models*. New Jersey: John Wiley and Sons, 2006.

[23] D. L. Olson and D. Delen, *Advanced Data Mining Techniques*. Heidelberg: Springer, 2008.

[24] L.-W. Chen and S.-J. Huang, "Accuracy and efficiency comparisons of single- and multi-cycled software classification models," *The Journal of Systems and Software*, vol. 51, no. 1, pp. 173–181, 2009.

[25] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, 2nd ed. San Francisco, CA: Morgan Kaufmann, 2006.

[26] M. A. Hall, "Correlation-based feature selection for discrete and numeric class machine learning," in *Proc. 17th International Conference on Machine Learning*, CA, USA, Jun./Jul. 2000, pp. 359–366.

[27] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Press, 2000.