Global Journal of Computer Science and Technology
Software & Data Engineering
Volume 13 Issue 8 Version 1.0 Year 2013
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Performance Evaluation of Non Functional Requirements

By K. Mahalakshmi & Dr. R. Prabhakar
Surya Group of Institutions, India

Abstract - Requirement engineering (RE) concerns goal identification by a system, operationalization of such goals into services and constraints, and assigning responsibilities, needs to agents including humans, devices/software. RE processes include negotiation, documentation, domain analysis, specification, elicitation, assessment, and evolution. It is difficult and critical to get high quality requirements. The paper gives a synopsis of the field of requirements engineering. RE is defined, and a brief history of main concepts and techniques is presented. The result got by using the method is very promising. It was evaluated extensively on Non Functional Requirements (NFR) dataset obtained from PROMISE repository, which is publicly accessible.

Keywords : requirement engineering, functional require-ments, non function requirements, performance.

GJCST-C Classification : H.3.4

Strictly as per the compliance and regulations of:

© 2013. K. Mahalakshmi & Dr. R. Prabhakar. This is a research/review paper, distributed under the terms of the Creative Commons Attribution-Noncommercial 3.0 Unported License http://creativecommons.org/licenses/by-nc/3.0/), permitting all non-commercial use, distribution, and reproduction inany medium, provided the original work is properly cited.

Performance Evaluation of Non Functional Requirements

K. Mahalakshmi [α] & Dr. R. Prabhakar [σ]

Abstract - Requirement engineering (RE) concerns goal identification by a system, operationalization of such goals into services and constraints, and assigning responsibilities, needs to agents including humans, devices/software. RE processes include negotiation, documentation, domain analysis, specification, elicitation, assessment, and evolution. It is difficult and critical to get high quality requirements. The paper gives a synopsis of the field of requirements engineering. RE is defined, and a brief history of main concepts and techniques is presented. The result got by using the method is very promising. It was evaluated extensively on Non Functional Requirements (NFR) dataset obtained from PROMISE repository, which is publicly accessible.

Keywords : requirement engineering, functional requirements, non function requirements, performance.

I. Introduction

Requirements engineering (RE) [1] is activities set concerning identifying/communicating a software-intensive system's purpose and contexts of use. So, RE spans users real-world needs, customers, and other constituencies affected by software systems and capabilities/opportunities provided by software-intensive technology. An abstract description of how a specific organization conducts activities, resource usage focused and dependencies between activities is a process model. Methods and process models difference is that while methods focus on technical activities (activities content), process models focus on activities management (how activities can be measured/improved).

A software system's success measure [2] is the degree to which it meets its intended purpose. Generally, software systems requirements engineering discovers that purpose through identification of stakeholders, their needs and documenting them in a process amenable to analysis, communication, and implementation. There are many difficulties in this. Stakeholders (paying customers, users and developers) could be numerous and distributed. "Requirements engineering is that branch of software engineering dealing with real-world goals for, functions of, and constraints on software systems. It concerns these factors, relationship to precise software behavior and to its evolution with time across software families."

Author α : Associate professor, Dept. of CSE, Surya Group of Institutions, Tamil Nadu, India.
E-mail : mailtok_mahalakshmi@rediffmail.com
Author σ : Emeritus Professor, Dept. of CSE, Coimbatore Institute of Technology, Tamil Nadu, India.

A requirement is a condition/capability to be met/fulfilled by a system satisfying a contract, specification, standard, or formally imposed documents. Requirements for a system should be verifiable, consistent, correct, and traceable. RE specifies, understands, elicits, and validates customers/users requirements. It identifies technological restrictions through which an application should be built/run. An iterative/co-operative process, it aims to analyze a problem, document results in various formats, evaluating results precision.

RE iterative process includes 3 activities [3]:

- Requirements elicitation
- Requirements specification
- Requirements validation

The process starts with requirements elicitation. A developers' set collect users and customers information. Information is got from documents, legacy applications, interviews used in preparation of requirements catalogue. Finally, requirements validation finds out if there are inconsistencies/mistakes/undefined requirements. Specification-validation is iterative being executed many times in complex projects.

Activities which are basic to all RE processes [4]:

- *Elicitation :* Identify information sources about system and discover requirements from them.
- *Analysis :* Understand requirements, their overlaps, and conflicts.
- *Validation :* Reverting to system stake holders to see if requirements are what they need.
- *Negotiation :* Inevitably stakeholders' views will differ from proposed requirements creating conflicts. Try to reconcile such views generating consistent requirements set.
- *Documentation :* Write requirements in a way that stakeholders/software developers understand.
- *Management :* Control requirements changes that will arise.

© 2013 Global Journals Inc. (US)

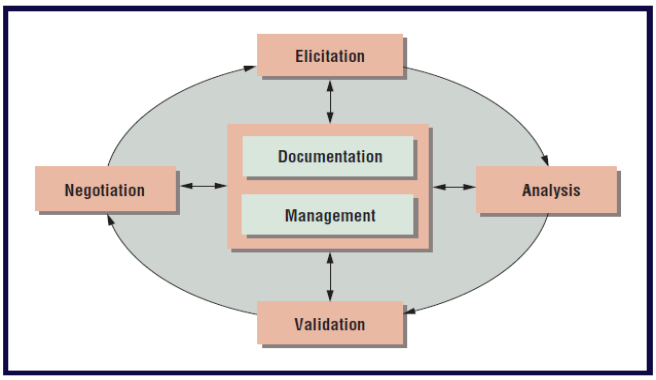

Figure 1 : The requirements engineering activity cycle

Requirements are software system's foundation. Functional requirements indicate what a system can do, data requirements indicate what it can store while quality requirements indicate how quickly /easily it performs.

a) Functional Requirements

Functional requirements [5] capture a system's intended behavior which could be expressed as services, tasks or functions the system has to perform. It is useful to distinguish between baseline functionality required for a system to compete in that product domain, in product development. F*eatures* differentiate a system from competitors' products, and from the company's own product line/family variants. Features may be added functionalities, or differ from basic functionality along some quality attribute (performance or memory utilization). Functional requirements of early (nearly concurrent) releases need to be considered. Later releases can be accommodated through architectural qualities like extensibility and flexibility.

b) Non Functional Requirements

A semantic definition would be "*any requirement that is not functional*" [6]. Non-functional requirements are those which cannot be categorized in Functional, Data or Process requirements. Generally,

- They are requirements
- They are not functional, data or process requirements

Non-functional requirements define overall qualities/attributes of the system that results. Non-functional requirements restrict product under development, development process, specifying external constraints to be met by that product.

Some of the non-functional requirements are,

⇒ Availability Requirements
⇒ Capacity Requirements
⇒ Performance Requirements
⇒ Reliability Requirements
⇒ Security Requirements

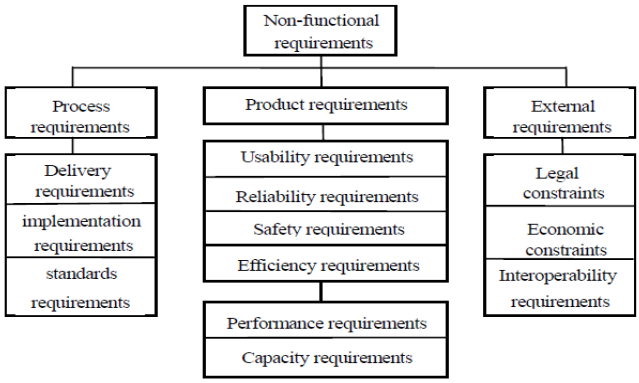

Figure 2 : Classification of Non-functional requirements

To measure ad hoc information retrieval effectiveness requires a test collection of three things:

1. A document collection.
2. Information needs test suite, expressible as queries.
3. A relevance judgments set, usually a binary assessment of either *relevant* or *non-relevant* for a query-document pair.

Usual approaches to information retrieval system evaluation include *relevant* and *non-relevant* documents notion. Regarding user information need, a test collection document is provided a binary classification either as relevant/non relevant. This decision is called the *gold standard* or *ground truth relevance* judgment.

NFR Locator extracts NFR sentences in unconstrained natural language documentation. The process takes project related natural language document as input. The former parses natural language into an internal representation based upon relevant features, to classify sentences into particular NFR categories or returns "not applicable" when it does not specify a NFR [7].

Step 1 : Parse Natural Language

The process enters text into a system, parsing it and converting parsed representation into NFR Locator's sentence representation (SR). SR represents every sentence as directed graph where vertices are words and edges the relationships between them.

Step 2 : Classify Sentences

Once parsing and initial sentence analysis is finished, a k-NN classification algorithm classifies every sentence into one/more NFR categories. Sentences classified other than "not applicable" appear on generated reports for use outside the system. A k-NN classifier predicts classification by taking a majority vote of existing k nearest neighbors' classification to the item under test.

II. Related Works

Non-functional requirements identification is important for development/deployment of software

© 2013 Global Journals Inc. (US)

products. Customers software product acceptance depends on non-functional requirements incorporated in the software. It should identify all non-functional requirements of stakeholders. Many approaches are unavailable for this. Rao and Gopich and [8] suggested a 4 layered analysis approach to identify non-functional requirements. The approach has advantages over non-layered approach. Rules were proposed for use in each layer as part of the approach which was successfully applied on 2 case studies. The identified non-functional requirements were validated through the use of a check list. Also, a metric ensured computation of completeness of the identified non-requirements.

Functionality and non-functional characteristics determine a software system's utility. Also usability, flexibility, performance, interoperability and security add to the score. There is currently a lop-sided emphasis on software functionality, though it was not useful or usable without non-functional characteristics. Chung and do Prado Leite [9] reviewed state of the art on treating non-functional requirements (NFRs), when providing prospects for future directions.

Liu et al [10] proved that continuous randomization spectrum existed where most existing tree randomizations operated around the spectrum's two ends leaving a major portion of the spectrum unexplored. The authors proposed A base learner VR-Tree generating trees with variable-randomness. VR-Trees spanned from conventional deterministic trees to complete-random trees by using a probabilistic parameter. Using VR-Trees as base models, the spectrum of randomized ensembles was explored along with Bagging and Random Subspace. It discovered that spectrum's two halves have distinct characteristics; understanding which led to the proposal of a new approach to build better decision tree ensembles. It was named Coalescence, as it coalesces many points in spectrum's random-half. Coalescence behaves like an experts committee to cater to unforeseeable conditions in training data. Coalescence performed better than the spectrum's any single operating point, without needing to tune in to a specific randomness level. The proposed empirical study ranks Coalescence top among benchmarking ensemble methods including Random Forests, Random Subspace and C5 Boosting. Coalescence was significantly better than Bagging and Max-Diverse Ensemble when compared with other methods. Though Coalescence was not greater than Random Forests, it identified conditions under which one can perform better than the other.

Pavlovski and Zou [11] proposed application of 2 new artifacts to model linked with a business process. This was operating condition denoting a business process constraint. Control case defined controlling criteria to mitigate the risk associated with an operational condition. Modeling constraints thus was an opportunity to capture such business process characteristics early in a systems development cycle. This contributes to a model providing a more through overall business process representation. The methods assist in risk mitigation and facilitate non-functional requirements early recovery during systems development.

Though all systems have non-functional requirements (NFRs), they are not clearly stated in formal specification requirements. Further, NFRs may be externally imposed through government regulations/industry standards. Slankas and Williams [12] examined document types (data use agreements, installation manuals, regulations, proposals requests, requirements specifications, and user manuals) containing NFRs categorized in 14 NFR categories (capacity, reliability, and security) measuring how to effectively identify/classify NFR statements in those documents. In documents evaluated, NFRs were present. Using a NFR word vector representation, a support vector machine algorithm performed twice as effectively compared to the same input on a multinomial Naïve Bayes classifier. The k nearest neighbor classifier with a unique distance metric had an F1 measure of 0.54, outperforming in experiments, optimal Naïve Bayes classifier which had a F1 measure of 0.32. It was also found that stop word lists beyond common determiners lacked minimal performance effect.

Asghar and Umar [13] discussed/critically evaluated RE challenges highlighted by researchers and provided a model encapsulating 7 major challenges recurring in a RE phase. The challenges were further categorized as problems. Further, the model was linked to earlier research elaborating challenges not specified earlier. Anticipating RE challenges could help RE engineers prevent software tower from destruction.

RE is an effective phase in software development aiming to collect good requirements from stakeholders correctly. It is important for an organization to develop quality software products satisfying user needs. RE for software development is a complex exercise taking into account product demands from many viewpoints, roles, responsibilities, and objectives. Hence, it is necessary to apply RE practices in all software development phases. Pandey et al [14] proposed an effective RE process model to produce quality software development requirements. Requirement management/planning were executed independently for effective requirements management. It was iterative for better RE and maintenance later. Successful implementation of the proposed RE process has good impact on quality software production.

III. Methodology

For classifier validation, NFR dataset available in the promise data repository [15] was used. It consists of 15 requirement specifications of MS student projects with a total of 326 NFRs and 358 FRs. NFR categories

© 2013 Global Journals Inc. (US)

included availability, scalability, usability and security. Features extraction was from each requirement document using word occurrence criteria. Extracted data was used to investigate bagging and boosting methods.

a) Boosting

Boosting [16] is a method to improve learning algorithms accuracy. Given a training set of labeled examples, {(*x*1*; y* 1)*;*(*x*2*; y*2),…,(*xm; ym*)}, where each *xi* is drawn from an underlying distribution *D* on a universe *X*, and *yi* ∈ {+1,-1}, a learning algorithm produces a hypothesis *h* : *X* → {+1,-1}. Ideally, *h* "describes" not just given samples, but also underlying distribution. Boosting converts a weak learner, producing a hypothesis that is slightly better than random guessing, into a strong/accurate learner. Many boosting algorithms share a basic structure. First, the sample set is given an initial (typically uniform) probability distribution. Computation proceeds in rounds. In each round *t:* (1) base learner is run on current distribution *Dt*, producing a classification hypothesis *ht;* and (2) the hypotheses *h*1,…, *ht* reweight samples, defining *Dt*+1. The process halts after predetermined rounds or when combining of hypotheses is accurate. Main design decisions on how to modify probability distribution from one round to next, and how to combine hypotheses $\{ht\}_{t=1,\ldots,T}$ to form a final output hypothesis.

Bagging [17] is based on bootstrapping and aggregating. Bootstrapping is based on random sampling with replacement. Hence, taking a bootstrap replicate $S^i = (X_1^i, X_2^i, \ldots, X_n^i)$ of the training set $S = (X_1, X_2, \ldots, X_n)$, sometimes has less misleading training instances in bootstrap training set. Thus, a classifier constructed on such training sets provides better performance. Aggregating means combining classifiers, Bagging provides good results when unstable learning algorithms (decision trees) are used as base-level classifiers, with small changes in training sets resulting in different classifiers.

b) The bagging algorithm

Input : Training examples *S*, Bag size *B*
Output : Ensemble *E*
E ← 0
for *i* = 1 to *B* do
S^i ← *BootstrapSample* (*S*)
C^i ← *ConstructClassifier* (S^i)
E ← *E* ∪ {C^i}
end for
return *E*

c) Random Forest

Random forests [18] are a recursive partitioning method suiting small *n* large *p* problems. They involve a classification ensemble (aka: set) or regression trees calculated on random data subsets, using a randomly restricted and selected predictor's subset for splits in each classification tree.

The posterior probability that a random tree predicts class *j* at *X*, given the training data (*xi, yi*), *i* = 1,…, *n*, is

$$Q_j(X) = P_\theta\left(h(X,\theta) = j\right)$$

Note that *h* depends on training data. In practice, *Qj is estimated* using

$$\hat{Q}_j(X) = \frac{1}{N}\sum_{k=1}^{N} I\left(h(X,\theta_k) = j\right)$$

where *I* denotes indicator function. The ensemble predicts class at *X* by

$$\hat{h}(X) = \arg\max_j \hat{Q}_j(X)$$

d) REP TREE

Reptree uses regression tree logic to create multiple trees in varied iterations. It then selects the best from generated trees which is then considered as representative. In tree pruning the measure used is mean square error on the tree's predictions.

IV. Experimental Results

The classification accuracy and the Root Mean Squared Error (RMSE) are shown in Table 1.

Table 1 : Classification and RMSE of the technique under consideration

Classifiers	Classification accuracy %	Root mean squared error
Bagging with Reptree	59.29	0.23
Bagging with Random Forest	62.82	0.2174
Bagging with Reptree and resampling	70.83	0.197
Bagging with Random Forest and resampling	82.37	0.1562
Logitboost with Reptree	59.94	0.228
Logitboost with decision stump	60.42	0.2232
Logitboost with Reptree and resampling	71.47	0.1972
Logitboost with decision stump and resampling	78.37	0.1739

In table 1, the performance variations of classifiers have been shown. The Classification Accuracy and RMSE results of the classifiers are shown in Figure 3 & 4.

© 2013 Global Journals Inc. (US)

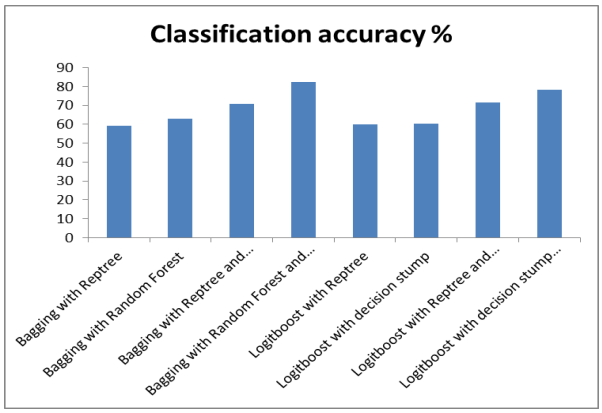

Figure 3 : Classification Accuracy

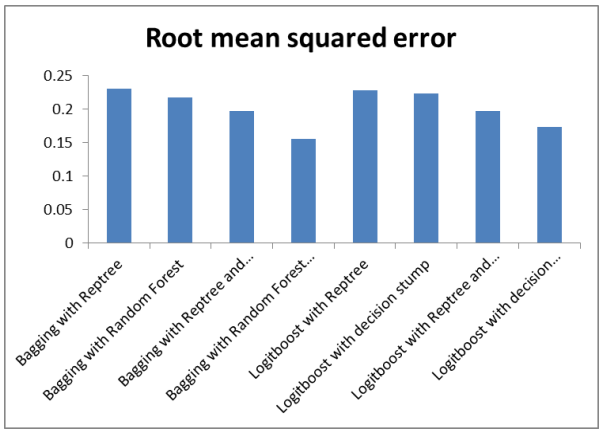

Figure 4 : Route Mean Squared Error

V. Experimental Results

RE activities occur across multiple phases. Of the 7 suggested activities, only elicitation is performed clearly in all projects. Interpreting & Structuring, and Negotiation were also performed in the projects, but they varied between implicit and explicit performance. When RE was considered as a continual task through the project, RE process model was iterative. RE activities occurred across multiple phases, making process models appear iterative. Boosting and Bagging classifiers were used in experiments with Reptree, Random forest and resampling. The results showed the performance variation between classifiers. Bagging with Random Forest and resampling achieves the best performance accuracy of 82.37%.

References Références Referencias

1. Easterbrook, S. (2004). What is Requirements Engineering?
2. Nuseibeh, B., & Easterbrook, S. (2000, May). Requirements engineering: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering* (pp. 35-46). ACM.
3. Escalona, M. J., & Koch, N. (2004). Requirements engineering for web applications-a comparative study. *J. Web Eng., 2* (3), 193-212.
4. Sommerville, I. (2005). Integrated requirements engineering: A tutorial. *Software, IEEE*, *22(1),* 16-23.
5. Malan, R., Bredemeyer, D., & Consulting, B. (1999). Functional requirements and use cases. *functreq. pdf, 39k) June*.
6. Beauchamp, G. (2009). 'Business Analysis - Delivering the Right Solution for the Right Problem. *Smart BA (February 20, 2007). www. smart-ba. com/articles/ba_chain_of_reasoning. pdf*.
7. John, S. and Laurie, W. (2013). Automated Extraction of Non-functional Requirements in Available Documentation.
8. Rao, A. A., & Gopichand, M. (2012). Four Layered Approach to Non-Functional Requirements Analysis. *arXiv preprint arXiv:1201.6141*.
9. Chung, L., & do Prado Leite, J. C. S. (2009). On non-functional requirements in software engineering. In *Conceptual modeling: Foundations and applications* (pp.363-379). Springer Berlin Heidelberg.
10. F T Liu, K M Ting, Y Yu, Z H Zhou. Spectrum of Variable-Random Trees. Journal of Artificial Intelligence Research 32 (2008) 355-384.
11. Pavlovski, C. J., & Zou, J. (2008, January). Non-functional requirements in business process modeling. In *Proceedings of the fifth Asia-Pacific conference on Conceptual Modelling-Volume 79* (pp. 103-112). Australian Computer Society, Inc..
12. John, S. & Laurie, W. (2013). Automated Extraction of Non-functional Requirements in Available Documentation, IEEE.
13. Asghar, S., & Umar, M. (2010). Requirement engineering challenges in development of software applications and selection of customer-off-the-shelf (COTS) components. *International Journal of Software Engineering*, *1(1),* 32-50.
14. Pandey, D., Suman, U., & Ramani, A. K. (2010, October). An effective requirement engineering process model for software development and requirements management. In *Advances in Recent Technologies in Communication and Computing (ARTCom), 2010 International Conference on* (pp. 287-291). IEEE.
15. Selvakumar, J., & Rajaram, M. (2011). Performance Evaluation of Requirements Engineering Methodology for Automated Detection of Non Functional Requirements. *International Journal, 3*.
16. Dwork, C., Rothblum, G. N., & Vadhan, S. (2010, October). Boosting and differential privacy. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on* (pp. 51-60). IEEE.
17. Pance, P. & Saso, D. (2007). Combining Bagging and Random Subspaces to Create Better Ensembles. Springer-Verlag Berlin Heidelberg.
18. S.K. Jayanthi and S. Sasikala. (2013). Reptree Classifier For Identifying Link Spam In Web Search Engines. Ictact Journal On Soft Computing, 3(2).

© 2013 Global Journals Inc. (US)

This page is intentionally left blank

© 2013 Global Journals Inc. (US)