Release Readiness Measurement: A Comparison of Best Practices

Nico Koprowski
RWTH Aachen University
Ahornstr. 55
52074 Aachen, Germany
Email: nico.koprowski@rwth-aachen.de

M. Firdaus Harun
RWTH Aachen University
Research Group Software Construction
Ahornstr. 55
52074 Aachen, Germany
Email: fharun@swc.rwth-aachen.de

Horst Lichter
RWTH Aachen University
Research Group Software Construction
Ahornstr. 55
52074 Aachen, Germany
Email: lichter@swc.rwth-aachen.de

***Abstract*—A crucial part of software project management consists of deciding when the software in development is ready to be released. We call this property of a software system release readiness. This work clarifies the meaning of release readiness by giving a presentation of three selected approaches which measure this property. This work distinguishes these approaches according to the criteria which they use to determine the release readiness: (1) Defect tracking uses the number and distribution of discovered defects inside the software for estimating release readiness. (2) The Software Readiness Index (SRI) makes use of quality and reliability metrics in order to determine release readiness. (3) ShipIt describes an elaborate measurement for release readiness which uses the development progress of the software product as an underlying metric A comparison between these three approaches shows that defect tracking is simple but limited in its application while ShipIt comes to short in measuring the important criteria of quality and reliability. Finally, SRI renders to be the most versatile and detailed approach for release readiness measurement.**

***Keywords*—release readiness measures, quality measures, reliability measures, release engineering.**

I. Introduction

A very important decision in software development projects is when to release the product. Generally, the project manager has to make a tradeoff between software quality and keeping the schedule. Deciding arbitrarily on instincts may lead to a poor quality release or unnecessary additional costs.

Besides that, the delivered product also has an influence to market share (i.e., the reputation of the software). As investigated by [1] on selected Web browsers (Chrome, Firefox and Internet Explorer), they found that by implementing a proper release engineering, there are significant benefits on software reputation.

Several techniques and approaches have been developed to support the decision for the perfect release time. Project estimation models give a rough prediction of the overall effort needed to finish a software project. Therefore, managers can anticipate the approximate project duration and with this, the release time of the product. However, predictions are just accurate in so far and may get useless if unexpected events occur. In the later phases of the development process descriptive measurements are needed to make a decision about the right release time.

In addition, current research on when the right release time (i.e., when-to-release) in an iterative development has been studied by [2]. They developed a plugin tool (called *W2RP*) which estimates the tradeoff relationship between number of features implemented and quantified quality prediction from related effort investment. The tool supports managers to analyze the impact of release dates on the product development. However, the quality quantification formulated in this tool lacks of measurements on code quality and technical debt. To deliver such high quality software, both measurement parameters important to consider without investing too much effort (i.e., time and cost consuming for refactoring) in the future.

The release readiness is a property of a software product indicating whether the product is (or in how far it is) ready to be released. For a start, like the properties maintainability, efficiency or reliability the release readiness is a qualitative property and needs to be quantified to have a useful meaning. While traditional software properties have established standards, the way of correctly measuring release readiness is still a matter of research.

This work presents and compares three approaches to measure the release readiness of a software. Section II discusses our main goal of this research, a brief of research methodology and explanation of three criteria selected. Section III introduces several techniques which use the estimated number of remaining defects in code as a criterion for the software readiness. Section IV presents an approach which uses a selected set of software quality and reliability metrics for measuring release readiness. Next, section V deals with a third approach which associates the release readiness of the software with the overall progress of its development. In section VI this work compares the presented approaches to selected criteria. Finally section VII gives a short evaluation of the three approaches based on the comparison.

II. Our Approach

The goal of our research is to compare release readiness measurements which published in scientific papers and have been used in industrial setting. Then, we describe these measurements and discuss the components, metrics or indexes that required for when-to-release software in Section III, IV and V.

To compare these release readiness measurements, we start

Software Engineering Conference (MySEC), 2014 8th Malaysian , vol., no., pp.166,171, 23-24 Sept. 2014
doi: 10.1109/MySec.2014.6986008

searching with the main term 'release readiness' and check the relevant papers based on the research goal. Then, we classify these papers based on three criteria defined. It results a few papers related with release readiness measurement that applied in industry.

Due to few papers related with the research goal and no relevant existing comparison published paper, the three criteria have been selected based on bottom-up approach to build our own classification. As a result, the classification can be distinct into three categories and they are:

1) Defect Tracking: A measurement of how many defects (will be) remained in the software before release.
2) Quality Measurement: A measurement of software and hardware readiness based on specific quality attributes before release.
3) Progress Measurement: A measurement of activities or components progress in software development process.

III. Defect Tracking

One of the key purposes of the verification phase in software development is to increase the reliability of the product. Software reliability is the probability of executing a software without failure for a specified time period [3]. Hence, the less defects hide inside the software the higher is its reliability. Typically, the verification phase ends and is succeeded by the release process as soon as the manager assumes a satisfiable software reliability. Therefore, we are able to measure release readiness by measuring the reliability. The time when to release the software equals the time it takes to acquire a satisfiable reliability. As a result, the estimation of remaining defects inside the product is an important criterion for measuring release readiness.

McConnell [4] presents three defect tracking techniques which are suitable for measuring the release readiness of a software product:

1) Defect Density: The defect density is the average number of defects per line of written code. By taking into account the defect density of former software products the development team is able to estimate the number of defects in their current product.
2) Defect Pooling: In this technique two test teams gather found defects in separate pools A and B. Each team operates independently from one another and tests the whole software. From this on, we can further group the defects in the pools into unique and common defects, i.e. defects which are only present in one pool or defects which are present in both. The approximate number of total defects inside the software is given by $defects_{Total} = defects_A * defects_B / defects_{A+B}$. Finally, we subtract the number of unique defects (i.e. defects which are only present in one pool and not in the other) to acquire the estimated number of remaining defects inside the software.
3) Defect Seeding: During this approach one team intentionally creates defects inside the product. Another team performs the actual testing. The idea is, that the ratio between seeded defects which have been found and the well known total number of seeded defects implies this ratio between the actual defects.

Quah et al. [5], [6] present another alternative for predicting defects and, in turn, the release readiness of a software. Here, the authors consider the architecture of the product to be a good index of the number of its defects. They present an architectural defect tracking approach. This method assumes that the software architecture can be divided into 3 functional tiers:

1) Data Access Tier: This software tier provides interfaces for accessing and modifying the application. A SQL library is an example of a data access tier.
2) Business Logic Tier: Here are functions located which make up the actual behaviour of the software. The business logic tier acquires the data from lower tiers and manipulates it according to the defined logic of the software product. For example, an algorithm calculating the shortest path between a starting point and a destination can make up a business logic tier.
3) Presentation Tier: Handles the visualization and presentation of the data which the business logic tier manipulates. Additionally, the presentation tier acts as an interface through which the end user can interact with the system. This tier is what generally is perceived to be the view component (as in the model-view-controller model of software development) of a software. An example for this tier is a GUI of a software.

The approach of architectural defect tracking suggests that certain architectural properties of a software tier directly correlate to the number of defects in that very tier. For this reason, the approach introduces metrics for each tier from which one is able to deduce the number of defects. Table I summarizes these metrics according to the corresponding tier. These metrics are processed in a two-layered neural network approach. For each software component the first layer receives the presented software metrics as input values. For data access components the network receives the data access metrics, for a business logic component the business logic metrics and so forth.

From these metrics the layer determines whether the component is ready to be released or whether it contains defects that have to be removed. If the component actually contains defects the second layer calculates the number of lines of code that have to be changed and the time it takes to change them. The amount of change necessary indicates the release readiness of each component and, in total, indicates the release readiness of the whole software. In order to acquire the desired behaviour of the neural network, training is necessary. In training the network has access to the actual data on the number defects, the changed lines of code and the change time for each component. By comparing the calculated output to the actual data the network is able to learn and adapt itself on its own. With long enough training the neural network is able to effectively predict flawed components and the amount of change necessary to fix them.

Normally, a software development project manager does not aim to remove *all* defects inside the product. This is

Software Engineering Conference (MySEC), 2014 8th Malaysian , vol., no., pp.166,171, 23-24 Sept. 2014
doi: 10.1109/MySec.2014.6986008

TABLE I. THE METRICS USED FOR DEFECT TRACKING NEURAL NETWORK GROUPED BY APPLICATION TIER ADOPTED FROM [5], [6].

Tier	#	Metric
Data Access	1	Total number of select-SQL commands
	2	Total number of insert/update operations
	3	Total number of delete operations
	4	Average number of search conditions in where-clauses found in all data manipulation statements
	5	Total number of subqueries in data retrieval statements
	6	Total number of group by clauses in data retrieval statements
Business	1	Lack of cohesion in methods
	2	Number of parents
	3	Number of children
	4	Depth of inheritance tree
	5	Coupling between objects
	6	Response for a class
	7	Number of methods added
	8	Number of attributes
	9	Weighted methods per class
	10	Average method complexity
Presentation	1	Total number of user interface objects
	2	Total number of messages between user interface objects

because a lower defect density leads to a higher search time for finding defects. In turn, this means that the cost for detecting the next defect increases steadily (assuming no new bugs are introduced meanwhile). Therefore, the decision whether to release the software may not only depend on the number of remaining defects but also on the costs for removing them. As a result, the software development manager has to decide on a threshold which he sees to be acceptable.

IV. QUALITY MEASUREMENT

An alternative approach for measuring release readiness is the consideration of software quality criteria. According to established standards [7] criteria for software quality include among others:

1) Reliability
2) Functionality
3) Efficiency
4) Usability

In [8], [9], Olivieri et al. argue that commonly used standards do not specify how these criteria can be quantified and thus, developer cannot comparably measure them. Also, developers can only measure some of these specified criteria post-development and hence, are useless for release readiness estimation. As a result the authors introduce the Software Readiness Index (SRI). SRI is a collection of metrics based on established quality criteria to measure the release readiness of a software product. These metrics spread on the five vectors Software Functionality, Operational Quality, Known Remaining Defects, Testing Scope & Stability and Reliability. The SRI covers the whole development life cycle with a heavy focus on defect related criteria.

SRI comes in a basic and an enhanced version which depends on the used metrics for each vector. Fig.1 illustrates an overview of the full version. Here, 22 metrics determine the five vectors of the SRI. The basic version reduces the number of metrics to the six most important ones. Table II shows the six chosen metrics of the basic version categorized according to their respective vectors.

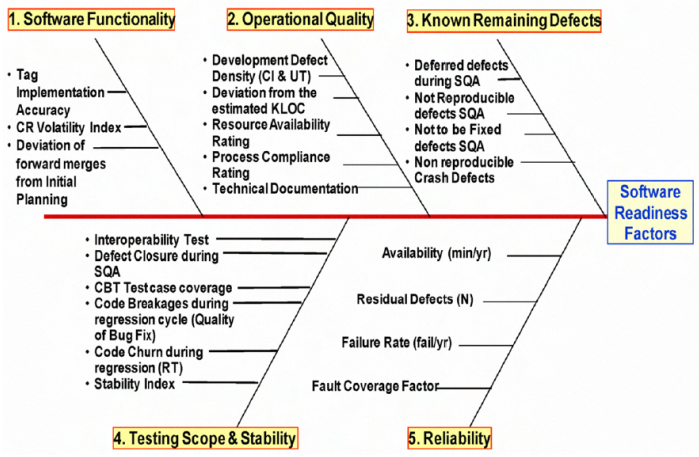

Fig. 1. Fish bone diagram of SRI taken from [9].

TABLE II. THE BASIC SRI VERSION ADOPTED FROM [8], [9].

Vector	Metric
1. Software Functionality	Tag implementation accuracy
2. Operational Quality	Development defect density
3. Known Remaining Defects	Deferred defects during software quality assurance Non-reproducible crash defects
4. Testing Scope and Stability	Total number of subqueries in data retrieval statements
5. Reliability	Residual defects

For reasons of calibration the manager can assign different weights to each criterion as well as to each vector. The overall release readiness is a value between 0 and 1 where 1 corresponds to a product which has maximum quality. To determine the release readiness the authors offer three different calculation models:

1) The additive model calculates the sum of each weighted(W) criterion/vector(V):
$SRI = \sum_{i=0}^{5}(W_i * V_i)$
2) The multiplicative model calculates the product of each weighted criterion/vector:
$SRI = \prod_{i=5}^{7}(W_i * V_i)$
3) In the hybrid model the manager can specify additive groups of weighted criteria/vectors which are then multiplied. For example:
$SRI = (\sum_{i=0}^{2}(W_i * V_i)) * (\sum_{i=3}^{5}(W_i * V_i))$
The authors prefer this model since it showed to be most suitable during their evaluation.

Choosing one calculation model, the manager is able to first calculate the readiness value of each vector separately from its criteria. In a second step he calculates the total software release readiness from the vector readiness values.

Like reliability, perfect quality scores are generally unreachable. Using SRI the manager has to aim for a certain threshold for the release readiness value. For this, the authors introduce the thresholds Green, Yellow and Red for each criterion, vector and the final release readiness. A value above the Green threshold means that the corresponding criterion fulfills the requirements and a value under the Red threshold is not acceptable. The manager determines thresholds by evaluating the data of previous projects, industry standards

Software Engineering Conference (MySEC), 2014 8th Malaysian , vol., no., pp.166,171, 23-24 Sept. 2014
doi: 10.1109/MySec.2014.6986008

and user expectations. If the overall release readiness value reaches the Green section and no sub-criterion is below the Red threshold the software product can be considered to be ready for release.

V. Progress Measurement

Satapathy [10] introduces a release readiness metric called $ShipIt$ based on the overall development progress of the software. The author is guided by the development life cycle of the waterfall model. From there on this approach identifies 7 essential components of the software development process and introduces criteria on how to measure their overall progress. Table III gives an overview of this. ShipIt uses a layered approach in which progress measurements on a lower layer are accumulated to obtain the overall progress of the higher layer. In this way, the approach introduces criteria to measure the progress of subcomponents which, in turn, are used to measure the respective component. From the obtained progress of all components the authors then measure the progress, and therefore the release readiness, of the whole product.

This approach is based on assumption that the development team finishes the gathering as well as analyzing of requirements and software designing first before they start the coding and testing of the software. This excludes models like agile development, for example. Further on, the authors consider the software ready to be released after it has been deployed at the customer. Therefore, they also consider the components supervision and support to be part of a release readiness estimation.

Like in SRI the release readiness is a value between 0 and 1 where 1 corresponds to a software product which cannot be enhanced any more for the release. Furthermore, the manager can calibrate the measurements by weighting each component. These weights represent experience from previous projects and personalize the importance of each component from the viewpoint of the manager. ShipIt measures each component according to quantifiable criteria. From the values gained for each component the model calculates the overall release readiness value as the sum of its weighted components: $ShipIt = (\sum_{i=0}^{7}(W_i * C_i))/100$

The following explanations present the criteria for each of the 7 components. The component requirement analysis and design essentially consists of the three software development processes: Gathering of requirements, analysis of requirements and software design. ShipIt provides metrics for each of these processes which are, in turn, used for calculating the value of the whole component. These metrics measure the progress of these development processes in the following way:

1) Requirements Gathering: The ratio between the requirements demanded by the customer and the already developed requirements.
2) Requirements Analysis: The ratio between already analysed requirements and those which still need to be analysed.
3) Software Design: The ratio between the requirements for which the design is done and those yet remain to be done.

Like for the overall readiness value the sum of the weighted measured processes is calculated as a value for C_1. Likewise, the value for the coding component C_2 is calculated. It consists of the three processes module creation, object creation and building:

1) Module Creation: The completed system, application and GUI modules in relation to the respective modules still to be done.
2) Object Creation: The ratio between the overall time needed to implement all objects and the already spent time on that. For the time estimation the authors propose either to use a combination of COCOMO [11] with Albrecht's Functional Points [12].
3) Building Process: The completed percentage of the times spent for compiling, handling warnings and the incremental build time as per platform dependency as well as per compiler dependency.

Component C_3, the testing, consists of the processes bug detection and debugging. $ShipIt$ measures the progress of bug detection by evaluating the completed percentage of covered features in unit testing, the covered interactions between features in integration testing and the executed overall integration tests which are required. The progress of the debugging process corresponds to the closed percentage of total issues.

Before measuring the quality assurance process $ShipIt$ requires a value for the desired software quality to be reached. Generally, this value is decided by the manager who must choose between quality and economic reasons. Having this value it is possible to estimate the required time necessary to attain the desired quality by applying appropriate models. As an example the authors propose the zero failure test hour method [13] which measures the reliability of a program by the occurrence of observed software bugs. From there on, the progress of software quality assurance can be measured by taking the spent percentage of total test hours the applied software quality model propose.

The documentation process is divided into creating requirement, design, implementation, test plan and user guide documents. For each of these documents we consider the percentage of created documents. The total number of respective documents needed can be directly calculated after the corresponding process has been completed.

TABLE III. Overview of the ShipIt components for progress measurement adopted from [10].

Index	Component	Subcomponent
C_1	Requirements	Requirements gathered Requirements analysed Software design
C_2	Coding	Modules created Objects created Build times
C_3	Testing	Test coverage Debugging
C_4	Quality	Test hours
C_5	Documentation	Requirements Design Implementation Test Plan User Guide
C_6	Supervision	Installation Training
C_7	Support	Beta test bugs

Software Engineering Conference (MySEC), 2014 8th Malaysian , vol., no., pp.166,171, 23-24 Sept. 2014
doi: 10.1109/MySec.2014.6986008

The processes installation and training make up the component supervision. For the installation progress $ShipIt$ considers the overall distribution of the software, the installation of the software and the executed acceptance tests. The training process progress is measured by evaluating the fraction of completion of developing training materials, validation of the training program and the implementation of the training program.

Finally, the last component, support, consists of the beta test process. Here, the development team receives and handles bug reports sent by beta customers. The crucial metric is the maintainability index for this component. Like for the quality the manager needs to agree with the customer on a required value. The progress of the support is then measured by taking into account the ratio between the actual and the required maintainability index.

VI. Comparison

This work presented three different approaches to estimate the release readiness of a software product. These approaches based on different techniques which are defect tracking, software quality measurement and progress measurement. In the following sections, we compare these approaches according to the scope of used criteria, the complexity of the software readiness estimation, the time when the estimation can be executed and finally possible restrictions of these approaches. Table IV shows a summary of these comparisons.

A. Scope

There is a great overlap between the scopes of the three approaches. Defect tracking concentrates on estimating the remaining defects in the software. By this, argue that the approach strongly focuses on a criterion for estimating reliability. Hence, defect tracking connects the release readiness of a software product with its reliability. The software is ready to be released when the probability of failure occurrence is satisfyingly low. The SRI, on the other hand, contains a vector especially dedicated to reliability. In this way the scope of the SRI includes the scope of the defect tracking approaches. Explicitly, the authors of SRI emphasis on quality and reliability criteria for the release readiness estimation. In fact, quality enhances the scope of reliability by additional criteria for maintainability, usability, etc. According to SRI (and all quality measurement based approaches) the software is ready for release when the software is able to meet the specified requirements. ShipIt measures the whole progress of the software development in regard to its release readiness. Therefore its scope includes quality aspects. As we mentioned before, the quality aspects include the aspect of reliability. In this regard, ShipIt has the broadest range of criteria of all three approaches: Progress measurement covers quality measurement which in turn covers defect tracking in scope.

Having a look at the actual criteria used for release readiness estimation this ratio does not necessarily hold up. SRI does not measure reliability by estimating remaining defects but uses other criteria for calculation, like the availability in minutes per year and the failure rate in failures per year. However, it is possible to use defect tracking in order to estimate these criteria if the actual data is not present.

ShipIt uses established techniques to estimate the time which is necessary to spend in order to acquire a certain software quality. Therefore, the approach differs significantly from SRI: While ShipIt estimates a time frame for when the metric expects the quality to be met, SRI only measures the current quality of the product.

B. Complexity

This section compares the number of required criteria to estimate the release readiness and the effort to acquire these criteria between the three approaches.

ShipIt is by far the most complex of the presented approaches: 30 end criteria are used to calculate the final release readiness value. Furthermore, several intermediate metrics have to be calculated and weighted for this approach. In addition to that ShipIt relies on estimation models like COCOMO for measuring the coding effort and the zero fail test hour method for measuring the software reliability.

In its full version SRI specifies 22 criteria with an overall much simpler calculation model than ShipIt. Only five intermediate metrics must be calculated for the final release readiness value. The criteria are based on direct project data at hand (Deviation from the actual KLOC) or standards for quality assurance (Stability Index). The basic version of SRI further simplifies the measurement process: The manager only has to consider 6 criteria. Whereby the method of calculation does not change for the basic version.

The complexity of defect tracking differs according to the used technique for defect estimation. Defect density, pooling and seeding are by far the simplest approaches for measuring release readiness. The necessary criteria are minimal and the effort for acquiring them is relatively low. However, when employing defect pooling the efficiency of the test team is basically halved since both teams have to cover the whole software separately. Likewise, there is an additional effort for seeding defects which is not to neglect. Defect tracking by architectural analysis, on the other hand, considers only 18 criteria which are comparatively easy to measure. All in all are the defect tracking techniques the least complex approaches for estimating release readiness.

C. Availability

Defect tracking is only available from the verification phase of software development. That is, when the implementation of the software has finished and testing starts. Comparatively, SRI only delivers meaningful values in the later phases of the development. This is because the approach is heavily focusing on defect related criteria. ShipIt on the other hand is designed to be available in every phase of the development - from the beginning to the release. The approach can measure the progress to release readiness at any point in the project. Missing actual data (for the lines of code for example) can be estimated fairly early. Also, since absent data resembles 0 percent of completion, ShipIt can still deliver meaningful readiness values with missing criteria.

Software Engineering Conference (MySEC), 2014 8th Malaysian , vol., no., pp.166,171, 23-24 Sept. 2014
doi: 10.1109/MySec.2014.6986008

TABLE IV. SUMMARY OF THE COMPARISON BETWEEN THE THREE RELEASE READINESS METRICS.

	Scope			Complexity	Availability	Restrictions		
Approach	Measurements	Focus	Thresholds	# Criteria	Dev. Phase	Architecture	Environment	Dev. Model
Defect Tracking [4]–[6]	Reliability	Reliability	No	1 [4] / 18 [5], [6]	Testing	All [4] / N-Tier [5], [6]	All [4] / OO [5], [6]	All
SRI [8], [9]	Reliability Quality	Quality	Yes	22	Coding	All	All	All
ShpiIt [10]	Reliability Quality Progress	Progress	No	30	Analysis	All	All	Waterfall

D. Restrictions

Due to their simplicity defect density, pooling and seeding can be applied in any software project which employs testing. Architectural defect tracking on the other hand clearly requires an object-oriented environment in contrast business logic metrics cannot be raised. Furthermore, the technique assumes a software can be divided into data access, business logic and presentation tiers. While the architecture is common in current software projects, in general, a clear separation between such tiers are hard to recognize.

ShipIt requires a software development process which orientates on the waterfall model. However, today agile processes become much more common. As a result the necessary conditions that the requirement analysis or the designing process is strictly separated from the coding may not be given.

SRI neither requires a specific development process model nor does its criteria require object orientation. In this way the approach is valid for any kind of software product.

VII. CONCLUSION

All three approaches come with advantages and disadvantages. In the end the best approach to choose depends on the situation of the project. Defect tracking has the least criteria to measure for release readiness determination. However, it should only be used in a project phase when everything besides software testing is out of question. Here, also the architectural defect tracking is an option if the software is implemented in an object oriented environment and sticks to the required n-tier model.

ShipIt is the most complex and at the same time the shallowest of the three approaches. ShipIt covers practically the progress of the whole development process but does not detail important aspects like quality and reliability. The approach supports giving an overview over the progress of the project but loses its usefulness for release readiness estimation in later phases where quality and reliability aspects become increasingly important.

The SRI generally seems to be the most comprehensive approach for most situations. In the later phases, where the question of release readiness becomes more important, the SRI offers meaningful criteria for tracking the quality of the product in term of software and hardware as well, reliability included. Also the approach supports decision making by providing Green and Red thresholds. Furthermore, the manager can trade off between precision and complexity by choosing the basic version of SRI. All these features and lack of restrictions make the SRI very versatile and the best practice of measuring the release readiness in many situations.

Overall, our comparison suggests that the measurements of release readiness is inadequate and non-holistic measurement is applied in literature. Consequently, when-to-release software still remains an unclear question in software organization. Release readiness measurement will be adequate and acceptable if the metrics include software quality and progress of the whole development process. This holistic measurement will give much better decision to release engineer in when-to-release software in the future.

REFERENCES

[1] C. Plewnia, A. Dyck, and H. Lichter, "On the influence of release engineering on software reputation." Mountain View, CA, USA: In 2nd International Workshop on Release Engineering, April 2014. [Online]. Available: http://releng.polymtl.ca/RELENG2014/html/proceedings/releng2014_submission__3.pdf

[2] J. Ho, S. Shahnewaz, and G. Ruhe, "A prototype tool supporting when-to-release decisions in iterative development." Mountain View, CA, USA: In 2nd International Workshop on Release Engineering, April 2014. [Online]. Available: http://releng.polymtl.ca/RELENG2014/html/proceedings/releng2014_submission__11.pdf

[3] J. Musa, A. Iannino, and K. Okumoto, *Software reliability: measurement, prediction, application*, ser. Software engineering series. McGraw-Hill, 1990. [Online]. Available: http://books.google.de/books?id=ucNQAAAAMAAJ

[4] S. McConnell, "Best practices: Gauging software readiness with defect tracking." *IEEE Software*, vol. 14, no. 3, pp. 135–136, 1997. [Online]. Available: http://dblp.uni-trier.de/db/journals/software/software14.html#McConnell97b

[5] J. Quah and S. Liew, "Gauzing software readiness using metrics," in *Soft Computing in Industrial Applications*. IEEE, June 2008, pp. 426 – 431.

[6] T.-S. Quah and M. M. T. Thwin, "Utilizing computational intelligence in estimating software readiness." in *IJCNN*. IEEE, 2006, pp. 2999–3006. [Online]. Available: http://dblp.uni-trier.de/db/conf/ijcnn/ijcnn2006.html#QuahT06

[7] D. Hoyle, *ISO 9000 Quality Systems Handbook: Using the Standards as a Framework for Business Improvement*. Butterworth-Heinemann, 2009. [Online]. Available: http://books.google.de/books?id=HWNWdBisJcoC

[8] J. Olivieri, "Hardware and software readiness: A systems approach," in *Systems Conference (SysCon)*. IEEE, March 2012, pp. 1 – 6.

[9] A. Asthana and J. Olivieri, "Quantifying software reliability and readiness," in *Communications Quality and Reliability*. IEEE, May 2009, pp. 1 – 6.

[10] P. R. Satapathy, "Evaluation of software release readiness metric [0,1] across the software development life cycle," 2008.

[11] B. Boehm, *Software Engineering Economics*, ser. Prentice-Hall Advances in Computing Science & Technology Series. Pearson Education, 1981. [Online]. Available: http://books.google.de/books?id=VphQAAAAMAAJ

[12] A. J. Albrecht and J. E. Gaffney, "Software function, source lines of code, and development effort prediction: A software science validation," *IEEE Trans. Softw. Eng.*, vol. 9, no. 6, pp. 639–648, Nov. 1983. [Online]. Available: http://dx.doi.org/10.1109/TSE.1983.235271

[13] H. Sandoh, "Reliability demonstration testing for software," *Reliability, IEEE Transactions on*, vol. 40, no. 1, pp. 117–119, 1991.

Software Engineering Conference (MySEC), 2014 8th Malaysian , vol., no., pp.166,171, 23-24 Sept. 2014
doi: 10.1109/MySec.2014.6986008