European Journal of Computer Science and Information Technology
Vol.6, No.1, pp.33-43, February 2018
Published by European Centre for Research Training and Development UK (www.eajournals.org)

MEASURABLE METRICS FOR SOFTWARE PROCESS IMPROVEMENT

Supun Dissanayake

University of Colombo School of Computing

ABSTRACT: *Software Engineering organisations should adapt new technologies for software process improvement (SPI) due to the advancement of technologies. Since big data processing is widely required for new organisations, it is required to build new technologies to reduce manual workloads and automate processes. Therefore, this research paper focuses on identifying measurable metrics that can be used by software development companies to improve the quality of the development process and their products by quantifying the maturity level achievement within the organisation.*

KEYWORDS: Software Engineering, Software Process Improvement, Measurable Metrics, Key Process Indicators, Maturity Models

INTRODUCTION

Software Process Improvement is the improvement of business processes in a software development organisation (Olson et al, 1989). Paulk et al (1993) define software processes as activities, practices, transformations and methodologies that allow development and sustenance of software products. Moreover, it improves the productivity of software and reduces development times. This implicates that the successful application of SPI across all sections of the organisation will allow the business to thrive against its competitors.

Prior to the application of SPI, it is vital to identify a structure of a software engineering organisation (Paulk et al., 1993). This allows SPI to be separated in terms of development processes, reliability management, cost analysis etc. Then measurable metrics could be identified to apply SPI.

Implementation of maturity models within the organisation is regarded as one of the most effective strategies to apply SPI (Sommerville, 2007). It allows the identification of areas of improvement in a software development company, thus implicating level of quality within a company (Herbsleb, 1997). Carnegie Mellon University introduced widely accepted maturity models for the software engineering industry in the form of CMM in 1989 and CMMI in 2002 (Bayrasken, 2009). Conversely, maturity models such as ICMM and SCMM can be used to improve CBSE practices. Moreover, maturity models such as ISO/IEC WD 15504 and SCMM are used in the software development industry. These models provide Key Process Areas (KPA) to achieve maturity criteria, however, most of these KPA are either too vague or lack the ability to quantify the achievability of the maturity level. Therefore, this research paper identifies measurable metrics that can act as measurable Key Process Indicators (KPI) for KPAs. Thus, organisations can produce a measurement of achievement for each KPA.

Software Process Improvement

Software Process Improvement (SPI) has obtained great attention from the software engineering industry during past few decades (Sjoberg et al, 2007). Software processes can be defined as human-centred functions, which can cause unexpected or unintentional behaviours

ISSN 2054-0957 (Print), ISSN 2054-0965 (Online)

with the organisation (Fuggetta, 2000). Hence, software quality identification is a very significant aspect of the organisation. Pressman (2009) denotes three subsections of software quality: good software development processes, setting up of quality standards and identification of functional requirements. Moreover, IEEE (Galin, 2004) defines software quality as the requirement achievements of the organisation in terms of processes, system and components. Therefore, software processes in the organisation should be continuously checked to validate their efficiency. Moreover, SPI provides a platform to carry out necessary evaluation processes to show the efficiency of software processes in the organisation (Unterkalmsteiner et al, 2011). Hence, weak areas of the development process can be identified for improvements. Furthermore, SPI had been widely evaluated for software development processes during past few decades and its strengths and weaknesses were meticulously identified (Salo, 2007).

Identification of Required Measurable Metrics

Prior to looking for measurable metrics, it is required gather relevant information from personnel who work in software development companies through a qualitative research methodology. Therefore, interviews were carried out with 10 personnel who are working in software development organisations.

Interviews collect participant's honest opinions about the subject area and provide a wide spectrum of information for each question (Kvale, 1996). The interview contained a semi-structured style of questions with open-ended answers to gather the maximum amount of information. Semi-Structured interviews allowed the participants to express their true opinions and feelings towards each question depending on their understanding of the question and personal experiences (Matthews, 2010). Therefore, these answers were used to build a valid hypothesis for this research study.

The participants were identified and contacted via LinkedIn and all the participants are professionals who are working in the software engineering industry, thus results are valid and accountable. Interview questions were designed to identify structures of organisations in terms of development methodologies, employees, culture etc. Moreover, participants could stay anonymous in the interview since substantial information about the participants' organisation, employees and development processes were gathered through this interview.

The interview was used to identify valid measurable maturity metrics that is required in the software development industry. It achieved this feature by identifying the use of SPI methodologies in companies and identifying areas of improvements suggested by participants. Therefore, this information was used to hypothesise measurable metrics that should be adopted by maturity models like CMMI, SCMM, ISO etc. to enhance their effectiveness. Following questions were asked from the participants and their results were thoroughly analysed.

Please specify the business area(s) of your organisation?

This question was developed to confirm that the participant is working in a software development company. This is a vital piece of information since it depicts the validity of the opinion received from the participant. Moreover, it identifies the type of organisation that the participant is working in the industry (e.g. Software Testing Company, IT Consultant, Software Development Company, Web Development Company). Thus, if there is a huge variation in answers to next set of questions, this question validates its reasoning. 100% of the answers collected for this question suggested that participants are working in a software development

ISSN 2054-0957 (Print), ISSN 2054-0965 (Online)

organisation. Therefore, this answer validates results gathered for all the questions since participants have knowledge of software development and software process improvement methodologies followed by their respective companies. This implicates the validation of results gathered for the next set of questions and there is no need to be concerned about answer variations.

What types of data does your organisation collect from employees?

This question was developed to identify whether the organisation is collecting measurable information from their employees. (e.g.: break times, working efficiency, work delivery time etc.). The answer to this question helps the identification of measurable maturity metrics for Key Process Areas (KPA). Thus, these results can be used to identify valid measurable metrics through the literature. There was a common theme in all the answers to this question. Almost all the respondents answered this question by saying that the organisation collect working days and hours. They said they are unaware of further data that organisations collect from their employees. This might be due to their lack of understanding of managerial criteria of the organisation since none of these participants hold managerial roles in their companies. This clearly implicates that there is a larger research area for identifying measurable metrics from individual employees to improve software process maturity. However, one participant elaborated by saying that his organisation collects Agile metrics such as "Actual Stories Completed vs. Committed Stories, Communication, Technical Debt Management etc.". Therefore, these could be viable metrics that could be used for the development of KPI for the maturity model. Thus, measurable metrics can be identified for these areas to identify methodologies to improve employee work rates.

What types of data does your organisation collect from development processes?

This question was developed to identify whether the organisation is collecting measurable information from their development processes. The answer to this question benefits the development of KPIs that are related to the development process efficiency. Moreover, it helps to identify and propose further data that organisations can collect from their development processes to benefit their overall SPI by comparing collected data with information analysed in the literature review. There were a wide variety of answers obtained for this question. They are,

Fig 1: Development Process Data

ISSN 2054-0957 (Print), ISSN 2054-0965 (Online)

Therefore, certain aspects of these categories were used to identify measurable metrics for a maturity model. These answers clearly implicate organisations use both agile and Component-Based Software Engineering. Moreover, mathematical metrics should be identified to measure the efficiency of most of the aspects identified through this question.

The critical analysis of the interviews allowed the identification of measurable metrics for a software development company. Hence, the data gathered from this qualitative research procedure was used to identify mathematical metrics to provide numerical feedback for SPI.

Measurable Metrics for software process improvement

The qualitative analysis identified that the mathematical metrics should be identified in terms of reusability, cost analysis, development efficiency, reliability etc. to measure SPI in a software development organisation. Moreover, it is vital to identify measurable metrics that support SPI in terms of Agile development and CBSE since these methods are widely used in the industry. Following sub sections depict these identified mathematical metrics and their evaluations.

Development method metrics

Reusability Assessment

Frakes (1995) depicts that reuse metrics identifies and monitors software reuse levels by assessing the percentage of reuse life-cycle objects for a period of time. This allows the organisation to understand that they are following a good amount of software reuse practices to improve development process efficiency. This metric can be measured using following equation,

$$\boldsymbol{Reuse\ Improvement\ Effort} = \frac{Amount\ of\ life\ cylce\ objects\ reused}{Total\ size\ of\ the\ cycle\ object}\ X\ 100\%$$

However, this metric can be simplified to identify lines of code reused in a software product. Lines of code (LOC) can be used to measure the reuse code percentage to make the development process more efficient (Dubey et al, 2015). Moreover, it improves the understandability, maintainability and the reusability of the code (Lorenz and Kidd, 1994). Lorenz and Kidd (1994) denote that it is not a strongly endorsed metric to be used for object-oriented systems, however, it is easy to measure thus it is widely used in the industry. Mathematical metric for LOC is (Dubey et al, 2015),

$$\boldsymbol{Percentage\ Reused}\ (\%) = \frac{Lines\ of\ reuse\ code\ in\ the\ system\ module}{Total\ lines\ of\ code\ in\ the\ system\ or\ module}\ X\ 100\%$$

Complexity

Identification of coupling, constraints and cohesion would allow the measurement of software complexity (Patel et al, 2016). Moreover, as the software component complexity increases, the quality of the software will automatically decrease. Component coupling, which is the identification process of relationship between classes by identifying internal structures of components can be used to measure component complexity (MajdiAbdellatiefab et al, 2012). Patel et al (2016) depicts the metric for Component Coupling,

ISSN 2054-0957 (Print), ISSN 2054-0965 (Online)

$$\textbf{Component Coupling} = \frac{No\ of\ other\ component\ sharing\ attributes\ or\ methods}{Total\ \ sharing\ pairs\ in\ the\ CBSE\ Application}$$

Moreover, as the complexity increases, components will develop more testing and debugging issues. Therefore, the application of constraints and configuration metrics should be used to measure these complexity levels (Patel et al, 2016).

$$\textbf{Constraints Complexity} = \frac{No\ of\ Constriants}{No\ of\ properties\ and\ operations\ in\ an\ interface}$$

$$\textbf{Configuration Complexity} = \frac{No\ of\ Configurations}{No\ of\ context\ of\ use\ of\ the\ component}$$

Therefore, these metrics can be used to mathematically identify software complexity.

4.1.3-Reuse Cost Analysis

Cost analysis was another measurable that was identified though the qualitative research. This could be achieved by following metrics proposed by Gaffney and Durek (1989). These metrics can be used to analyse the cost of software component reuse.

C= software development cost relative to new code (where C =1).

R= reuse code proportion in the software product (R<=1).

b= the cost that is relative to newer code merging the reuse code when developing a software product (b =1 for newer code).

Therefore, the mathematical metric for this process is (Frakes, 1995),

$$\textbf{Cost} = (1)(1-R) + (b)(R)$$

Therefore, the productivity is (Frakes, 1995),

$$\textbf{Productivity} = \frac{1}{Cost}$$

It is important that b should be less than 1 for the reusable code to be cost-effective and its size is purely dependent upon its reusable life-cycle (Frakes, 1995).

Therefore, these metrics can be used for evaluation of cost and productivity in a software development company.

Work in Progress (WIP)

WIP is the process of tracking number of developing processes that are currently in progress (Little et al, 2008). These processes should be constantly tracked by software development organisations to improve the overall flow of the development process. Some organisations use Kanban boards to track development progress of their organisations. Hence, WIP can be

ISSN 2054-0957 (Print), ISSN 2054-0965 (Online)

measured by adding the number of unfinished processes on the Kanban Board or by analysing a Cumulative Flow Diagram using a Kanban board software.

X= unfinished tasks

$$\boldsymbol{WIP} = \sum_{i=1}^{n} X_i$$

This was another aspect that was identified through the qualitative research, thus it satisfies the research scenario. However, it is important to denote that WIP is not enough for SPI since it is only one of the factors that can be used to improve the development life cycle (Little et al, 2008). Therefore, it is required to calculate the average throughput.

Throughput

Throughput is the average output of development processes per unit time (Little et al, 2008). For example, many processes are completed per day, week, month etc. It is important to recognise and outline throughput depending on the way it affects the economy of the development process; also, outliers that could become effective must be identified (Little et al, 2008). Therefore, this metric aids business decisions made by the organisation. Throughput can be calculated through the following equation.

$$\boldsymbol{Throughput} = \frac{No\ of\ processes\ completed\ per\ day,\ week,\ month\ etc.}{Time\ period\ (day, week, month\ etc.)}$$

However, since the throughput is an average value, the forecast made only using throughput does not seem to be a very reliable result (Little et al, 2008). Hence, it is very important to combine throughput with cycle time and lead time to satisfy Little's Law.

Cycle Time

Cycle time is the average time that it takes for the development process to go from the initiation to completion (Little et al, 2008). Therefore, the reduction of cycle time is beneficial for the organisation since it allows the business to meet its deadlines in a quicker time span. Cycle Time can be calculated by following Little's Law using measurements identified in previous two subsections.

$$\boldsymbol{Cycle\ Time} = \frac{Work\ in\ Progress\ (WIP)}{Throughput}$$

This metric depicts a hypothetical demo for the process development time. Moreover, it clearly implicates that the increase in WIP can cause an increase in predicted Cycle Time. For example, if there are 25 processes to be completed (WIP) and the throughput is 1.5 per day, the average cycle time will be 16.67. However, if the throughput stays the same and the WIP is decreased to 20, the cycle time becomes 13.33, which is an improvement for the business. It can be argued that the increase in throughput can also decrease the cycle time, however, this is proven to be a difficult practice in real-life business scenarios since it causes costs to increase (Little et al, 2008).

ISSN 2054-0957 (Print), ISSN 2054-0965 (Online)

Lead Time

Lead time is the measurement of the time taken to deliver the product from the customer request (Muharremoglu et al, 2003). It is the duration period from the project initiation to the end, which includes process times, queue time, delays etc. The measurement of this metric determines influences that changes have made to the development process. It can be calculated by following mathematical metric (Muharremoglu et al, 2003),

$$\boldsymbol{Lead\ Time} = \text{Cycle Time X WIP}$$

It allows SPI for a similar project in the future through the development and changes to the inputs to make the lead time faster. Moreover, it identifies the causes and effects of the organisation that could affect the end-product. Hence, it allows the organisation to inform new customers about the exact time frame that the organisation will be able to deliver the product compared to making educated guesses about the delivery date.

Mean time to IPL (MTI)

Reliability of software and software systems is an extremely important factor for software development processes since it allows the software development processes to run smoothly. Therefore, organisations must track system and software reliability and take measures for improvements. Therefore, Kan et al (2001) propose Mean time to IPL (MTI) as a mathematical metric to achieve this principle. This calculation requires the division of CPU running hours per week with a number of unplanned breakdowns plus 1. Moreover, it uses weighting factors to depict results from previous weeks to make the result more meaningful. The equation for this metric is Kan et al (2001),

$$Weekly\ MTI_n = \sum_{i=1}^{n} W_i * \left(\frac{H_i}{I_i + 1}\right)$$

n= week no

W= weighting factor

H= weekly running hours of CPU

I= Unplanned IPLs resulted from system failures per week

Software Quality Metrics

After identifying measurable metrics that are applicable for development processes, the next step is to identify measurable metrics for software quality analysis. Therefore, this section critically evaluates metrics that can be used to improve software efficiency.

Weighted method per class (WMC)

WMC is the sum of total methods in a class or the method complexity, which is measured via cyclometric complexity (Harrison et al, 2001). Hence, the identification of WMC allows the prediction of time and effort needed to build and preserve a class. Moreover, if a class have

ISSN 2054-0957 (Print), ISSN 2054-0965 (Online)

more methods, the inheritance level for child classes will be higher (Chidamber and Kemerer, 1994). Therefore, it will hinder the reusability factor of the application. Moreover, WMC will allow the identification of maintainability, understandability and reusability (Chidamber and Kemerer, 1994). The mathematical metric for WMC is,

$$\boldsymbol{WMC} = \sum_{i=1}^{n} C_i$$

c= number of methods

4.2.2-Depth of Inheritance Tree (DIT)

The maximum length between the class node and the root of the tree can be explained as a depth of a class within its inheritance order (Chidamber and Kemerer, 1994). Hence, the summation of ancestor classes is used to measure this metric. The complexity to predict the class behaviour rapidly increases if the depth of the class increases (Chidamber and Kemerer, 1994). Deeper trees consist of more methods and classes and therefore they cause higher design complexities, however, it increases the potential for reuse methods (Harrison et al, 2001). DIT is mainly used for the evaluation of reuse, understandability, efficiency and testability (Harrison et al, 2001). Therefore, the DIT can be measured through the following metric,

$$\boldsymbol{DIT} = \sum_{i=1}^{n} A_i$$

A= number of ancestor classes that can potentially affect the selected class

4.2.3-Number of Children (NOC)

NOC is the immediate subclasses in a class hierarchy; it can influence software classes (Harrison et al, 2001). The inheritance is a type of reuse, therefore, higher the NOC, higher the reusability factor of the class. However, the testing time rapidly increases with increasing number of children (Chidamber and Kemerer, 1994). Thus, it implies that NOC is a valuable metric to measure testability, efficiency and reusability. Therefore, a viable metric for NOC should be,

$$\boldsymbol{NOC} = \sum_{i=1}^{n} X_i$$

Coupling between object classes (CBO)

CBO is the summation of classes that are coupled with a particular class (Chidamber and Kemerer, 1994). It can be measured by adding different non-inheritance associated class hierarchies that are dependent on the software (Chidamber and Kemerer, 1994). Therefore, higher the CBO, lower the reusability of the class and the maintainability (Harrison et al, 1997). Moreover, strong coupling makes it more difficult to understand the functionality of the class (Chidamber and Kemerer, 1994). This implicates that classes with weak coupling allow the complexity to reduce. Therefore, this improves modularity, encapsulation and promotes reusability and efficiency of software. Following metric allows the quantification of CBO (Harrison et al, 1997),

ISSN 2054-0957 (Print), ISSN 2054-0965 (Online)

$$\boldsymbol{CBO}\ (\%) = \frac{Number\ of\ links\ between\ classes}{Number\ of\ classes}\ X\ 100\%$$

Lack of Cohesion of Methods (LCOM)

LCOM is a quality measurement metric for the class cohesiveness, which can be achieved by measuring common attributes in multiple methods. It calculates the level of similarity between methods through data input variables or attributes (Harrison et al, 2001). Cohesion is the relationship of methods within a class (Chidamber and Kemerer, 1994). When the cohesion increases, the effect of encapsulation increases, which is good for object-oriented design (Chidamber and Kemerer, 1994). If a class have low cohesion, the complexity rises, thus more errors will occur during the development procedure (Harrison et al, 2001). Therefore, those classes that have low cohesion should be reduced to a couple of subclasses to increase cohesion. High cohesion clearly depicts the good nature of class subdivision (Chidamber and Kemerer, 1994). LCOM metric evaluates reusability and efficiency of the software product.

Table I: SPI Metrics Summary

Table 1: Application of measurable metrics for business processes

Process	Measurable Metrics for SPI
Reusability	Reuse Improvement Effort
Development Efficiency	WIP, Throughput, Cycle Time, Lead Time
Cost	Cost Analytics and Productivity Analytics, Productivity
Reliability	MTI
Software Quality	WMC, DIT, RFC, NOC, CBO, LCOM
Complexity	Component Coupling, Constraints Complexity, Configuration Complexity

$$LCOM(C) = \begin{cases} P - Q, & if\ P > Q \\ 0, & Otherwise \end{cases}$$

P = methods with no common attributes with C

Q = methods with one or more common attributes with C

This section clearly depicts existing SPI metrics in the software development industry that can improve development processes and software quality. Since they were identified to match the qualitative research results implies their validity and relevance to the software engineering industry. Moreover, the measurable nature of these mathematical metrics will benefit organisations to quantify the level of maturity in areas where they constantly gather information. Table 1 depicts main areas of data gathering identified through the interview

ISSN 2054-0957 (Print), ISSN 2054-0965 (Online)

process and mathematical metrics that can be used to identify their maturity. Thus, the effectiveness of these processes can be measured and improved.

Conclusion and Future Work

This research study proposes the use of measurable metrics to improve the software development processes in software engineering companies. It depicts that the lack of measurable criteria in current maturity models provides a vague impression for improvement areas for organisations that use these models. Hence, this paper proposes the use of measurable metrics as Key Process Indicators (KPI) for Key Process Areas (KPA) in a maturity model.

Moreover, it was recognised that big data analytics in organisations has become a vital functionality due to the growth of digital data processing in past few decades. Thus, combining big data analytics with mathematical measurable metrics to identify process maturity can largely benefit software development originations.

Qualitative research was carried out with 10 personnel who are working in software engineering industry to validate the rationale of the research study to identify structures of the software engineering organisation and measurable metrics that companies obtain from their employees/ development processes. Thus, information gathered through these interviews were used to research mathematical SPI metrics that could be used to measure maturity levels of a software development organisation. The application of these mathematical metrics through a maturity model would clearly allow the organisation to quantify the maturity level.

Therefore, this research could be further enhanced by developing a maturity model that contains purely measurable Key Process Areas to depict the organisation maturity. Thus, mathematical metrics identified in this research study could be used to achieve this process. Moreover, it could be further enhanced through the application of machine learning algorithms to automatically predict future changes to these maturity levels since these mathematical metrics provide numerical data sets from the existing data. Hence, quantifying the maturity identification will allow event prediction by completely revolutionising the software process improvement.

REFERENCES

A. Dubey, H. Kaur, “Reusability Types and Reuse Metrics: A Survey”, in: International Journal of Computer Applications, vol 131, 2015, pp. 12–16.

A. Fuggetta, “Software process: a roadmap”, In Proceedings Conference on The Future of Software Engineering, Limerick, Ireland, 2000, pp. 25– 34.

A.Muharremoglu, J. Tsitsikli, “ Dynamic Leadtime Management in Supply Chains, working paper”, Graduate School of Business, Columbia University, New York, 2003.

B.Matthews, (2010), “Research methods: a practical guide for the social sciences” [Online] Harlow: Longman. Available from: < https://www-dawsoneracom.ezproxy.leedsmet.ac.uk/readonline/9781408226186> [Accessed on 15 August 2017]

D. Galin, “Software Quality Assurance: from Theory to Implementation”, Pearson Education Limited, United Kingdom, 2004

D. Sjoberg, T. Dyba, M. Jorgensen, “The future of empirical methods in software engineering research”. Future of Software Engineering (FOSE), Minneapolis, 2007, pp. 358–378.

ISSN 2054-0957 (Print), ISSN 2054-0965 (Online)

H. Bayrasken, "A report on the capability maturity model**"**. Research Paper. Computer Science Department, University of Nottingham, 2009.

I.Sommerville, "Software Engineering", 8th edition, 2007, pp- 6-56.

J. E. Gaffney, T. A. Durek "Software reuse— key to enhanced productivity: some quantitative models". Inf. Softw. Technol. 31, 5, 1989, pp-258 –267.

J. Herbsleb, D. Zubrow, D. Goldenson, W. Hayes, M. Paulk "Software quality and the capability matwurity model" Communications of the ACM 40(6), 1997, pp- 30–40.

J. Little, S. Graves, "Little's law. Building Intuition", Research Paper, Chapter 5, Massachusetts Institute of Technology, 2008, pp- 81–100.

M. Paulk, B. Curtis, M. Chrissis, C. Weber, "Capability Maturity Model for Software, Version 1.1". Research Paper. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1993.

M. Unterkalmsteiner, T. Gorschek, A. Islam., R. Permadi, R. Feldt, " Evaluation and Measurement of Software Process Improvement - A Systematic Literature Review", IEEE Transactions on Software Engineering, 2011 .

MajdiAbdellatiefab, A. Sultana, A. AzimAbdGhania, M. Jabara, "Component-based Software System Dependency Metrics based on Component Information Flow Measurements", ICSEA: The Sixth International Conference on Software Engineering Advances, 2012.

O. Salo, "Enabling Software Process Improvement in Agile Software Development Teams and Organizations". VTT publications, 2007.

R. Harrison, S. Counsell, R. Nithi, "An overview of object-oriented design metrics". In International Conference on Software Technology and Engineering Practice, (STEP), IEEE Computer Society Press, 2001, pp 230–234.

R. S. Pressman, "Software Engineering: A Practitioner's Approach", 7th ed., New York: McGraw-Hill International, 2009.

S. H. Kan, J. Parrish, D. Manlove, "In-Process Metrics for Software Testing", IBM Systems Journal 40, No. 1, 2001, pp. 220– 241.

S. Kvale, "Interviews: An Introduction to Qualitative Research Interviewing", 1 Edition. SAGE Publications,1996.

S. Patel, J. Kaur, "A Study of Component Based Software System Metrics", in: International Conference on Computing, Communication and Automation (ICCCA2016), 2016, pp. 824–828.

S. R. Chidamber, C. F. Kemerer, "A metric suite for object oriented design", IEEE Transactions on Software Engineering, pp. 476–493.

T. Olson, W. Humphrey, D. Kitson, "Conducting SEI-Assisted Software Process Assessments". Technical Report, CMU/SEI-89-TR 7, Pittsburgh, 1989.

W. B. Frakes, "Software reuse. In Encyclopedia of Microcomputers", A. Kent and J. G. Williams, Eds. Marcel Dekker, Inc., New York, NY, 1995, pp. 179–184.

ISSN 2054-0957 (Print), ISSN 2054-0965 (Online)