Software Development Life Cycle early phases and quality metrics: A Systematic Literature Review

Journal of Physics: Conference Series **1694** (2020) 012007 doi:10.1088/1742-6596/1694/1/012007

Software Development Life Cycle early phases and quality metrics: A Systematic Literature Review

Shokhista Ergasheva, Artem Kruglov
Russia, Innopolis University
E-mail: s.ergasheva@innopolis.university

Abstract. Most of the currently used software development metrics are concentrated on the latter stages like development and testing. However, early revealing of errors during the SDLC(Software Development Life Cycle) tremendously affects the efficiency of the team work by spending more time on prevention and less on correction in later stages.

Furthermore, reworking in later stages increase the cost of quality, lead to extra waste of time of the development team. The objective of this review is to examine the classification of the existing SDLC(Software Development Life Cycle) early phases and define the set of software process quality metrics. Based on the SRL research protocol, we selected the most relevant studies from overall 200 publications by the use of search keywords and inclusion/exclusion criteria for quality assessment of primary studies. This systematic literature review yields the correlation of cost, time and software product quality with the SDLC stages.

1. Introduction

For the past few decades, the software development methodologies have been evolving intensively. Newly created Agile methodologies have changed the concept of SDLC and Software development methods implementation. Software Development Life Cycle (SDLC) is the time required for the activities such as defining, developing, testing, delivering, operating and maintaining the software or the system. The development team's productivity and the software quality depend on the effectiveness of defining and analyzing the software process metrics throughout the SDLC. Early defects detection in early phases of the software process is one of the sources in the way to a successful project. However, the classification of early phases can vary referring to the methodologies the company uses. Hence, methods for assessing and evaluating the quality of the software process depend on the company preferences. Therefore the set of measurements that should be tracked during the evaluation process differ as well. This paper demonstrates the classification of SDLC phases, in particular early phases, sundry software quality evaluation methodologies and set of measurements.

Therefore, we decided to conduct a Systematic Literature Review of Software Development Life cycle phases and metrics that can be collected during early phases of software process. This research will help us further sort metrics according to the process methods, define development stages of assessment methods and to create the base set of metrics to predict the efficiency of the team and the process.

1.1. Research method

Research methods we chose research questions on software development life cycle phases and metrics to perform the review of relevant literature to address in this SLR. Referring to [1] first we defined search strategy to detect as much of the relevant literature as possible, then we use explicit inclusion and exclusion criteria for assessing potential primary study.

The aim of this Systematic Literature Review is to obtain information about existing types of phases in software development process in variety SDLC's (Software Development Life-Cycle) from primary study. Later in this section, the quality criteria to evaluate the primary study will be also provided.

1.2. Research questions

The following Research Questions for definition and analysis of the relevant literature were formulated:

(i) What is the classification of the software development life cycle phases?
(ii) What are the existing models/approaches to assess and evaluate software quality in early phases?
(iii) What activities are performed in the software development life cycle phases?
(iv) What metrics are collected during the software development life cycle phases?

2. Review protocol

This section describes the set of activities performed to answer the formulated research questions during this SLR. Firstly, search strategy was defined by setting the search parameters to use for search strategy. Search parameters include:

- "Software development life cycle phases and quality metrics"
- "Software development early life cycle phases and metrics"

According to Kitchenham Achimugu [1], we used enhances strategy of deriving major terms from the research questions.

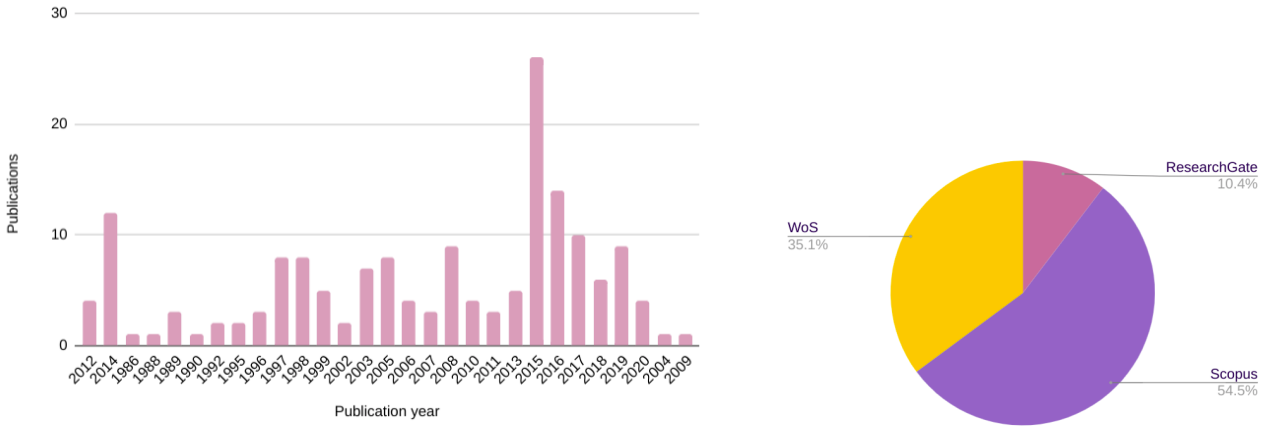
Data sources for searching the search terms were selected based on the availability of free libraries, which are:

- Scopus
- ResearchGate
- Web of Science

As a result, we get more than 200 publications in the listed open-source libraries above. The results of the search terms were sorted into a primary selected studies in accordance with the selection instructions that is proposed in [2]. The approach includes several steps to conduct a qualitative assessment of the publications selected:

(i) First, we selected the initial set of publications with automatic and manual search techniques;
(ii) Second, studies were selected as primary studies according to their related keywords, title and abstract.
(iii) Third, the primary studies were then scanned in a details by fully reviewing the papers.

The studies were not selected on the report of the year of publication. In the Figure **??** Owing the fact that the field of early stages of SDLC is not fully researched yet. During the SLR, we applied exclusion criteria of eliminating the duplications, irrelevant and not complete studies. The studies that do not represent exact separation of stages of SDLC were not included.



(a) (b)

Figure 1: Distribution of publications a) by publication year; b) by data sources retrieved from

2.1. Data extraction and Quality assessment

Due to the above mentioned selection criteria, the first selection of studies was conducted. The first selection included a string of keywords insertion into data sources. As a result, we collected more than 200 publications selected distribution of which is as following(also see **??**): 110 publications from Scopus, 70 from Web Of Science and 22 publications from ResearchGate.

In the first step of studies selection according to title and abstract, only 29% of studies were accepted and 5% were eliminated for duplications. The next step were conducted by reading the whole publication with details.

2.2. Results

This Systematic Literature Review assess software metrics, models and methods to track and analyze software quality in early phases, concentrating mainly on Requirements management and Design phases of SDLC.

In the first iteration proceeded by inserting search strings to open-source data libraries,slightly over 200 publications were selected. Whereas, in the second iteration where we analysed Title, Keywords and Abstract we sorted 60 primary studies related to our topic of interest. After scanning primarily selected studies whole content in detail, they were classified from 0-2 scale in the order of relevance(0-not relevant,1-partially relevant, 2-relevant) in Table 1. From the whole set of papers 6,9% works were eliminated as duplication.

NotApl	0	1	2	NA
156	12	11	13	10

Table 1: Publications classification of relevance to SLR

Overall, studies under consideration were of types 2,5% Journals, 31,3% Articles, 57,2% Conference paper and 9% Book chapter.

3. Results

In this section, the results of conducted review is provided. By analyzing the existing software process early phases, methodologies and metrics to assess and evaluate the quality of the software we have answered all four research questions.

3.1. What is the classification of the software development life cycle phases?

During this SLR(Systematic Literature Review) we defined a general set of phases and a set of metrics that are applicable to track and analyze software quality; these can be metrics of code [3], of design [4], or of system as a whole [5], also considering the process to collect them [6–10], the underlying software model [11–14], the target system [15–17], their usage in building models [4, 18–30], the reference licensing mode [31–36].

In general, almost all of the studies elaborated software life early phases into Requirements Management and Design phase, and sometimes Code. The publications weight representing the respective early phases of software life is described in the following list:

- Requirements phase - 17,2%
- Requirements and Design phases - 24,1%
- Design - 31,0%
- Design and Code phases - 3,4%
- Code and Testing phases - 3,4%
- All phases - 17,2%

But some of the papers defined phases of software process differently. As the paper [37] defines the software development life cycle early phases includes User Needs Analysis, Definition of the Solution Space, External Behavior Definition and Preliminary Design. Where the first three stages union is the Requirements stage. It consists of all the activities up to the decomposition of the software architectural components.

Whereas,researchers in [37] have established typical software Life Cycle early phases preceded by these phases listed below:

- Initial planning phase - the technical and economic basis for the project should be established
- Analysis - the functional performance requirements for the software configuration items are defined. This phase's result is the successful completion of Preliminary Design Review (PDR)
- Design - this phase include the allocation of requirements to software components and culminates with complete Critical Design Review (CDR)

As it can be seen here, the initial planning and analysis phases stated in [37] can be traced to Requirements Management phase according to the activities proceeded during these phases.

3.2. What are the existing models/approaches to assess and evaluate software quality in early phases?

In particular there are some studies and surveys suggesting models or approaches to assess and evaluate the software quality in early phases. One of these kinds of researches that analysed Requirements documentation can be Aversano et al [38], which performed assessment of the documentation of ERP(Enterprise Resource Planning) open source systems to understand the high quality documentation. They analysed the quality in terms of two aspects: Structure Quality and Content Quality. The paper highlights that the documentation is composed of documents that can be of the different kinds, API, Wiki, text and code comments. First, Structure Quality indicators are listed in the Table 4 in Appendix.

As the paper suggests,for evaluation of these metrics NLP application, Information Extraction and Information Retrieval techniques are required in order to make an objective analysis [39].

Second, content quality model includes a set of quality attributes to evaluate the documentation(see in Table 5 in Appendix)

The authors were applying three ERP systems: Openbravo, Compiere and ADempiere to obtain results.

According to [40], suggests that quantitative measures and metrics be integrated with requirements specification language and other automated tools like Ada, ISDOS, PSL, PSA, SREM, SADT, SAMM, etc.

CAME tools - (Computer Assisted Software Measurement and Evaluation) tools are tools for modelling and determining the metrics of software development components referring to the process, the product and the resource. Presently, the CAME tool area also includes the tools for model-based software components analysis, metrics application, presentation of measurement results, statistical analysis and evaluation [41].

ESQUT software quality evaluation tool - which was used by many software development departments in authors' company,can measure C and COBOL source code and detail design documents are expressed by tree-structured charts [42].

Service Oriented Requirements Traceability Tool (SORTT) - authors developed a prototype tool towards process automation. The tool automates the aforementioned translating, indexing, querying, filtering, and rendering activities. It's purpose is to mitigate factors such as the time and effort of extracting trace links [43].

Source Monitor and CCCC tool - to compare the programs written using design patterns with those written without using design patterns [44].

Some of the research studies interpret several clustering analysis techniques to predict software metrics quality like fuzzy c-means, k-means Gaussian mixture model and etc [45] [46] [47] [48].

3.3. What activities are performed in the software development life cycle phases?

The set of activities during the different phases of software development process. However, not many studies have included the complete introduction of activities that can be done during the mentioned phases. Here are some researches containing information about the activities of the indicated phases.

According to [37], the taxonomy of early phases includes phases and activities include User needs analysis, Definition of the Solution Space, System requirements, System Design

In general, the software development process early phases involve Requirements Analysis and Definition, and Design phases [49].

Requirements Analysis and Definition phase involves activities like Feasibility study, requirements elicitation, requirements analysis, requirements validation, and requirements documentation.

Design phase, in which the overall system architecture is established [**?**]. This phase involves several activities like examining the requirements document, choosing the architectural design method, choosing the programming language, verifying, specifying, and documenting design activities.

To conclude, we can argue that the activities to be conducted during the phases of software process depend on the type of quality assessment and evaluation method the company choose. However, the default activities during the general phases are given as the results from the several studies in this section above.

3.4. What metrics are collected during the software development life cycle phases?

SDLC software metrics are usually associated with uncertainty that can be assessed by fuzzy set theory [50]. The authors in [46] suggests that the way to capture uncertainty, vagueness and imprecision in software metrics is a fuzzy set theory. The same concern have the other

studies that define the phases and metrics of software process according to the uncertainty theory are [51] [52] [53].

Victor R. Basili [54] conducted an empirical validation of OO metrics that in the environment of Object oriented analysis and design method. They discussed the relationship between Chidamber and Kemerer's OO metrics and Fault Probablity in high- and low-level design phases of the life-cycle. The result of the research has shown that five out of the six Chidamber and Kemerer's OO metrics was useful to predict class fault-proneness.

Similarly, Kumar et al. [55] suggests a model for the defect detection in the early stages of development life cycle like design and early coding phases can be pre-defined with the Complexity, coupling and cohesion (CCC) related metics.

According to Bharathi et al. [56], the realiability of the system can be decided by mapping of design metrics that depend on the external parameters like operational profile, maturity and the domain of the project.

Also in [37], the primary factors of the user interest in the requirements management phase were defined as functions performed, response time, user interface and reliability. Whereas the primary factors of customer interest include functions performed, development time, cost, maintainability, modifiability, and reliability.

However, [46] argues that predicting the reliability of the software is impossible due to the computational complexity. Consequently, authors selected top reliability relevant metrics for requirements engineering phase - Requirement Defect Density, Requirement Specification Change Request and Requirement Inspection and Walk through; for design phase - Cyclomatic Complexity and Design Review Effectiveness;

Whereas, Phillips et al. [57] offered methodology approach to improve space system resilience with architecture design, system engineering and increased software security by reducing the risk latent software defects and vulnerabilities.

During the SLR, we have collected variety set of metrics, nonetheless, McCabe, Halstead, Cyclomatic Complexity and CK metrics were the most popular software metrics that were mentioned in majority of selected studies [42] [55] [46] [47] [54] [50] [58]. Besides, many studies have been concentrated on metrics derivation based on different viewpoints (see Figure 2):

- Module Complexity [59] [60] [47] [61]
- Module Maintainability [56] [40] [46]
- Module Functionality [50] [55] [54] [38]

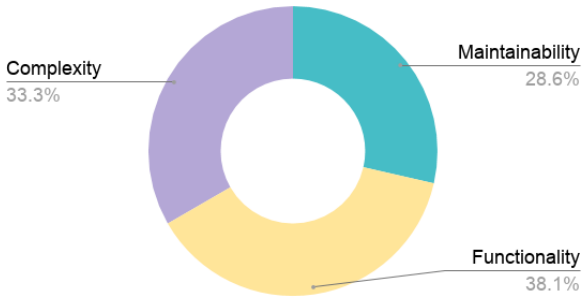


Figure 2: Viewpoints concentrated on metrics

Overall, in the result of SLR of selected studies we have derived metrics for Requirements Management (see in Table 2) and Design phases (see in Table 3).

Metric	Description
Requirement Defect Density	
Requirement Specification Change Request	
Requirement Inspection and Walk through	
Requirements Traceability	what portion of the requirements have been allocated to the software requirements documents, and then translated into the design, coded, and tested
Requirements Definition	The portion of the software requirements are defined satisfactorily (unambigu- ous, testable, and acceptable to the user).
Requirements Stabilty	Change action on the requirements. number of units or modules effected by change requests compared to the total number.
Requirement fault density	Measures the fraction of faulty requirements specification documents.
Review,inspection and walk-through	Ensure that the software requirement specification is feasible, complete, consistent and accurate
Actual Requirement	Total number of initial requirements present in the project.
Volatile Requirements	Total number of requirements change requests (insert/delete/update) made during the project cycle.
Requirement Schedule	Total no of days (estimated) is required to complete the task
Requirement stability	Depends on requirement specification change request
Experience of requirement team	Review, inspection and walkthroughs metric depends on the experience of requirement team
Quality of specification requirement document	The degree to which a software system meets the specified requirements, design and coding

Table 2: Requirements Management phase Metrics

4. Dicussions

As we stated from the beginning, the aim of this SLR is to define and evaluate the methods, their corresponding metrics and the early phases to conduct assessment and evaluation of quality of the software process.

Created research questions to define are the classification of the software development life cycle phases, the existing models/approaches for software quality in early phases, activities that are performed during these phases and metrics accordingly.

Firstly, we defined a general set of phases and helpful metrics that are applicable to the widely used models like Waterfall and Agile methodologies. The limitations of our research paper is that we took the general classification of SDLC phases not classifying them into Agile or Waterfall. Despite the fact that Agile methodology phases are performed in a short period and iteratively as opposed to Waterfall model.

The results of SLR has shown that 75.7% of the studies indicate Requirements Management and Design phase as early phases of software development process.

Second, there exist many methods to assess and evaluate the software quality like: Software tools (CAME tool [3], Source Monitor [4], CCCC, BBN model[5], ESQUIT tool [6]) use of Machine Learning approaches [7] (decision trees, SVM, genetic algorithm Naive Bayes) analysis of particular modules to analyze complexity, functionality and maintainability (Enterprise Resource Planning [8], UML diagrams [9]) Some of the research studies interpret several

Metric	Description
Depth of Inheritance Tree of a class (DIT)	DIT is defined as the maximum depth of the inheritance graph of each class
Number Of Children of a Class(N0C)	The number of direct descendants for each class
Coupling Between Object classes (CB0)	Class is coupled to another one if it uses its member functions and/ or instance variables. CBO provides the number of classes to which a given class is coupled
Response For a Class	This the number of methods that can potentially be executed in response to a message received by an object of that class.
Weighted Methods per Class (WMC)	It measures the complexity of an individual class
Lack of Cohesion on Methods (LC0M)	The number of pairs of member functions without shared instance variables, minus the number of pairs of member functions with shared instance variables
EDGE_COUNT	Number of edges found in a given module control from one module to another
NODE_COUNT	Number of nodes found in a given module
BRANCH_COUNT	Branch count metrics
CALL_PAIRS	Number of calls to other functions in a module
CONDITION_COUNT	Number of conditionals in a given module
CYCLOMATIC_COMPLEXITY	The cyclomatic complexity of a module $v(G) = e - n + 2$
DECISION_COUNT	Number of decision points in a given module
DECISION_DENSITY	$\text{Condition}_count/Decision_count$
DESIGN_COMPLEXITY:iv(G)	The design complexity of a module
DESIGN_DENSITY	Design density is calculated as: $iv(G)/v(G)$
ESSENTIAL_COMPLEXITY:ev(G)	The essential complexity of a module
ESSENTIAL_DENSITY	Essential density is calculated as: $(ev(G)1)/(v(G)1)$
$\text{MAINTENANCE}_SEVERITY$	Maintenance Severity is calculated as: $ev(G)/v(G)$
MODIFIED_CONDITION_COUNT	The effect of a condition affect a decision outcome by varying that condition only
MULTIPLE_CONDITION_COUNT	Number of multiple conditions that exist within a module
PATHOLOGICAL_COMPLEXITY	A measure of the degree to which a module contains extremely unstructured constructs
Software Progress metric	Simple top level measure of progress in completing the design, code, and integration phases
Defect Density	Requires tracking defect reports against the size of the software (in lines of code, non- comment source statements, or functional points)
Size of modules	Size of each module (in LOC non-comment source statements/functional points)
Design review effectiveness	Make sure that the design meets the stakeholder's requirements or to find whether design requires modification
EJ	The number of strong entity types in data model
ES	The number of weak entity types in data model
EM	The number of mixed entity types in data model
V	The number of relationship types
AEJ	The number of attributes for strong entity types
AES	The number of attributes for weak entity types
AEM	The number of attributes for mixed entity types

Metric	Description
Design Review Effectiveness	Effectiveness of Review process in design phase
Software complexity	The cyclomatic complexity, data flow complexity, and system design complexity
Experience of design and development team	Measures the relevant design and development experience, motivation, programmer capability during the design and development phase of SDLC
Review effort (Requirements & Design)	Assessed in terms of the person hours
New functionality implemented	The extent of working on new functionality rather than just enhancing the older functionalities of software

Table 3: Design phase Metrics

clustering analysis techniques to predict software metrics quality like fuzzy c-means, k-means Gaussian mixture model and etc. The method can be chosen at any phase of the software process aiming at evaluation of quality parameters unders interest. Starting with cost evaluation frameworks for fault prediction models in analysis level till formal measurement approaches to evaluate static code and automated tools for assessing maintainability of the software. Third, SDLC software metrics are usually associated with uncertainty that can be assessed by fuzzy set theory [10]. The authors in [11] suggest that the way to capture uncertainty, vagueness and imprecision in software metrics is a fuzzy set theory. There are considerable number of studies dedicated to software quality evaluation, nonetheless, the base metrics that were mentioned in almost all of these researches are almost the same: Chidamber and Kemerer's OO metrics - software metrics tailored to object-oriented software design Cyclomatic complexity - responsible for measurement of structural complexity of the code Lines of Code - emphasizes on calculating numbers of lines in a piece of code. Halstead complexity metric - identifies measurable properties of source code, and the relations between them. To sum up, selected metrics give more insights if the basic measurement is correctly traced and analysed.

5. Conclusion

In conclusion, a successful project acquires the track, control and analysis of the software development process throughout the SDLC. To maintain the constant pace of project development [12] and timing, one needs to measure the software process as early as possible. Generally,the early phases include requirements management and design phases. The metrics can vary depending on the methodology and the goal of the company. The future works in this direction will be the further discussion of Uncertainty level,experiment on the useful and working methods to conduct the software quality evaluation process. As Benjamin Franklin said[13], "The bitterness of poor quality remains long after the sweetness of low price is forgotten". Therefore, one should never delay in ensuring the process quality that will lead to a consequent product.

Acknowledgments

This research project is carried out under the support of the Russian Science Foundation Grant № 19-19-00623.

Appendices

Indicator name	Description
Title, Author, Format	General information regarding the document
Structureness	Structure of the textual documentation: number of chapters, sections, sub-sections, document length, density of tables and figures
Dimension	length of the document sentences: to analyse the length of the document sentences
External references	External references understanding to support the comprehension of software engineers and users.
Graphical Support	verify the availability of visual aids(figures and tables),m their citations, clear captions
Updating or alignment	verify the document update with reference to the project release it refers to
Readability	verify clearness of the sentences, concepts understandability
Completeness	evaluate the documentation with reference to the source code, description of the source code items(packages, classes, methods)
Usability	API and Wiki and assessing the organization from a usability point of view
Comments appropriateness	Estimate the density of comments in the code

Table 4: Structure Quality indicators

Indicator name	Description
Adaptability	evaluate the documentation adequacy referring to the supported operating system and DBMS.
Installability	verify the documentation adequacy of the description of the installation process.
Replaceability	evaluate description existence in the documentation about the possibility of replacing the ERP system with a different one.
Migration	verify the documentation adequacy that describes the migration process referring the generated reports.
Recoverability	evaluate the functionality for the transactions management adequacy
Interoperability	evaluate the documentation discussions on the web-services support
Importability	evaluate the documentation adequacy of describing the importing data formats
Configuration	analysis of ERP system configuration functionality, charts of accounts, supported languages, tax category management
Customization	verify customization functionalities regarding user interface, full system, workflow and reports adequacy

Table 5: Content Quality indicators

[1] B. Kitchenham. Procedures for performing systematic. 2004.
[2] Barbara Kitchenham, O. Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. Systematic literature reviews in software engineering – a systematic literature review. *Information and Software Technology*, 51(1):7–15, January 2009.
[3] Tullio Vernazza, Giampiero Granatella, Giancarlo Succi, Luigi Benedicenti, and Martin Mintchev. Defining Metrics for Software Components. In *Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics*, volume XI, pages 16–23, July 2000.
[4] Marco Ronchetti, Giancarlo Succi, Witold Pedrycz, and Barbara Russo. Early estimation of software size in object-oriented environments a case study in a cmm level 3 software firm. *Information Sciences*, 176(5):475–489, 2006.
[5] Jeremy Kivi, Darlene Haydon, Jason Hayes, Ryan Schneider, and Giancarlo Succi. Extreme programming: a university team design experience. In *2000 Canadian Conference on Electrical and Computer Engineering. Conference Proceedings. Navigating to a New Era (Cat. No.00TH8492)*, volume 2, pages 816–820 vol.2, May 2000.
[6] Frank Maurer, Giancarlo Succi, Harald Holz, Boris Kötting, Sigrid Goldmann, and Barbara Dellen. Software Process Support over the Internet. In *Proceedings of the 21st International Conference on Software Engineering*, ICSE '99, pages 642–645. ACM, May 1999.
[7] Alberto Sillitti, Andrea Janes, Giancarlo Succi, and Tullio Vernazza. Measures for mobile users: an architecture. *Journal of Systems Architecture*, 50(7):393–405, 2004.
[8] Marco Scotto, Alberto Sillitti, Giancarlo Succi, and Tullio Vernazza. A Relational Approach to Software Metrics. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, SAC '04, pages 1536–1540. ACM, 2004.
[9] Marco Scotto, Alberto Sillitti, Giancarlo Succi, and Tullio Vernazza. A non-invasive approach to product metrics collection. *Journal of Systems Architecture*, 52(11):668–675, 2006.
[10] Andrea Janes and Giancarlo Succi. *Lean Software Development in Action.* Springer, Heidelberg, Germany, 2014.
[11] Giuseppe Marino and Giancarlo Succi. Data Structures for Parallel Execution of Functional Languages. In *Proceedings of the Parallel Architectures and Languages Europe, Volume II: Parallel Languages*, PARLE '89, pages 346–356. Springer-Verlag, June 1989.
[12] Andrea Valerio, Giancarlo Succi, and Massimo Fenaroli. Domain analysis and framework-based software development. *SIGAPP Appl. Comput. Rev.*, 5(2):4–15, September 1997.
[13] Alberto Sillitti, Tullio Vernazza, and Giancarlo Succi. Service Oriented Programming: A New Paradigm of Software Reuse. In *Proceedings of the 7th International Conference on Software Reuse*, pages 269–280. Springer Berlin Heidelberg, April 2002.
[14] Justin Clark, Chris Clarke, Stefano De Panfilis, Giampiero Granatella, Paolo Predonzani, Alberto Sillitti, Giancarlo Succi, and Tullio Vernazza. Selecting components in large cots repositories. *Journal of Systems and Software*, 73(2):323–331, 2004.
[15] Luis Corral, Alberto Sillitti, Giancarlo Succi, Alessandro Garibbo, and Paolo Ramella. Evolution of Mobile Software Development from Platform-Specific to Web-Based Multiplatform Paradigm. In *Proceedings of the 10th SIGPLAN Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2011, pages 181–183, New York, NY, USA, 2011. ACM.
[16] Luis Corral, Anton B. Georgiev, Alberto Sillitti, and Giancarlo Succi. Can execution time describe accurately the energy consumption of mobile apps? An experiment in Android. In *Proceedings of the 3rd International Workshop on Green and Sustainable Software*, pages 31–37. ACM, 2014.
[17] Luis Corral, Alberto Sillitti, and Giancarlo Succi. Software Assurance Practices for Mobile Applications. *Computing*, 97(10):1001–1022, October 2015.
[18] Giancarlo Succi, Luigi Benedicenti, and Tullio Vernazza. Analysis of the effects of software reuse on customer satisfaction in an RPG environment. *IEEE Transactions on Software Engineering*, 27(5):473–479, 2001.
[19] Giancarlo Succi, Witold Pedrycz, Michele Marchesi, and Laurie Williams. Preliminary analysis of the effects of pair programming on job satisfaction. In *Proceedings of the 3rd International Conference on Extreme Programming (XP)*, pages 212–215, May 2002.
[20] Petr Musílek, Witold Pedrycz, Nan Sun, and Giancarlo Succi. On the Sensitivity of COCOMO II Software Cost Estimation Model. In *Proceedings of the 8th International Symposium on Software Metrics*, METRICS '02, pages 13–20. IEEE Computer Society, June 2002.
[21] Witold Pedrycz and Giancarlo Succi. Genetic granular classifiers in modeling software quality. *Journal of Systems and Software*, 76(3):277–285, 2005.
[22] Raimund Moser, Witold Pedrycz, and Giancarlo Succi. A Comparative Analysis of the Efficiency of Change Metrics and Static Code Attributes for Defect Prediction. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE 2008, pages 181–190. ACM, 2008.

[23] Raimund Moser, Witold Pedrycz, and Giancarlo Succi. Analysis of the reliability of a subset of change metrics for defect prediction. In *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '08, pages 309–311. ACM, 2008.

[24] Bruno Rossi, Barbara Russo, and Giancarlo Succi. Modelling Failures Occurrences of Open Source Software with Reliability Growth. In *Open Source Software: New Horizons - Proceedings of the 6th International IFIP WG 2.13 Conference on Open Source Systems, OSS 2010*, pages 268–280, Notre Dame, IN, USA, May 2010. Springer, Heidelberg.

[25] Witold Pedrycz, Barbara Russo, and Giancarlo Succi. A model of job satisfaction for collaborative development processes. *Journal of Systems and Software*, 84(5):739–752, 2011.

[26] Witold Pedrycz, Barbara Russo, and Giancarlo Succi. Knowledge Transfer in System Modeling and Its Realization Through an Optimal Allocation of Information Granularity. *Appl. Soft Comput.*, 12(8):1985–1995, August 2012.

[27] Alberto Sillitti, Giancarlo Succi, and Jelena Vlasenko. Understanding the Impact of Pair Programming on Developers Attention: A Case Study on a Large Industrial Experimentation. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 1094–1101, Piscataway, NJ, USA, June 2012. IEEE Press.

[28] Luis Corral, Anton B Georgiev, Alberto Sillitti, and Giancarlo Succi. A method for characterizing energy consumption in Android smartphones. In *Green and Sustainable Software (GREENS 2013), 2nd International Workshop on*, pages 38–45. IEEE, May 2013.

[29] Enrico Di Bella, Alberto Sillitti, and Giancarlo Succi. A multivariate classification of open source developers. *Information Sciences*, 221:72–83, 2013.

[30] Irina D Coman, Pierre N Robillard, Alberto Sillitti, and Giancarlo Succi. Cooperation, collaboration and pair-programming: Field studies on backup behavior. *Journal of Systems and Software*, 91:124–134, 2014.

[31] Giancarlo Succi, James Paulson, and Armin Eberlein. Preliminary results from an empirical study on the growth of open source and commercial software products. In *EDSER-3 Workshop*, pages 14–15, 2001.

[32] György L Kovács, Sylvester Drozdik, Paolo Zuliani, and Giancarlo Succi. Open Source Software for the Public Administration. In *Proceedings of the 6th International Workshop on Computer Science and Information Technologies*, October 2004.

[33] James W Paulson, Giancarlo Succi, and Armin Eberlein. An empirical study of open-source and closed-source software products. *IEEE Transactions on Software Engineering*, 30(4):246–256, 2004.

[34] Etiel Petrinja, Alberto Sillitti, and Giancarlo Succi. Comparing OpenBRR, QSOS, and OMM assessment models. In *Open Source Software: New Horizons - Proceedings of the 6th International IFIP WG 2.13 Conference on Open Source Systems, OSS 2010*, pages 224–238, Notre Dame, IN, USA, May 2010. Springer, Heidelberg.

[35] Brian Fitzgerald, Jay P Kesan, Barbara Russo, Maha Shaikh, and Giancarlo Succi. *Adopting open source software: A practical guide*. The MIT Press, Cambridge, MA, 2011.

[36] Bruno Rossi, Barbara Russo, and Giancarlo Succi. Adoption of free/libre open source software in public organizations: factors of impact. *Information Technology & People*, 25(2):156–187, 2012.

[37] Alan M. Davis. A taxonomy for the early stages of the software development life cycle. *Journal of Systems and Software*, 8(4):297–311, September 1988.

[38] Lerina Aversano, Daniela Guardabascio, and Maria Tortorella. Analysis of the documentation of ERP software projects. *Procedia Computer Science*, 121:423–430, 2017.

[39] L. Aversano, Daniela Guardabascio, and M. Tortorella. Evaluating the quality of the documentation of open source software. In *ENASE*, 2017.

[40] Yosef S. Sherif, Edward Ng, and Jodi Steinbacher. Computer software development: Quality attributes, measurements, and metrics. *Naval Research Logistics*, 35(3):425–436, June 1988.

[41] REINER R. DUMKE and HEIKO GRIGOLEIT. *Software Quality Control*, 6(2):157–169, 1997.

[42] H. Ogasawara, A. Yamada, and M. Kojo. Experiences of software quality management using metrics through the life-cycle. In *Proceedings of IEEE 18th International Conference on Software Engineering*. IEEE Comput. Soc. Press.

[43] Arthur Marques, Franklin Ramalho, and Wilkerson L. Andrade. Towards a requirements traceability process centered on the traceability model. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing - SAC '15*. ACM Press, 2015.

[44] Nosheen Qamar and Ali Afzal Malik. Impact of design patterns on software complexity and size. *April 2020*, 39(2):342–352, April 2020.

[45] Bingbing Yang, Xin Zheng, and Ping Guo. Software metrics data clustering for quality prediction. In *Lecture Notes in Computer Science*, pages 959–964. Springer Berlin Heidelberg, 2006.

[46] H.B. Yadav and D.K. Yadav. Defects prediction of early phases of software development life cycle using fuzzy logic. In *Confluence 2013: The Next Generation Information Technology Summit (4th International*

Conference). Institution of Engineering and Technology, 2013.

[47] Harikesh Bahadur Yadav and Dilip Kumar Yadav. Construction of membership function for software metrics. *Procedia Computer Science*, 46:933–940, 2015.

[48] Bingbing Yang, Qian Yin, Shengyong Xu, and Ping Guo. Software quality prediction using affinity propagation algorithm. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. IEEE, June 2008.

[49] Fred van den Bosch, John R. Ellis, Peter Freeman, Len Johnson, Carma L. McClure, Dick Robinson, Walt Scacchi, Ben Scheff, Arndt von Staa, and Leonard L. Tripp. Evaluation of software development life cycle. *ACM SIGSOFT Software Engineering Notes*, 7(1):45–60, January 1982.

[50] Chandan Kumar and Dilip Kumar Yadav. A method for developing node probability table using qualitative value of software metrics. In *Proceedings of the 2015 Third International Conference on Computer, Communication, Control and Information Technology (C3IT)*. IEEE, February 2015.

[51] Pongtip Aroonvatanaporn, Thanida Hongsongkiat, and B. Boehm. Improving software development tracking and estimation inside the cone of uncertainty. 2012.

[52] Wouter Tengeler. Cone of uncertainty for agile projects, 2014. Online: http://www.themotionstudio.nl/en/cone-of-uncertainty-for-agile-projects/, on 5^{th} October 2020.

[53] Stefan Luyten. The cone of uncertainty and how to avoid it turning into a wormhole, 2014. Online: https://medium.com/@stefanluyten/the-cone-of-uncertainty-82d21e99fcc2, on 5^{th} October 2020.

[54] V.R. Basili, L.C. Briand, and W.L. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, 22(10):751–761, 1996.

[55] Prathipati Ratna Kumar and G.P Saradhi Varma. A novel probabilistic-ABC based boosting model for software defect detection. In *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*. IEEE, March 2017.

[56] R. Bharathi and R. Selvarani. A framework for the estimation of oo software reliability using design complexity metrics. In *2015 International Conference on Trends in Automation, Communications and Computing Technology (I-TACT-15)*, pages 1–7, 2015.

[57] Dewanne M. Phillips, Thomas A. Mazzuchi, and Shahram Sarkani. An architecture, system engineering, and acquisition approach for space system software resiliency. *Information and Software Technology*, 94:150–164, February 2018.

[58] R Bharathi and R. Selvarani. A framework for the estimation of OO software reliability using design complexity metrics. In *2015 International Conference on Trends in Automation, Communications and Computing Technology (I-TACT-15)*. IEEE, December 2015.

[59] Narimane Zighed, Nora Bounour, and Abdelhak-Djamel Seriai. Comparative analysis of object-oriented software maintainability prediction models. *Foundations of Computing and Decision Sciences*, 43(4):359–374, December 2018.

[60] Yue Jiang, Bojan Cuki, Tim Menzies, and Nick Bartlow. Comparing design and code metrics for software quality prediction. In *Proceedings of the 4th international workshop on Predictor models in software engineering - PROMISE '08*. ACM Press, 2008.

[61] Sun-Jen Huang and Richard Lai. Deriving complexity information from a formal communication protocol specification. *Software: Practice and Experience*, 28(14):1465–1491, December 1998.