The Impact of Software Development Process on Software Quality: A Review

Conference Paper · December 2016

DOI: 10.1109/CICN.2016.137

2 authors, including:

Brijendra Singh
University of Lucknow
30 PUBLICATIONS **214** CITATIONS

SEE PROFILE

The Impact of Software Development Process on Software Quality: A Review

Brijendra Singh
ICT Research Lab, Department of Computer Science
University of Lucknow, Lucknow, 226007, India
drbri_singh@hotmail.com

Shikha Gautam
ICT Research Lab, Department of Computer Science
University of Lucknow, Lucknow, 226007, India
shikhagautam90@gmail.com

***Abstract*–Quality of software products depends upon various phase of software development process. Process of software development is used to create and achieve quality in software products. Software development process uses four main phases which have its own importance for development. Software quality is a conformance to requirements which is divided into functional and non-functional requirements. The objective of this paper is to present a review on the impact of software development process on software quality. In this paper various quality attributes have been considered during analysis of development process to see the impact on software quality. During analysis we observe that software architecture is more important phase than other phase because it provides abstract representation of overall structure of software.**

Keywords– Software development process, software quality, software requirement, software design, software coding/implementation, software testing.

I. INTRODUCTION

Software process can be defined as "a set of activities, methods, practices, and transformations that people use to develop and maintain software and the associated products" [1]. According to IEEE software development process is a process by which user needs are translated into a software product. The process involves translating user needs into software requirements, transforming the software requirements into design, implementing the design in code, testing the code, and sometimes, installing and checking out the software for operational use [2]. The main objective of the development of a system is its efficient integration in real-life situations. Various software development methods have been adopted to develop the software products such as: waterfall model, iterative and incremental model, spiral model, V model, rapid application development, prototyping model, agile model, and hybrid spiral model [3-6]. Some of the most commonly used are waterfall, spiral, V model, and agile model. Software development organizations have realized that adherence to a suitable well defined life cycle model helps to produce good quality products [7-8].

Mainly there are four phase of software development; software requirement, software design, software coding/implementation and software testing, which have been used in various models. Each and every phase have an individual impact on software quality attributes. These phase play an important role to improve quality in finished products [9-10]. A suitable life cycle model can possible be selected based on an analysis of issues such as: characteristics of the software to be developed, characteristics of development team, and characteristics of customer.

There are various issues in traditional software development process. The failure of many software projects in terms of not meeting user/business requirements, prone to errors etc has led to software quality becoming one of the key issues from all stakeholders' perspective [11]. In a competitive environment quality based product is basic need for any product success. To achieve quality, efficient process is required.

The objective of this paper is to present a review on the impact of software development process on software quality. Research objective of this review paper is to analyze the impact of software development process on software quality.

Software requirement analysis used to collect needs or requirement of software. Requirement analysis is the first step which involve to the quality because this step used to capture all functional and non-functional requirements to be implemented in final product. Software design is next step to derive quality to create complete structure or architecture of software which is stated into requirement specification. Design provides not only to find out how the software product is going to be appear, but also allows both software users and developers to realize how it's going to function. Because design is the only way to completely translate a requirements into a finished product.

After software design software coding/implementation phase is used for implementing the software. Software implementation is based upon programming language. This phase also play an important role because using coding an executable version of software is created. Programming language can impact not only the coding process, but also the properties of the resulting product and its quality. Software testing is conducted when executable software exists. Testing used to find out errors and fix them to support software quality. Testing check what all functions software supposed to do & also check that software is not doing what he not supposed to do.

In this paper we have organized the activities in software development process and analyze the impact of individual phase on various well defined attributes of quality. As we have seen every phase has an individual impact on software quality attributes. Section II describes the software development process. Section III describes impact of software requirement on software quality. Section IV describes impact of software design on software quality. Section V describes impact of software coding/implementation on software quality. Section VI describes impact of software testing on software quality. Section VII provides analysis and conclusion. In Section VII various quality attributes have been considered during

analysis of development process to see the impact on software quality.

II. SOFTWARE DEVELOPMENT PROCESS

Software process is an organized set of activities required to develop a software product. Software development is the process of taking a set of requirements from a user, analyzing them, designing a solution to the problem, and then implementing that solution on a computer [12]. The international standard for describing the method of selecting, implementing and monitoring the life cycle for software is ISO/IEC 12207 [13]. There are many approaches to software development, known as software development life cycle models, methodologies, or processes. The waterfall model is a traditional version, and agile software development is a newest version for development. There are many different software processes but every process includes four activities: Requirement, Design, Coding, and Testing. After that maintenance is required for further changes [14].

Presently software organization shifting its focus from product issues to process issues [15]. Software quality is important concern for software industry and organization [16]. A general process model for software encompasses a set of framework and umbrella activities, actions, and work tasks. Process can be used to solve common problems that are occurred as part of the software process. Every model provides a different process flow, but all perform the same set of activities: communication, planning, modeling, construction, and deployment [17]. Sequential process models, such as waterfall [18] and V models [19] are linear process models. This is applicable in situations where requirements are well defined and stable [20].

Incremental process models are iterative in nature and develop working versions of software simply [21]. Evolutionary process models use the iterative, incremental nature to implement software product. Evolutionary models, such as prototyping [22] and spiral model [23], produce incremental work products quickly. These process models can be used from development to long-term system maintenance. Agile [24] is a new model for software development which uses iterative/incremental development, less documentation, lightweight and fewer process controls. It was targeted at small to medium-size software projects and smaller teams of developers and develops complete software quickly [25].

Special models include the component-based model [26] that incorporate component reuse and assembly; the formal methods model that encompasses a mathematical based approach to software development; and the aspect-oriented model [27] which uses crosscutting concerns spanning for system architecture. The Unified Process [28] is a “use case driven, architecture-centric, iterative and incremental” software process designed for UML [29] methods and tools. Personal [30] and team [31] models for the software process have been developed. Both provide measurement, planning, and self-direction as key ingredients for a successful software process [4]. Quality of process affects quality of product. Process is important because quality is derived by well defined process. Selection of most appropriate process model is important concern to achieve quality [32].

III. IMPACT OF SOFTWARE REQUIREMENT ON SOFTWARE QUALITY

Software requirements are defined during the early stages of a software development as a specification of what should be implemented. They are descriptions of how the system should behave, or of a system property or attribute. [33]. IEEE defines requirement analysis is a process of studying user needs to arrive at a definition of system, hardware, or software requirements [2].

Requirements analysis is important to the success of a systems or software project [34]. The requirements should be documented, actionable, measurable, testable, traceable, related to identified business needs and defined to a level of detail sufficient for system design. Software requirements include business requirements, user requirements, system requirements, external interface requirement, functional requirements, and non-functional requirements. In requirement statements every individual business, user, functional, and nonfunctional requirement would exhibit the qualities. Characteristics of requirement statements are complete, correct, feasible, necessary, prioritized, unambiguous, and verifiable. But it’s not enough to have excellent individual requirement statements. So requirements collections are used to collect a set of requirement or group of requirement [35].

Requirements should state what to do and the design should describe how it does this. Characteristics of requirements are complete, consistent, modifiable, and traceable which reflect the software quality. Quality is heavily dependent on functional or non-functional requirements. In this section we have analyzed the various parameters of requirement i.e. consistency, completeness or correctness and modifiable.

Consistency means providing predictable, maintainable and reliable results to the customer. Consistency refers to situations where a specification contains no internal contradictions, whereas completeness refers to situations where a specification entails everything that is known to be “true” in a certain context. It contains all necessary information to avoid ambiguity and need no amplification to enable proper implementation and verification. Correctness is usually meant to be the combination of consistency and completeness. Correctness is often more pragmatically defined as satisfaction of certain business goals [36].

Modifiable refers to each requirement be uniquely labeled and expressed separately from others so you can refer to it unambiguously. If its structure and style are changes to the requirement can be made easily, completely & consistently [35]. Table I depicts the relationship between the software requirement characteristics and software quality attributes.

TABLE I
Relationship between the Software Requirement Characteristics and Software Quality Attributes

Requirement Characteristics	Quality Attributes
Consistency [36, 37, 40, 41]	traceability, tailorability, interoperability, usability, scalability, and reliability
Completeness or Correctness [38, 40]	maintainability, functionality, and reliability
Modifiable [39, 42]	performance

Using consistent requirement traceability, tailorability, interoperability, usability, scalability, and reliability improve [36, 37, 40, 41]. Using completeness or correctness of requirement maintainability, functionality, and reliability enhance [38, 40]. Modification in requirement engineering (RE) process contributes to improved business performance [39, 42]. Based on Table I the impact of software requirement on various attributes of software quality has been shown in Figure 1.

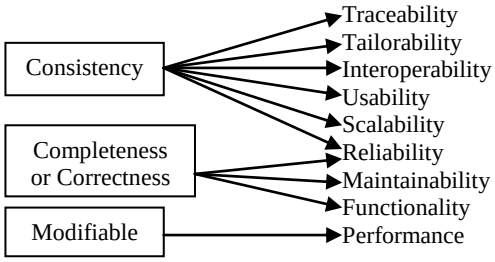

Figure 1 Impact of software requirement on software quality

Figure 1 also shows that all quality attribute individually dependent on requirement characteristics but reliability dependent on all three Cs of requirement consistency, completeness and correctness.

IV. IMPACT OF SOFTWARE DESIGN ON SOFTWARE QUALITY

The importance of software design can be stated with a single word—Quality. Design is the place where quality is achieved. Design provides representations of software that can be assessed for quality. Design is the only way to accurately translate a customer's requirements into a finished software product [4]. Software design is the process by which an agent creates a specification of a software artifact, intended to accomplish goals, using a set of primitive components and subject to constraints [43].

IEEE defines design is a process of defining the architecture, components, interfaces, and other characteristics of a system or component [2]. Software design usually involves problem solving and planning a software solution. This includes both low-level design and high-level design. Fundamental concept of software design has evolved to modularity, software architecture, design patterns and software component design. Table II depicts the relationship between the software design attributes and software quality attributes.

The design process is concerned with describing how a requirement is to be met by the design product [44]; there are various design attribute but software architecture, pattern, abstraction and modularity play an important role. Quality captured as nonfunctional requirements in the early steps of software development, influence greatly the software systems architecture. Software architecture is a coordination tool among the different phases of software development. Architectural design decisions directly impact software quality [45].

Software architecture provides abstract representation of overall structure of software [46]. Software architecture improves functionality, security, efficiency, portability, interoperability, reusability, maintainability, reliability, usability, performance, evolvability, and adaptability [47-53]. Software architecture includes modularity; software is divided into separately named and addressable components, called modules that are integrated to satisfy problem requirements. Modularity enhances changeability and adaptability [54-55].

TABLE II
Relationship between the Software Design Attributes and Software Quality Attributes

Design Attributes	Quality Attributes
Software Architecture [47, 48, 49, 50, 51, 52, 53]	adaptability, functionality, security, efficiency, portability, interoperability, reusability, maintainability, reliability, usability, performance, and evolvability
Modularity [54, 55]	changeability and adaptability
Design Pattern [56, 57, 58, 59]	reusability, composability, completeness, maintainability, stability, understandability, flexibility, simplicity, and expandability
Software Component Design [60, 61]	accountability, stability, and modularity

Design pattern is a reusable solution of problem occurring in software design. Design pattern is not a finished design that directly transformed into source code. It is a description or template for how to solve a problem that can be used in many different situations. Using design pattern reusability, composability, completeness, maintainability, stability, understandability, flexibility, simplicity, and expandability are improved [56-59]. Software component design also affects the software quality attributes because that transforms structural elements of the software architecture into a procedural description of software components and improves accountability, stability, and modularity [60-61]. Based on Table II the impact of software design on various attributes of software quality has been shown in Figure 2.

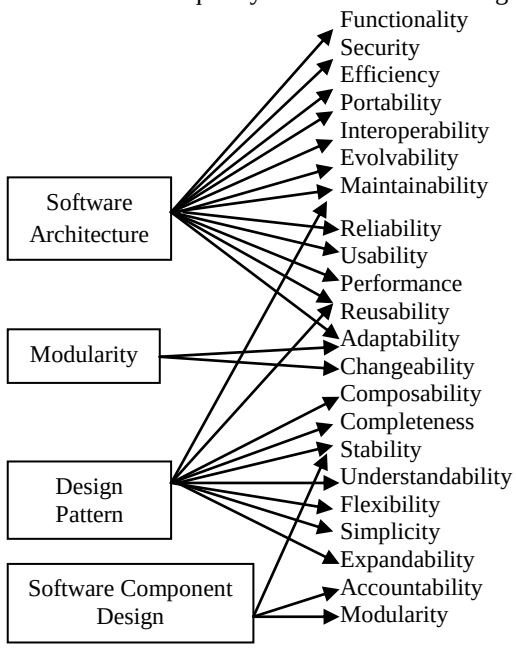

Figure 2 Impact of software design on software quality

Figure 2 shows that all attributes individually dependent on design attributes except adaptability, maintainability, reusability, and stability. Maintainability and reusability depends upon software architecture and design pattern. Adaptability depends upon software architecture and modularity. Design pattern and software component design reflects the stability.

V. IMPACT OF SOFTWARE CODING/IMPLEMENTATION ON SOFTWARE QUALITY

Implementation/coding may involve developing programs in high-level or low-level programming languages or tailoring and adapting generic, off-the-shelf systems to meet the specific requirements of an organization. Programming used to create an executable program. IEEE defines coding is a process of expressing a computer program in a programming language [2].

There are various characteristics of different languages that affect software quality. The choice of language for a particular application is a difficult. The selection affects result and quality of software [62]. Most software codes are written by distant teams. Developers may frequently join or leave software companies. It's very important for source codes to be understandable so it can be easily adapted [63]. In a traditional methodology heavy weight coding is involve there is a need to adopt a light weight codling policy.

Software quality is a broadly characteristic of the software life cycle, defined as what software product should be and what it must contain. The quality of software coding is defined by several characteristics: maintainability, traceability, availability, reliability, reusability, testability, and readability. There are various factors which affect the quality, few are: simple and standardized code, programming features, flexible, traceable and readable code. Table III depicts the relationship between the software coding factors and software quality attributes.

TABLE III
Relationship between the Software Coding Factors and Software Quality Attributes

Coding Factors	Quality Attributes
Programming Features [63, 64, 65]	readability and maintainability
Standardized Code [66, 67]	maintainability and stability
Code Clones [68, 69]	reliability and maintainability

Software coding affects various quality attributes using programming features, standardized code and code clone. Programming Features provides overall characteristics of programming code which influences the code readability and maintainability. Code readability is usually connected with comments and naming standards [63-65]. Standardized Code provides desired product and enhances the comparability of results. Using standardized code maintainability and stability achieves. The technical quality of source code is an important feature for software maintainability [66-67].

Code clone (duplicated code) provides reusability of code and save effort by copying a pre-existing program section. Code clone affects the maintainability and reliability [68-69]. Based on Table III the impact of software coding/implementation on various attributes of software quality has been shown in Figure 3.

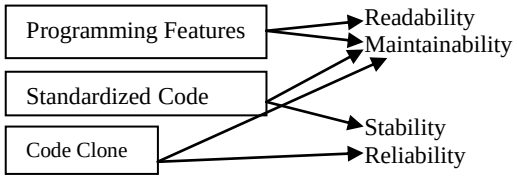

Figure 3 Impact of software coding on software quality

Figure 3 shows that every attribute individually depend on coding factor but maintainability depend all three factor of coding.

VI. IMPACT OF SOFTWARE TESTING ON SOFTWARE QUALITY

Software testing is a process of executing a program or application with the intent of finding the errors or quality of the product or service under test. Software testing can be conducted as soon as possible when executable software exists. The overall approach to software development often determines when and how testing is performed. IEEE defines testing is a process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component [2].

There are many types of software testing. Reviews, walkthroughs, and inspections are referred to as static testing, whereas actually executing programmed code with a given set of test cases is referred to as dynamic testing. Software testing methods are divided into white-box and black-box testing. There are four levels of tests: unit testing, integration testing, component interface testing, and system testing. Software testing is more difficult because of the vast array of programming languages, operating systems, and hardware platforms that have evolved [70].

Testing is an important factor for software quality. Testing phase support quality by executing software code and find out errors. In testing various important parameters which improve software quality such as: test criteria, coverage, testing effort and software testability. Table IV depicts the relationship between the software testing parameters and software quality attributes.

TABLE IV
Relationship between the Software Testing Factors and Software Quality Attributes

Testing Parameters	Quality Attributes
Test Criteria [71, 72, 73]	completeness, effectiveness, and maintainability
Testing Effort [74, 75]	user friendliness, portability, reliability, and maintainability
Software Testability [76, 77]	reliability, correctness, performance, and effectiveness
Coverage [78]	reliability

Test criteria focus on measuring the coverage of the test suite upon the structural elements of the program. To test code quality system testing is involve. Code coverage is the most frequently used metric for test code quality assessment and there exist many tools for dynamic code coverage estimation. Test criteria improve completeness, effectiveness, and maintainability [71-73]. To test effort is required; testing effort is collection of run test case and fixes the errors. Testing effort used to fix error, using that user friendliness, portability, reliability, and maintainability enhance [74-75].

Software testability is the tendency of code to reveal existing faults during random testing. This is an analysis for measuring the value provided by a particular software testing scheme, where the value of a scheme can be assessed in different ways. Software testability improves reliability, correctness, performance, and effectiveness [76-77].

Coverage testing helps the tester create a thorough set of tests and gives a measure of test completeness. Coverage analysis of programs during testing not only gives a clear measure of testing quality but also reveals important aspects of software structure. Structure of a program can be a significant component in confident assessment of overall software quality and reliable product [78]. Based on Table IV the impact of software testing on various attributes of software quality has been shown in Figure 4.

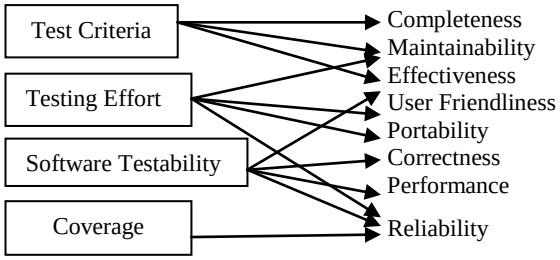

Figure 4 Impact of software testing on software quality

Every attribute individually depend on software testing except maintainability, effectiveness, and reliability as shown in Figure 4. Maintainability depends on test criteria and testing effort. Effectiveness depends on test criteria and software testability. Reliability depends on software testability, testing effort and coverage.

VII. ANALYSIS AND CONCLUSION

Purpose of analysis is to see the impact of software development process on software quality. The impact of software development process consisting four phase of development process on software quality is shown in Figure 5.

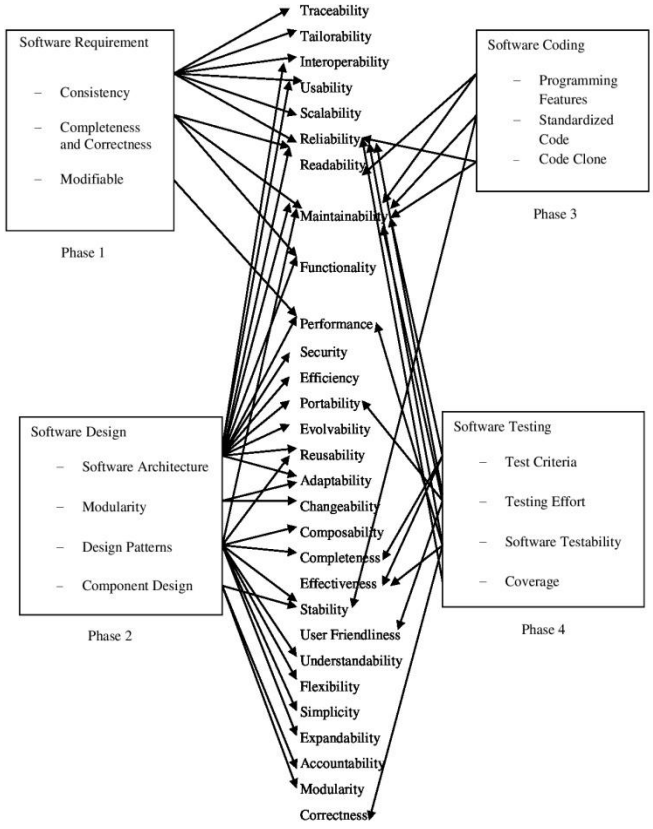

Figure 5 Impact of software development process on software quality

Based on analysis every phase of software development is important for software quality however software design is more important than other phase. Software design is used to capture the set of requirements and create architecture of software which realizes the broad class of requirements.

Software architecture provides abstract representation of overall structure of software and strategic design for creating or structuring software using the general principles of software design.

The impact of software architecture on software quality is shown in Figure 6. To achieve quality in finished product accurate software architecture is required.

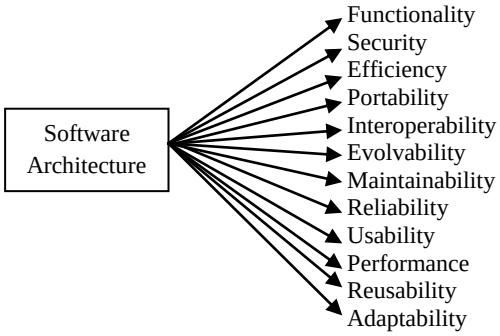

Figure 6 Impact of software architecture on software quality

In literature we have seen that every quality attributes are important but maintainability is most important attribute for software product because software maintenance is the process of refining the software to make sure it continuous to meet business needs. The importance of maintainability for software development process is shown in Figure 7.

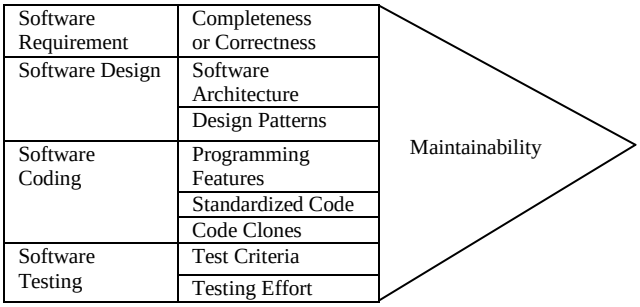

Figure 7 Importance of maintainability for software development process

Using maintainability software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment. So maintainability is important for software development process. The main finding of this analysis is software architecture play an important role to achieve quality in software products.

ACKNOWLEDGMENT

Ms. Shikha Gautam is thankful to UGC for providing UGC-RGNF-JRF fellowship to sustain her research.

REFERENCES

[1] M. C. Paulk, C. Weber, B. Curtis, and M. B. Chrissis, The Capability Maturity Model: Guidelines for Improving the Software Process, Addison Wesley, 1994.
[2] J. Radatz, A. Geraci, and F. Katki, IEEE standard glossary of software engineering terminology, IEEE Std. 610.12-1990, 1990.
[3] K. K. Aggarwal and Y. Singh, Software Engineering, New Age International Publishers, 2007.

[4] R. S. Pressman, Software engineering: a practitioner's approach, McGraw-Hill, 2010.
[5] B. Singh and S. P. Kannojia, "A model for software product quality prediction," Journal of Software Engineering and Applications, Vol. 5, pp. 395-401, May 2012. doi: 10.4236/jsea.2012.56046.
[6] B. Singh and S. Gautam, "Hybrid Spiral Model to Improve Software Quality Using Knowledge Management," International Journal of Performability Engineering, Vol. 12, Issue 4, pp. 341-352, July 2016.
[7] P. Isaias and T. Issa, High level models and methodologies for information systems, Springer, 2015.
[8] P. Sujatha, G. V. Sankar, A. S. Rao, and T. Satyanarayana, "The Role of Software Verification and Validation in Software Development Process," IETE Technical Review, Vol. 18, Issue 1, pp. 23-26, Jan 2001. doi: 10.1080/02564602.2001.11416938.
[9] N. S. Godbole, Software quality assurance: Principles and practice, Alpha Science Int'l Ltd., 2004.
[10] B. Singh, Quality Control and Reliability Analysis, Khanna Publishers, 2011.
[11] L. P. W. Land and J. Higgs, "An Empirical Study of Software Quality Improvement Practices from Multiple Perspectives-An Australian Case Study," In Proceedings of the 11th Pacific-Asia Conference on Information Systems (PACIS), pp. 547-560, Jan 2007.
[12] J. Dooley, Software development and professional practice, Apress, 2011.
[13] IEEE Standard 12207-2008, Systems and Software Engineering-Software life cycle processes, IEEE, 2008.
[14] J. Münch, O. Armbrust, M. Kowalczyk, and M. Sotó, Software process definition and management, Springer, 2012.
[15] M. J. Davis, "Process and Product: Dichotomy or Duality?," ACM SIGSOFT Software Engineering Notes, Vol. 20, Issue 2, Apr 1995. doi: 10.1145/224155.565634.
[16] G. Cugola and C. Ghezzi, "Software Processes: a Retrospective and a Path to the Future," Software Process: Improvement and Practice, Vol. 4, Issue 3, pp. 101-123, Sep 1998. doi: 10.1.1.17.2499.
[17] W. Scacchi, Models of software evolution: life cycle and process. Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 1987.
[18] W. W. Royce, "Managing the development of large software systems," In Proceedings of IEEE WESCON, Vol. 26, Issue 8, pp. 328-338, Aug 1970.
[19] C. Bucanac, The V-Model. University of Karlskrona/Ronneby, 1999.
[20] B. Boehm, "Software Engineering," IEEE Trans. on Computer, Vol. C-25, Issue 12, pp. 1226-1241, 1976. doi: 10.1109/TC.1976.1674590.
[21] J. McDermid and P. Rook, Software development process models, Software Engineer's Reference Book, 1991.
[22] P. Jalote, A concise introduction to software engineering, Springer, 2008.
[23] B. Boehm, "A spiral model of software development and enhancement," IEEE Computer, Vol. 21, Issue 5, pp. 61-72, May 1988. doi: 10.1109/2.59.
[24] A. Cockburn, Agile software development: the cooperative game, Pearson Education, 2006.
[25] M. Cohn, Succeeding with agile: software development using Scrum. Pearson Education, 2010.
[26] A. W. Brown and K. C. Wallnan, "Engineering of component-based systems," In Second IEEE International Conference on Engineering of Complex Computer Systems, pp. 414-422, Oct 1996. doi: 10.1109/ICECCS.1996.558485.
[27] S. Clarke and E. Baniassad, Aspect-oriented analysis and design, Addison-Wesley Professional, 2005.
[28] J. Arlow and I. Neustadt, UML 2 and the unified process: practical object-oriented analysis and design, Pearson Education, 2005.
[29] G. Booch, J. Rumbaugh, and I. Jacobsen, The unified modeling language user guide, Pearson Education India, 2005.
[30] W. S. Humphrey, "The Personal Software Process-Status and Trends," IEEE Software, Vol. 17, Issue 6, pp. 71-75, Nov 2000.
[31] W. S. Humphrey, Introduction to the team software process, Addison-Wesley Professional, 2000.
[32] C. Ghezzi, M. Jazayeri, and D. Mandrioli, Fundamentals of software engineering, Prentice Hall PTR, 2002.
[33] I. Sommerville and P. Sawyer, Requirements engineering: a good practice guide, John Wiley & Sons, Inc. 1997.
[34] T. ur Rehman, M. N. Khan, and N. Riaz, "Analysis of requirement engineering processes, tools/techniques and methodologies," International Journal of Information Technology and Computer Science (IJITCS), Vol. 5, Issue 3, pp. 40-48, Feb 2013. doi: 10.5815/ijitcs.2013.03.05.
[35] K. Wiegers and J. Beatty, Software requirements, Pearson Education, 2013.
[36] D. Zowghi and V. Gervasi, "The Three Cs of requirements: consistency, completeness, and correctness," In International Workshop on Requirements Engineering: Foundations for Software Quality, pp. 155-164, Sep 2002.
[37] A. Cimatti, M. Roveri, A. Susi, and S. Tonetta, "Validation of requirements for hybrid systems: A formal approach," ACM Transactions on Software Engineering and Methodology (TOSEM), Vol. 21, Issue 4, 22, Nov 2012. doi: 10.1145/2377656.2377659.
[38] D. Sinnig, P. Chalin, and F. Khendek, "Use case and task models: an integrated development methodology and its formal foundation," ACM Transactions on Software Engineering and Methodology (TOSEM), Vol. 22, Issue 3, 27, Jul 2013. doi: 10.1145/2491509.2491521.
[39] I. Sommerville and J. Ransom, "An empirical study of industrial requirements engineering process assessment and improvement," ACM Transactions on Software Engineering and Methodology (TOSEM), Vol. 14, Issue 1, pp. 85-117, Jan 2005.
[40] I. Sommerville, Integrated requirements engineering: A tutorial. IEEE software, Vol. 22, Issue 1, pp. 16-23, Jan 2005.
[41] M. Alshamari and P. Mayhew, "Technical review: Current issues of usability testing," IETE Technical Review, Vol. 26, Issue 6, pp. 402-406, Nov 2009. doi: 10.4103/0256-4602.57825.
[42] H. H. Liu, Software performance and scalability: a quantitative approach, John Wiley & Sons, 2011.
[43] P. Ralph and Y. Wand, "A Proposal for a Formal Definition of the Design Concept," Annual Review of Policy Design, Vol. 1, Issue 1, pp. 1-35, Sep 2013.
[44] D. Budgen, Software design. Pearson Education, 2003.
[45] I. Mistrík, R. Bahsoon, P. Eeles, R. Roshandel, and M. Stal, editors. Relating System Quality and Software Architecture, Morgan Kaufmann, 2014.
[46] H. Zhu, Software design methodology: From principles to architectural styles, Elsevier, 2005.
[47] F. Losavio, L. Chirinos, N. Lévy, and A. Ramdane-Cherif, "Quality characteristics for software architecture," Journal of Object Technology, Vol. 2, Issue 2, pp. 133-150, Mar 2003.
[48] N. Juristo, A. M. Moreno, and M. I. Sanchez-Segura, "Analysing the impact of usability on software design," Journal of Systems and Software, Vol. 80, Issue 9, pp. 1506-1516, Sep 2007. doi: 10.1016/j.jss.2007.01.006.
[49] A. Martens, H. Koziolek, S. Becker, and R. Reussner, "Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms," In Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering, ACM, pp. 105-116, Jan 2010.
[50] H. P. Breivold, I. Crnkovic, and M. Larsson, "A systematic review of software architecture evolution research," Information and Software Technology, Vol. 54, Issue 1, pp. 16-40, Jan 2012. doi: 10.1016/j.infsof.2011.06.002.
[51] B. J. Williams and J. C. Carver, "Characterizing software architecture changes: A systematic review," Information and Software Technology, Vol. 52, Issue 1, pp. 31-51, Jan 2010. doi: 10.1016/j.infsof.2009.07.002.
[52] T. Karthikeyan and J. Geetha, "A Study and Critical Survey on Service Reusability Metrics," International Journal of Information Technology and Computer Science (IJITCS), Vol. 4, Issue 5, pp. 25-31, May 2012. doi: 10.5815/ijitcs.2012.05.04.
[53] S. Saxena and S. K. Dubey, "Impact of Software Design Aspects on Usability," International Journal of Computer Applications, Vol. 61, Issue 22, pp. 48-53, Jan 2013.
[54] K. J. Sullivan, W. G. Griswold, Y. Cai, and B. Hallen, "The structure and value of modularity in software design," In ACM SIGSOFT Software Engineering Notes, Vol. 26, Issue 5, pp. 99-108, Sep 2001.
[55] J. W. Banks, Modularity and adaptability. Lawrence Livermore National Laboratory (LLNL), Livermore, Apr 2013.
[56] H. Zhu and I. Bayley, "On the Composability of Design Patterns," IEEE Transactions on Software Engineering, Vol. 41, Issue 11, pp. 1138-1152, Nov 2015. doi: 10.1109/TSE.2015.2445341.
[57] A. Ampatzoglou, A. Chatzigeorgiou, S. Charalampidou, and P. Avgeriou, "The effect of GoF design patterns on stability: a case study," IEEE Transactions on Software Engineering, Vol. 41, Issue 8, pp. 781-802, Aug 2015. doi: 10.1109/TSE.2015.2414917.
[58] A. Ampatzoglou, G. Frantzeskou, and I. Stamelos, "A methodology to assess the impact of design patterns on software quality," Information and Software Technology, Vol. 54, Issue 4, pp. 331-46, Apr 2012. doi: 10.1016/j.infsof.2011.10.006.

[59] F. Khomh and Y. G. Gueheneuce, “Do design patterns impact software quality positively?,” In 12th European Conference on Software Maintenance and Reengineering (CSMR), IEEE, pp. 274-278, Apr 2008.
[60] W. Benghabrit, H. Grall, J. C. Royer, and M. Sellami, “Accountability for abstract component design,” In 40th EUROMICRO Conference on Software Engineering and Advanced Applications, IEEE, pp. 213-220, Aug 2014.
[61] L. P. Tizzei, M. Dias, C. M. Rubira, A. Garcia, and J. Lee, “Components meet aspects: Assessing design stability of a software product line,” Information and Software Technology, Vol. 53, Issue 2, pp. 121-36, Feb 2011. doi: 10.1016/j.infsof.2010.08.007.
[62] B. Singh and S. P. Kannojia, “Languages and Their Importance in Quality Software,” In International Conference on Communication Systems and Network Technologies (CSNT), IEEE, pp. 956-959, May 2012. doi: 10.1109/CSNT.2012.203.
[63] Y. Tashtoush, Z. Odat, I. Alsmadi, and M. Yatim, “Impact of programming features on code readability,” International Journal of Software Engineering and Its Applications, Vol. 7, Issue 6, pp. 441-458, 2013. doi: 10.14257/ijseia.2013.7.6.38.
[64] R. P. Buse and W. R. Weimer, “A metric for software readability,” In Proceedings of the 2008 international symposium on Software testing and analysis, ACM, pp. 121-130, Jul 2008.
[65] R. P. Buse and W. R. Weimer, “Learning a metric for code readability,” IEEE Transactions on Software Engineering, Vol. 36, Issue 4, pp. 546-558, Jul 2010.
[66] R. Baggen, J. P. Correia, K. Schill, and J. Visser, “Standardized code quality benchmarking for improving software maintainability,” Software Quality Journal, Vol. 20, Issue 2, pp. 287-307, Jun 2012. doi: 10.1007/s11219-011-9144-9.
[67] D. E. Peercy, “A software maintainability evaluation methodology,” IEEE Transactions on Software Engineering, Vol. SE-7, Issue 4, pp. 343-351, Jul 1981.
[68] A. Monden, D. Nakae, T. Kamiya, S. I. Sato, and K. I. Matsumoto, “Software quality analysis by code clones in industrial legacy software,” In proceedings of Eighth IEEE Symposium on Software Metrics, IEEE, pp. 87-94, 2002.
[69] C. K. Roy, J. R. Cordy, and R. Koschke, “Comparison and evaluation of code clone detection techniques and tools: A qualitative approach,” Science of Computer Programming, Vol. 74, Issue 7, pp. 470-495, May 2009. doi: 10.1016/j.scico.2009.02.007.
[70] G. J. Myers, C. Sandler, and T. Badgett, The art of software testing, John Wiley & Sons, 2011.
[71] D. Athanasiou, A. Nugroho, J. Visser, and A. Zaidman, “Test code quality and its relation to issue handling performance,” IEEE Transactions on Software Engineering, Vol. 40, Issue 11, pp. 1100-1125, Nov 2014. doi: 10.1109/TSE.2014.2342227.
[72] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, “Experiments of the effectiveness of dataflow-and controlflow-based test adequacy criteria,” In proceedings of the 16th international conference on Software engineering, IEEE, pp. 191-200, May 1994.
[73] K. Kapoor and J. Bowen, “Experimental evaluation of the variation in effectiveness for DC, FPC and MC/DC test criteria,” In proceedings of International Symposium on Empirical Software Engineering (ISESE), IEEE, pp. 185-194, Sep 2003.
[74] T. J. Yu and H. E. Dunsmore, The Analysis of Software Development and Testing Processes: An Empirical Study, Technical Report Purdue University, 1985.
[75] C. Y. Huang, J. H. Lo, S. Y. Kuo, and M. R. Lyu, ‘Software reliability modeling and cost estimation incorporating testing-effort and efficiency,” In proceedings of 10th International Symposium on Software Reliability Engineering, IEEE, pp. 62-72, 1999.
[76] J. M. Voas and K. W. Miller, “Improving the software development process using testability research,” In proceedings of Third International Symposium on Software Reliability Engineering, IEEE, pp. 114-121, Oct 1992.
[77] A. P. Mathur, “Performance, effectiveness, and reliability issues in software testing,” In Proceedings of the Fifteenth Annual International Conference on Computer Software and Applications (COMPSAC'91), IEEE, pp. 604-605, Sep 1991.
[78]J. R. Horgan, S. London, and M. R. Lyu, “Achieving software quality with testing coverage measures,” IEEE Computer, Vol. 27, Issue 9, pp. 60-69, Sep 1994.