Fall 2021

Agile Literature Review

Kevin Cochran Jr.

Agile Literature Review
Senior Honors Project

Kevin Cochran Jr.

BUS 452H
Project Management (Honors)
Dr. Gary Mitchell
20 December 2021

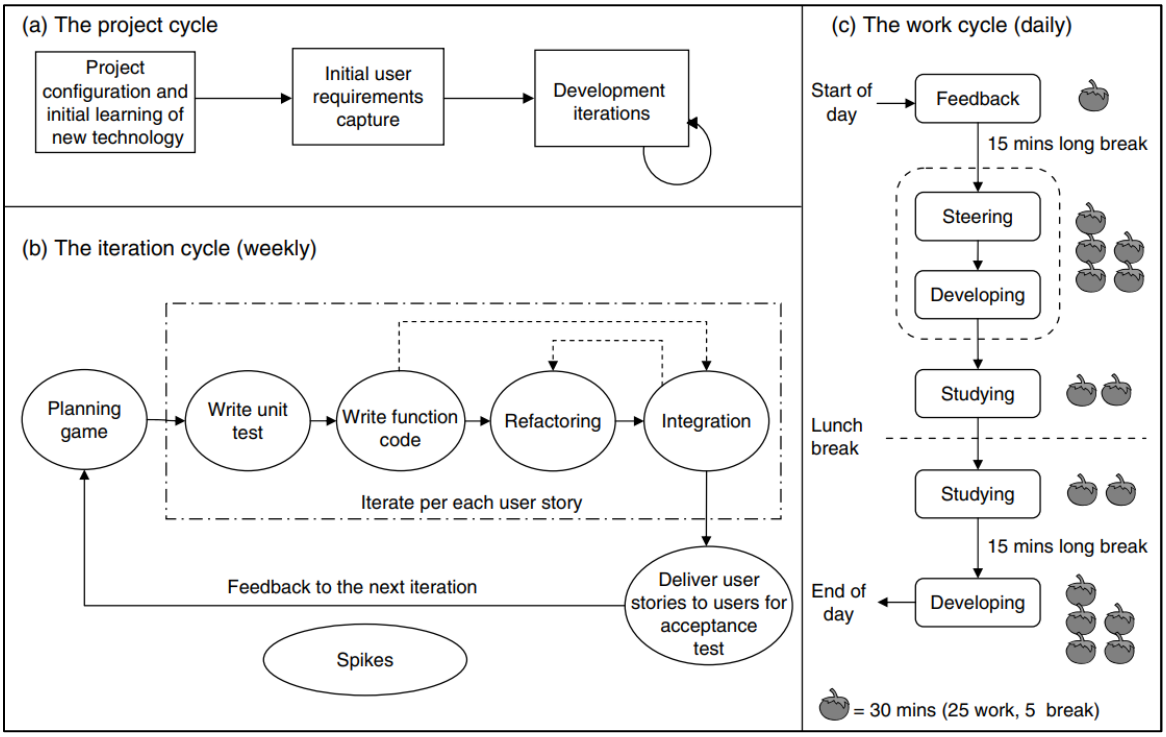
Abstract

Background: Over the last 20 years the software development community has implemented agile techniques over the traditional approach to software development. Agile methods require less upfront costs and increase project flexibility; however, agile methodology is not infallible. Objective: This research seeks to validate the assumption that there is a lack of robust research regarding agile project management and its use in the software development industry. This extensive review of existing literature on the topic will serve as a basis for new research on areas with existing ambiguity. Method: The search engines used to identify relevant literature from 1987 to 2021 on the topic were Business Source Premier and Google Scholar. The procedure used to narrow the search queries was the use of deliberate keywords and phrases such as "agile software development" and "cost of requirement errors". All results were cross-referenced on both search engines to validate the accuracy of each source. Results: 76 papers containing relevant information to agile project management within the software community have been identified: 55 academic journals, 1 book, 1 conference paper, 1 magazine article, 7 periodicals, 10 professional journals, and 1 textbook. 35 papers are critical of Agile methodology, 16 focus mostly on its strengths, 12 focus mainly on its weaknesses, and 13 contain relevant information regarding the cost of requirement errors.

Introduction and Literature Review

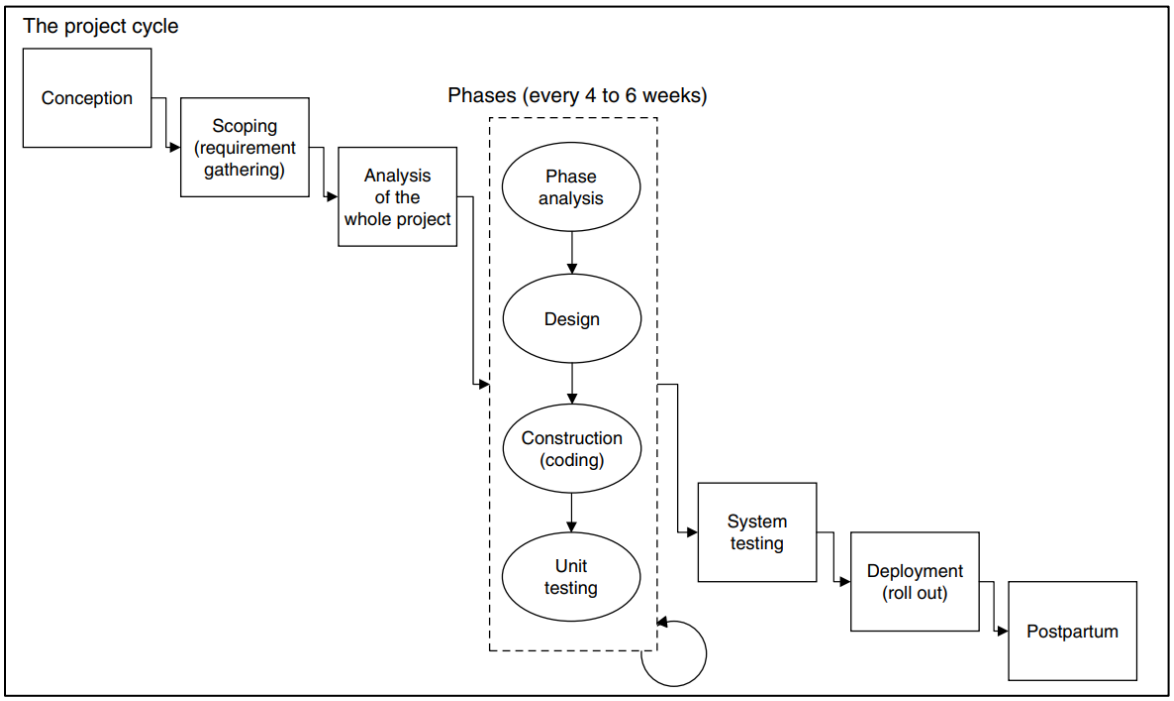
As the dependence on quality software increases with an increasing demand from global infrastructure, the reliance on a precise yet efficient software development process also increases (Luong, 2019). There are several methods of developing software. The two most common are the waterfall-based development cycle, the more traditional approach which defines costs and requirements clearly upfront, and agile software development, a more expeditious path to achieving a goal with partially defined preconditions (Vigden, 2009). Since many organizations think of agile differently, Richard Vigden devised a framework to define "enablers and inhibitors of agility and the emergent capabilities of agile teams" (Vigden, 2009). This served as the groundwork for identifying the advantages and disadvantages for using agile software development over the waterfall approach. Figure 1 is a diagram of the first team observed by Vigden who used an agile methodology while Figure 2 represents the second team that used the waterfall approach. The layout of the Figure 2 reveals the accurate terminology used to describe the software approach since it cascades like a waterfall.

Figure 1 – Development Life Cycle of the Pongo Team



Note. Adapted from Vigden and Wang 2006. A flow following the dotted arrow may happen, but less often.

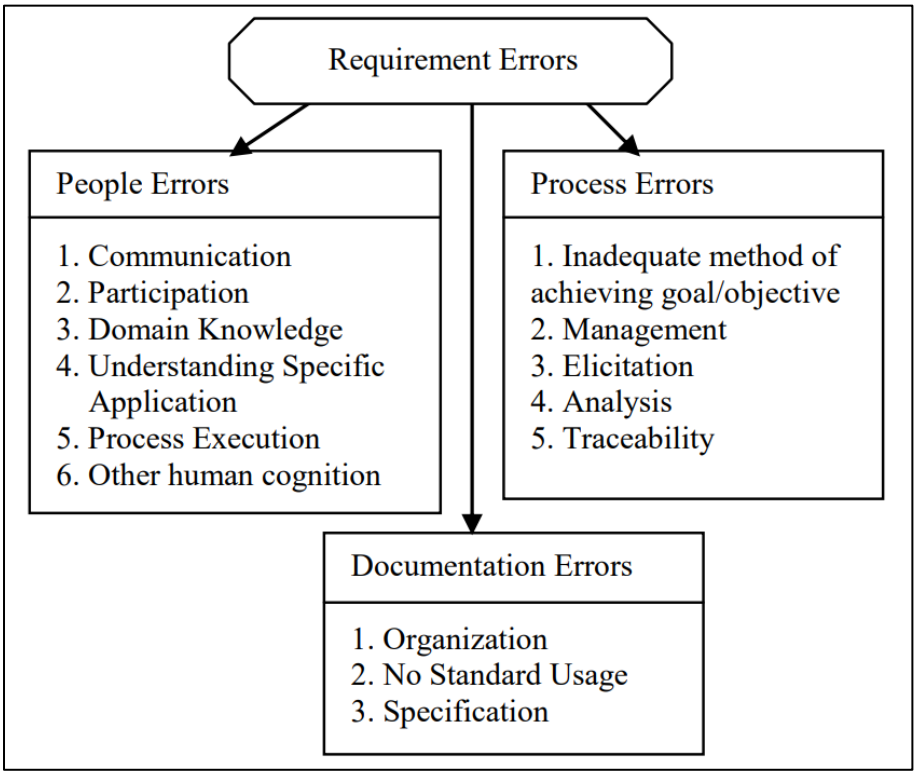
Figure 2 – Development Life Cycle of the SysCheck Team



Note. Adapted from Wang and Vigden 2007.

A critical journal that serves as a foundation for future research on agile software development is from Barry Boehm and Philip Papaccio titled “Understanding and Controlling Software Costs”. Boehm found that “rework typically consumes 30% - 50% of a typical project budget” (Boehm and Papaccio, 1988). This quantification of the impact rework has on project costs plays a large role in influencing the decision organizations must execute when beginning a new software development cycle. If there is a large margin for error when defining requirements at the beginning of a software development project, it may be reasonable to use the waterfall method and clearly define the project scope to reduce future costs of rework. Dean Leffingwell was able to conclude in his book *Managing Software Requirements: A Unified Approach* that “requirements errors can easily consume 25% - 40% of the total project budget” (Leffingwell and Widrig 2000). This builds upon Boehm’s research and further supports the assertion that a high probability of requirements errors will likely increase project costs. Gursimran Walia defined a way of identifying requirements errors using the Error Abstraction Process (EAP) and Requirement Error Taxonomy (RET). The EAP was able to increase productivity while the RET was “useful for improving software quality” (Walia, 2006). Figure 3 presents an accurate organization of requirement errors that arise during a software development life cycle. This RET assists software developers as a catalog of potential errors they may produce within their projects. The first step to solving any problem is by clearly identifying its existence and defining it precisely.

Figure 3 – Requirement Error Classification



By 2015, Pawan Chaurasia recognized the vast number of tools and methods established by researchers to enhance the quality of software development by identifying the possible errors which occur during the project life cycle. First, Chaurasia detailed the stages that lead to failure and focused his paper on the cause of that failure which he refers to as "faults" as shown in Figure 4 (Chaurasia, 2015). He went a step further than Walia and defined an extensive list of all potential errors within a project and a clear description of each as shown in Figure 5 (Chaurasia, 2015). For researchers and developers seeking ways to correct their errors, Chaurasia provides a reference to the academic journal which addresses each possible error. His reference numbers can be found below Figure 5. All articles cited within this review are located in the Agile Literature Directory.

Figure 4 – Failure Life Cycle

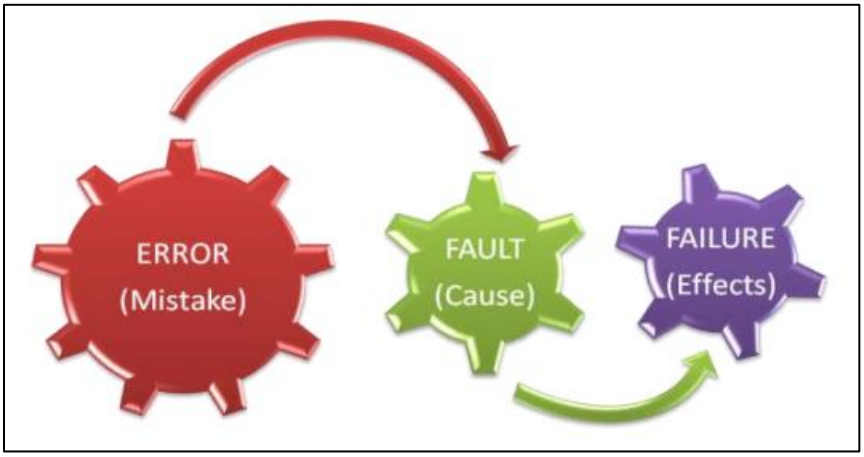


Figure 5 – Requirement Error Taxonomy

SN	Types of Errors	Description	Reference
1	Communication Errors	Insufficient project communication	6
		Requirement editing is not communicated	7
		Lack of communication between developers and users	24
		Poor communication between developers team	11
		Poor communication between development process	25
		Lack of communication information not reach between peoples	26
2	Participation errors	Involvement of users at requirement level	5
		Participate only selected users	27
		Do not involve all the neutrals	7
3	Domain knowledge errors	Lack of domain knowledge	10, 28
		Complexity of problem	11, 12
		Lack of appropriate proper knowledge and information	29
		Lack of proper training	6
		Misunderstanding due to complexity	28
4	Specific application errors	Knowledge of hardware and software specification	31
		Knowledge of input, output and process mappings	32

		Errors in expected output	15
		Requirements are interpreted or predict while solving conflict problems	9
		Knowledge of software interface module	30
5	Process execution errors	Errors in sequence of execution or requirement process	33, 34
		Storage problem, sequence order of stages and missing stages	34, 28
6	Human knowledge errors	Lack of situation awareness problem	14. 26
		Environmental conditions	25
7	Inadequate method	Incomplete knowledge for achieving goals	28
		Errors in achieving goals	28
		Selection of wrong method	36
		Transcription error	8
8	Management errors	Poor management of people & resources	29
		Lack of leadership	13
9	Specification errors	Missing conditions	10
		Errors while documenting requirements	36
10	Organizational requirement errors	Poor organization of requirement	6
		Errors in organizing requirements	22
11	Requirement analysis errors	Selection of incorrect model	35
		Misuse of error solution process	24
		Unsolved issues and problems	10
		Errors while analyzing requirement use cases	24, 37, 38
12	Requirement simulation errors	Inadequate requirement gathering process	10
		Lack of information for source of resources	29

[1] Nikora, A. P.; Lyu, M. R. Software reliability measurement experience. Handbook of Software Reliability Engineering, Lyu, M. R. Ed.;McGraw-Hill: New York, 1996; 255–301.

[2] P. Fusaro, F. Lanubile, G. Visaggio, A replicated experiment to assess requirements inspection techniques, Journal of Empirical Software Engineering 2(1) (1997) 39– 57.

[3] F. Lanubile, F. Shull, V.R. Basili, Experimenting with error abstraction in requirements documents, in: Proceedings of Fifth International Software Metrics Symposium, METRICS'98, IEEE Computer Society, Bethesda, MD, 1998, pp. 114–121.

[4] S. Basu, N. Ebrahimi, Estimating the number of undetected errors: Bayesian model selection, in: Proceedings of the Ninth International Symposium on Software Reliability Engineering, IEEE Computer Society, Paderborn, Germany,1998, pp. 22–31.

[5] D.N. Card, Learning from our mistakes with defect causal analysis, IEEE Software 15 (1) (1998) 56–63.

[6] R.B. Grady, Software failure analysis for high-return process improvement, Hewlett–Packard Journal 47 (4) (1996) 15–24.

[7] J. Jacobs, J.V. Moll, P. Krause, R. Kusters, J. Trienekens, A. Brombacher, Exploring defect causes in products developed by virtual teams, Journal of Information and Software Technology 47 (6) (2005) 399–410.

[8] R.G. Mays, C.L. Jones, G.J. Holloway, D.P. Studinski, Experiences with defect prevention, IBM Systems Journal 29 (1) (1990) 4–32.

[9] T. Nakashima, M. Oyama, H. Hisada, N. Ishii, Analysis of software bug causes and its prevention, Journal of Information and Software Technology 41 (15) (1999) 1059–1068.

[10] Bhandari, M. Halliday, E. Tarver, D. Brown, J. Chaar, R. Chillarege, A case study of software process improvement during development, IEEE

Transactions on Software Engineering 19 (12) (1993) 1157–1170.

[11] S. Beecham, T. Hall, C. Britton, M. Cottee, A. Rainer, Using an expert panel to validate a requirements process improvement model, The Journal of Systems and Software 76 (3) (2005) 251–275.

[12] G.J. Browne, V. Ramesh, Improving information requirements determination: a cognitive perspective, Journal of Information and Management 39 (8) (2002) 625–645.

[13] C. Debou, A.K. Combelles, Linking software process improvement to business strategies: experiences from industry, Journal of Software Process: Improvement and Practice 5 (1) (2000) 55–64.

[14] M.R. Endsley, Situation awareness and human error: designing to support human performance, in: Proceedings of the High Consequence Systems Surety Conference, SA Technologies, Albuquerque, NM, 1999, pp. 2–9.

[15] D.A. Norman, Design rules based on analyses of human error, Communications of the ACM 26 (4) (1983) 254– 258

[16] D.A. Norman, Steps towards a cognitive engineering: design rules based on analyses of human error, Communications of the ACM 26 (4) (1981) 254–258.

[17] C. Trevor, S. Jim, C. Judith, K. Brain, Human Error in Software generation Process, University of Technology, Loughborough, England, 1994.

[18] Endres, An analysis of errors and their causes in system programs, IEEE Transactions on Software Engineering 1 (2) (1975) 140–149.

[19] T.E. Bell, T.A. Thayer, Software requirements: are they really a problem?, in: Proceedings of Second International Conference on Software Engineering, IEEE Computer Society Press, Los Alamitos, CA, 1976, pp 61–68.

[20] T. Berling, T. Thelin, A case study of reading techniques in a software company, in: Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE'04), IEEE Computer Society, 2004, pp. 229–238.

[21] B. Freimut, C. Denger, M. Ketterer, An industrial case study of implementing and validating defect classification for process improvement and quality management, in: Proceedings of the 11th IEEE International Software Metrics Symposium, IEEE Press, 2005.

[22] P.C. Cacciabue, A methodology of human factors analysis for systems engineering: theory and applications, IEEE Transactions on System, Man and Cybernetics – Part A: Systems and Humans 27 (3) (1997) 325–329.

[23] Endres, An analysis of errors and their causes in system programs, IEEE Transactions on Software Engineering 1 (2) (1975) 140–149.

[24] S.H. Kan, V.R. Basili, L.N. Shapiro, Software quality: an overview from the perspective of total quality management, IBM Systems Journal 33 (1) (1994) 4–19.

[25] D. Batra, Cognitive complexity in Data modelling: causes and recommendations, Requirements Engineering Journal 12 (4) (2007) 231–244. [26] K. Sasao, J. Reason, Team errors: definition and taxonomy, Journal of Reliability Engineering and System Safety 65 (1) (1999) 1–9.

[27] D.A. Gaitros, Common errors in large software development projects, The Journal of Defense Software Engineering 12 (6) (2004) 21–25.

[28] J. Galliers, S. Minocha, A. Sutcliffe, A causal model of human error for safety critical user interface design, ACM Transactions on Computer–Human Interaction 5 (3) (1998) 756–769.

[29] J. Coughlan, D.R. Macredie, Effective communication in requirements elicitation: a comparison of methodologies, Requirements Engineering Journal 7 (2) (2002) 47–60.

[30] P. K Chaurasia, Software Reliability Chain Model, International Journal of Software and Web Services (IJSWS), Vol 1, Issue 8, pp 46-50.

[31] R.R. Lutz, Analyzing software requirements errors in safety-critical, embedded systems, in: Proceedings of the IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, San Diego, CA, USA, 1993, pp. 126–133.

[32] S. Sakthivel, A survey of requirement verification techniques, Journal of Information Technology 6 (2) (1991) 68–79.

[33] B. Cheng, R. Jeffrey, Comparing inspection strategies for software requirement inspections, in: Proceedings of the 1996 Australian Software Engineering Conference, IEEE Computer Society, Melbourne, Australia, 1996, pp. 203–211.

[34] P.M. Fitts, R.E. Jones, Analysis of factors contributing to 460 ‘pilot error’ experiences in operating aircrafts control, in: Proceedings of Selected Papers on Human Factors in the Design and Use of Control Systems, Dover Publications Inc., New York, 1961, pp. 332–358.

[35] A.J. Ko, B.A. Myers, Development and evaluation of a model of programming errors, in: Proceedings of IEEE Symposium on Human Centric Computing Languages and Environments, IEEE Computer Society, 2003, pp. 7–14.

[36] Swain, H. Guttman, Handbook of Human Reliability Analysis with Emphasis on Nuclear Power Plant Applications, Nuclear Regulatory Commission, Washington, DC, 1983.

[37] P. K Chaurasia, How Accountability Improves Software Reliability?, International Journal of Computer Science and Technology (IJCSET), Vol 5, Issue 9, pp 868-871.

[38] S.T. Shorrock, B. Kirwan, Development and application of a human error identification tool for air traffic control, Journal of Applied Ergonomics 33 (4) (2002) 319–336.