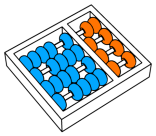

On the relationship between software architecture and delivery capability

B. B. N. de França *P. S. M. dos Santos* *S. Matalonga*

Technical Report - IC-20-07 - Relatório Técnico
May - 2020 - Maio

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

On the relationship between software architecture and delivery capability

Breno Bernard Nicolau de França[1], Paulo Sérgio Medeiros dos Santos[2], Santiago Matalonga[3]

[1]Instituto de Computação Universidade Estadual de Campinas (UNICAMP), Caixa Postal 6176
13083-970 Campinas-SP, Brasil
breno@ic.unicamp.br

[2]Centro de Ciências Exatas e Tecnologia, Universidade Federal do Estado do Rio de Janeiro
Rio de Janeiro-RJ, Brasil
pasemes@uniriotec.br

[3]School of Computing Engineering and Physical Science, The University of the West of Scotland, Scotland
Santiago.Matalonga@uws.ac.uk

Abstract: As the adoption of continuous delivery practices increases in software organizations, different contexts struggle to make it scale for their products in a long-term evolution scenario. Several studies point out the software architecture as a relevant factor for successfully achieving continuous delivery goals. This technical report presents the research protocol for a systematic literature review to explore the relationship between the software architecture and delivery capability. From goals to analysis, we describe the rationale and planned procedures to investigate this matter.

Keywords: Software Architecture, Delivery Capability, Continuous Delivery.

1. Introduction

1.1. Context

Organizations developing software-intensive solutions, involving innovation or meaningful time-to-market constraints, need to constantly adapt to untimely changes so that they can achieve business goals in face of a dynamic and competitive market. In this scenario, organizations need to adopt software practices assuring that new features can be quickly available as quality products or services in an end-to-end perspective.

Hence, the software industry has shown an increasing interest and adoption for practices establishing a more continuous flow for the software lifecycle, taking into account the need for change on how software has been conceived, developed and maintained over the last years, especially in the context of uncertainty and technological innovation. This new thinking regarding the software lifecycle has been named Continuous Software Engineering (CSE) (Fitzgerald and Stöl, 2015) (Figure 1), since it includes practices requiring a more constant pacing to be performed.

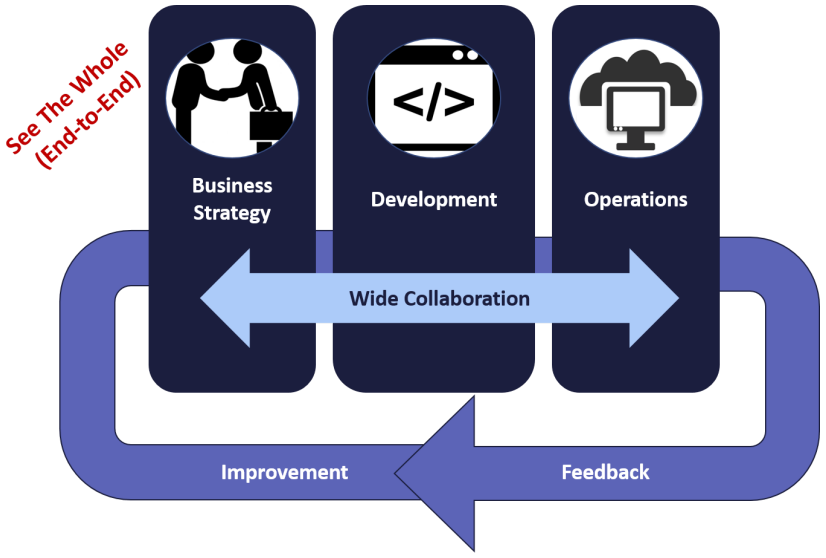

Figure 1. Continuous Software Engineering [adapted from (Fitzgerald and Stöl, 2015)]

Firstly, agile development practices as short iterations fostered rapid releases, making strong use of automation, mainly in software build and testing activities. Such automation usually happens in the context of the Continuous Integration (CI) practice (Beck and Gamma, 2000). Lately, with the perceived agility by the use of CI, an extension was proposed with an additional step: the automated deployment. This step is responsible for, after successfully executing tests, make the application available in a functional state and configured in another environment, staging or production. This extension is the so-called Continuous Delivery or Continuous Deployment (CD) (Humble and Farley, 2010).

Rapid release practices decrease the software development cycle, delivering it as fast as possible and triggering the continuous evolution. Unlike plan-driven approaches, projects adopting these practices start evolutive maintenance just after the first release, usually

conceived as an MVP (Minimum Viable Product), consisting of core functionalities. Therefore, the transition from development to maintenance stage became blurred.

In this context, methods and practices supporting software development and operation, along with supporting tools, are proposed and released in the market (by industry or academia) aiming at making the achievement of goals easier, particularly w.r.t. time to deliver software products and services (Bosch, 2014). However, several technologies have neither proven their effectiveness or efficiency in providing the intended or claimed benefits nor known limitations associated with their continuous use. Additionally, there is not enough evidence to support decision making in the organizations on the adequacy of such technologies to their contexts, causing the failure of initiatives to introduce them with the goal of improving processes with respect to delivery velocity, changeability, and the improvement of the product or service quality. Some challenges associated with the introduction of such practices include:

- Increase effort on testing activities, including test automation, to keep up with constants changes (Marschall, 2007) (Olsson *et al*, 2013) (Mäntylä *et al*, 2015) (Claps *et al*, 2015) (Leppänen *et al*, 2015) (Rodríguez *et al*, 2017);
- Increase of the software failure rate for not having enough time to plan and execute efficient tests (Marschall, 2007) (Claps *et al*, 2015);
- **Increase of the architectural decay during continuous software evolution (Riaz *et al*, 2009);**
- **Growth of Technical Debt (TD) due to reduced time for planning the implementation of change requests (Mäntylä *et al*, 2015);**
- The domain in which a company operates (telecommunications, embedded systems, games…) may impose restrictions on rapid release practices (Leppänen *et al*, 2015) (Rodríguez *et al*, 2017);
- Resistance by the software teams to changes required when adopting rapid release processes (Leppänen et al, 2015) (Rodríguez *et al*, 2017);
- Customers unwillingness to deal with frequent releases (Mäntylä *et al*, 2015) (Claps *et al*, 2015) (Rodríguez *et al*, 2017).

Additionally, Rodríguez et al (2017) identified the following ten groups of factors contributing to the adoption of continuous deployment: fast and frequent release, **flexible product design and architecture**, continuous testing and quality assurance, automation, configuration management, customer involvement, continuous and rapid experimentation, post-deployment activities, agile and lean, and organizational factors.

Particularly, in this study we are concerned with issues associated with software architecture, including its evolution, decay and technical debt, on the use of rapid release practices, which may impact directly on the delivery capability.

Delivery and deployment capability are represented by the minimum frequencies (time) at which software teams can deliver or deploy software artifacts (Mäkinen et al, 2016). In this sense, two frequency metrics can be used: (1) the *actual releasable software cycle,*

which represents the cycle a development team takes to produce an artifact that could be released, but factors out of control of the team prevent such release; and (2) the *actual release cycle* means how often the software artifact is actually released. We are particularly interested in the first given perspective .

A possible explanation for the relationship between internal quality (including architecture) decay and delivery capability relies on the possibility of time pressure over the development team to deliver fast and frequently. And, to achieve this goal, the software team may sacrifice the internal quality of the product (Codabux and Williams, 2013) (Martini et al, 2014) (Alves et al, 2016). Untimely decisions may incur a specific type of TD (architectural debt), which may result in architectural decay on the medium and long-term.

1.2. Problem

The literature on Continuous Delivery and Deployment recognize flexible product design and architecture as an important factor (Rodríguez et al, 2017) (Laukkanen et al, 2017) for its successful adoption. However, no further detail or strategies are given to characterize architectural solutions for CD. Additionally, scientific evidence reveals the incidence of internal quality decay caused by the adoption of rapid releases in the context of open source and industrial software development (Torkar et al, 2011) (Codabux and Williams, 2013). In addition, Codabux and Williams (2013) show that architectural decay is one of the most difficult problems associated with internal quality to handle. This way, the main problem to be investigated in this study regards how the software architecture, including its continuous evolution using rapid releases practices like CI/CD, affects the delivery capability.

1.3. Research Goals

The main goal of this study is to build a theoretical model by capturing and synthesizing evidence from the literature to describe how (i.e., causal mechanisms) delivery capability is affected by elements of the software architecture in the context of continuous delivery and deployment practices.

The delivery capability is associated with the frequency at which the software team makes a new feature available in the operating environment to the potential users. This research is concerned with several aspects of delivery capability:

- The quality of the delivery, in terms of defects identified “in the wild”;
- The cycle time for features to be delivered to the operational environments (the time between a feature is identified and deployed);
- The lead time between for features to be delivered to the operational environments (the time between a feature is elicited and deployed);
- Delivery Frequency between consecutive deliveries to the operational environment.

This research will identify the elements in the software development lifecycle that have influence on delivery capability (as measured by the above attributes). These elements might come in the form of architectural styles, patterns, the quality of the concrete architecture, and other internal quality elements.

A theoretical model capturing the mechanism by how factors associated with continuous evolution influence on the delivery capability allows foreseeing semi-quantitatively (trends instead of absolute values) at which evolution step the architecture starts to decay significantly. This is relevant since a negative impact on the delivery capability slows down product releases and increases maintenance costs.

Finally, a systematic review synthesizing scientific evidence from the literature can drive the building of this theoretical model as sparse evidence is available, but there is no work explaining how this behavior occurs in the context of continuous delivery and deployment. Causal mechanisms, i.e., constructs (factors, mediators and moderators, and response variables) along with propositions, can be identified in field study reports.

2. Research Questions

RQ1: Which architectural characteristics contribute to the delivery capability?

RQ1.1: **What** are the **variables** concerning the quality of the concrete architecture (factors, mediators and moderators) influencing the delivery capability?

RQ1.2 **How** does the quality of the concrete architecture impact the delivery capability?

RQ1.3: **How** do these **variables** relate among each other to contribute to the delivery capability?

RQ1.4: How does the effect of the identified variables **evolve over time**?

RQ2: Does developers' confidence on automated tests, recommended for CI/CD practices, restrict the architecture and design quality?

3. Search Process

3.1. Sources

We are considering automated and manual search strategies. For digital libraries, the following are selected as they include the most relevant venues on continuous software engineering (agile and lean software development) and software architecture (decay and microservices):

- Scopus
- IEEEXplore
- Web of Science

We are excluding non-scientific results as the main goal is to synthesize scientific evidence for developing a theoretical model. Manual search will be conducted by snowballing included papers after the full reading.

3.2. Control Papers

Shahin, Mojtaba et al. An empirical study of architecting for continuous delivery and deployment. Empirical Software Engineering, p. 1-48, 2018.

Bellomo, S., Nord, R. L., & Ozkaya, I. (2013, May). A study of enabling factors for rapid fielding: combined practices to balance speed and stability. In Proceedings of the 2013 International Conference on Software Engineering (pp. 982-991). IEEE Press.

Bosch, J., & Eklund, U. (2012, October). Eternal embedded software: Towards innovation experiment systems. In International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (pp. 19-31). Springer, Berlin, Heidelberg.

3.3. Search Strategy

The search strategy for this research involves the use of two search mechanisms Automatic Search and Snowballing. Automatic search will be executed first. The results from the automatic search process will be subjected through the steps of this SLR (selection and data extraction), and the resulting papers will be used as seed to the snowballing procedure. This way, forward and backward snowballing will be executed. Papers from the snowballing search mechanisms will be evaluated using the same criteria as was used in the Automatic Search.

For the automatic search, one reviewer will apply the search string on the selected engines. And, for the snowballing, the collection of papers will be divided among the three reviewers.

3.4. Search String for Automatic search engines.

Involved concepts: continuous delivery and software architecture.

("continuous* evol*" OR "continuous architecting" OR "continuous software evolution" OR "continuous delivery" OR "continuous deployment" OR "continuous integration" OR "continuous software engineering" OR "rapid releases" OR "rapid fielding") AND ("software architecture" OR "software design" OR "architectural design")

3.5. Considerations for the snowballing search process

The guidelines proposed by Wholin (2014) for conducting snowballing in systematic literature studies will be followed in the research. In particular, we draw attention to the following decision taken:

- Tools:
    - Backward snowballing data will be based on Title from the reference section of the selected papers
    - Forward snowballing data will be taken from Scopus, IEEE, and Web of Science (duplicates will be deleted)
- Seeds
    - Seeds for the snowballing process will be the resulting papers (after data extraction) from the automatic search mechanism

- Review of consistency
  - Before initiating the snowballing process, the Seeds set will be checked against the criteria detailed in the guidelines (diversity of community of practice, the size of the seed set, diversity of authors and publishers). This might result in divergens between the snowballing seeds and the output from the automatic search process (See section Revision of seeds).

3.6. Inclusion/Exclusion Criteria

We are adopting the following criteria.

Inclusion Criteria:

I1: Discuss continuous delivery.

I2: Discuss software system's architecture within the continuous delivery in terms of its variables (factors, mediators and moderators).

I3: Present an empirical/experimental study or experience report with observations in the context of continuous delivery.

Exclusion Criteria:

E1: Gray literature.

E2: Papers not written in english.

E3: The same study reported twice, the most complete report will be considered.

E4: Paper presenting no empirical/experimental evidence (only proposal).

E5: Papers discussing build system's architecture, including testing harness.

3.7. Selection Procedure

Three researchers will conduct this review (R1-R3). Reviewers will select the returned papers from the sources based on the pre-defined criteria. Every selection should be reviewed by one of the other reviewers. This way, every selected paper should be included after two reviewers agree upon the inclusion.

To guarantee the coherence of the researchers criteria, ten papers will be randomly selected and distributed to R1-R3, who will then classify them as Include, Uncertain, Exclude independently and the set will be revised by the research team until the rationale for each classification is reached.

Finally, the whole set of papers will be divided in thirds. Each researcher will review two thirds of the papers. Selection process will be guided by the following table [adapted from Petersen et al (2015)]. Papers classified as F will be excluded. Papers classified as C and D will be read in full to justify inclusion/exclusion.

		Reviewer Y		
		Include	Uncertain	Exclude
Reviewer X	Include	A	B	C
	Uncertain	B	C	D
	Exclude	C	D	F

4. Information Extraction

The information to be extracted from the selected papers is presented in Table 1.

Table 1. Information extraction form

Field	**Description**	**RQ**
Bibliographic information	*Title, authors, affiliations, abstract, publication venue and date, ...*	*N/A*
Variables and quality metrics influencing delivery capability	*Architecture-related factors, moderators and mediators impacting on the delivery capacity, as well as their associated description/definition and how they are measured in the study.*	*RQ1*
Direction of the influence	*Positive or Negative. Explain*	*RQ2 and RQ3*
Measurement/Metric for delivery capability		*RQ2*
Evolution perspective	*Is the paper discuss variables in the perspective of evolution (over time)?*	*RQ4*
Type of study	*Controlled experiment, case study, action-research, other observational studies.*	*N/A*
Contextual information	*Characteristics from the organization, product, and/or project in which the evidence was observed.*	*N/A*
Architectural elements described in the paper	*Passage detailing the architectural elements or issues*	*RQ1*
Concrete architecture described in the paper	*Evidence of implementation of the architectural style into a concrete architecture*	*RQ1*

5. Quality Appraisal Criteria

We derived our quality criteria (Table 2) based on (Runeson and Höst, 2009)(Kitchenham et al, 2012), so that we included a more qualitative set of criteria for evaluating quality and a more experimental set, respectively. All questions have an equal weight and must be

answered with a score up to 2pts, with possible values being No (0), Partially (1), or Yes (2). This way, the paper quality score, ranging from 0 to 10, is the sum of all answer scores divided by 2.

Table 2.Quality appraisal criteria

ID	Quality Criterion
	Category: questions on aims
QC1	Are the objectives, research questions, and hypotheses (if applicable) clear and relevant?
QC2	Is the suitability of the case to address the research questions clearly motivated?
	Category: questions on design, data collection, and data analysis
QC3	Do the authors describe the cases (samples or experimental units)?
QC4	Do the authors describe the design of the study?
QC5	Do the authors describe the data collection procedures and define the measures?
QC6	Do the authors define the (quantitative/qualitative) data analysis procedures?
QC7	Is a clear chain of evidence established from observations to conclusions?
QC8	Are threats to validity analyses conducted in a systematic way and are countermeasures taken to reduce threats?
	Category: questions on study outcome
QC9	Do the authors state the findings clearly?
QC10	Is there evidence that the study can be used by other researchers/practitioners?

6. Results

Applying the search strings on the selected engines in January 2019, we got the results:

Engine	Papers
Scopus	444
IEEE	79
Web of Science	46
Total	*569*
Removing Duplicates	476

Ready for selection (removing non-papers entries)	**468**

Additionally, the engines provided the following duplications:

Involved Engines	**Papers**
Scopus internal	7
Scopus and IEEE	29
Scopus and Web of Science	25
Scopus and IEEE and WoS	*32 (16x2)*

6.1. Automatic search method

From the 468 papers, we selected 25 for the full reading and extraction. From these, we excluded 13 papers after the full reading. Mostly, these exclusions were motivated by (1) the lack of presentation or discussion of architectural elements and/or (2) the lack of an empirical/experimental study providing evidence. We present the list of included papers for the information extraction in the following:

[S1] S. Bellomo, R. L. Nord and I. Ozkaya, "A study of enabling factors for rapid fielding combined practices to balance speed and stability," *2013 35th International Conference on Software Engineering (ICSE)*, San Francisco, CA, 2013, pp. 982-991. doi: 10.1109/ICSE.2013.6606648

[S2] Bellomo, Stephany, Neil Ernst, Robert Nord, and Rick Kazman. "Toward design decisions to enable deployability: Empirical study of three projects reaching for the continuous delivery holy grail." In 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pp. 702-707. IEEE, 2014.

[S3] Chen, Lianping. "Towards architecting for continuous delivery." In 2015 12th Working IEEE/IFIP Conference on Software Architecture, pp. 131-134. IEEE, 2015.

[S4] H. Chen, R. Kazman and S. Haziyev, "Agile Big Data Analytics Development: An Architecture-Centric Approach," *2016 49th Hawaii International Conference on System Sciences (HICSS)*, Koloa, HI, 2016, pp. 5378-5387. doi: 10.1109/HICSS.2016.665

[S5] L. Chen, "Microservices: Architecting for Continuous Delivery and DevOps," *2018 IEEE International Conference on Software Architecture (ICSA)*, Seattle, WA, 2018, pp. 39-397.
doi: 10.1109/ICSA.2018.00013

[S6] V. Ivanov e K. Smolander, "Implementation of a DevOps Pipeline for Serverless Applications", in *Product-Focused Software Process Improvement*, 2018, p. 48–64.

[S7] Martin Lehmann and Frode Eika Sandnes. 2017. A framework for evaluating continuous microservice delivery strategies. In Proceedings of the Second International Conference on Internet of things, Data and Cloud Computing (ICC'17). Association for Computing Machinery, 1–9. DOI:https://doi.org/10.1145/3018896.3018961

[S8] Mårtensson, Torvald, Daniel Ståhl, and Jan Bosch. "Continuous integration

impediments in large-scale industry projects." In 2017 IEEE International Conference on Software Architecture (ICSA), pp. 169-178. IEEE, 2017

[S9] Shahin, Mojtaba, Muhammad Ali Babar, and Liming Zhu. "The intersection of continuous deployment and architecting process: practitioners' perspectives." In Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, p. 44. ACM, 2016

[S10] M. Shahin, M. Zahedi, M. A. Babar, e L. Zhu, "An empirical study of architecting for continuous delivery and deployment", *Empirical Software Engineering*, set. 2018.

[S11] Schermann, Gerald, Jürgen Cito, Philipp Leitner, Uwe Zdun and Harald C. Gall. "We're doing it live: A multi-method empirical study on continuous experimentation." Information & Software Technology 99 (2018): 41-57.

[S12] Villamizar, Mario, Oscar Garcés, Harold Castro, Mauricio Verano, Lorena Salamanca, Rubby Casallas, and Santiago Gil. "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud." In 2015 10th Computing Colombian Conference (10CCC), pp. 583-590. IEEE, 2015.

6.2. Snowballing method

Revision of seeds

We reviewed the seeds for possible bias due to authorship and communities. Out of the 12 seeds, three authors repeat (S. Bellomo, L. Chen, and Shahin and Babar). Even though, we decided to keep all papers from the automatic search method to use as seeds because:

- The two papers by Bellomo look at different aspects.
- Though papers from Chen and Shahin and Babar might describe the same research, the difference in publication year might result in losing forward citations if only the newest ones are selected (y. 2018).

Results from the snowballing search method

Using the papers listed in Section 6.1, we searched backwards for each paper. Therefore, the reviewer applied the selection (inclusion/exclusion) criteria using the paper titles in their references section. Besides, we searched for papers citing these seeds in the established search engines (Section 3.1) and also applied the selection criteria.

From these references and citations, we selected 21 candidates from the backward snowballing and 18 from the forward. Then, we reviewed these candidates in a full reading and, based on the selection criteria, we included two additional paper to the review:

[S13] F. Bachmann, R. L. Nord, and I. Ozkaya, "Architectural tactics to support rapid and agile stability," CrossTalk: The Journal of Defense Software Engineering, vol. 25, no. 3, pp. 20-25, 2012.

[S14] Balalaie A, Heydarnoori A, Jamshidi P (2016) Microservices architecture enables DevOps: migration to a cloudnative architecture. IEEE Softw 33(3):42–52.

At the end, we extracted information from **a total of 14 papers**, according to the extraction form presented in Section 4.

6.3. Quality appraisal results

Following the criteria defined in Section 5, the 14 studies were assessed as shown in Table 3. As stated before, the paper quality score, ranging from 0 to 10, is the sum of all answer scores divided by 2. The median for the 14 papers is 5, which indicates that a significant part of the papers (half) was evaluated as relatively low quality.

Table 3. Papers' quality appraisal

Study ID \ Quality Criterion	QC1	QC2	QC3	QC4	QC5	QC6	QC7	QC8	QC9	QC10	Total score
S1	2	1	2	2	2	2	1	0	2	2	8
S2	2	1	2	2	2	0	2	0	2	2	7.5
S3	2	2	0	0	0	0	0	0	1	0	2.5
S4	2	2	2	2	0	0	0	0	2	1	5.5
S5	2	2	0	0	0	0	0	0	1	0	2.5
S6	2	1	1	2	1	0	0	0	2	1	5
S7	0	0	0	0	0	0	0	0	0	0	0
S8	2	2	2	2	2	2	2	2	2	2	10
S9	2	2	2	2	2	2	2	2	2	0	9
S10	2	2	2	2	2	2	2	2	2	1	9.5
S11	2	2	2	2	2	2	2	2	2	2	10
S12	0	0	2	1	0	0	0	0	1	0	2
S13	0	0	0	0	0	0	0	0	0	0	0
S14	0	0	2	2	1	0	0	0	2	1	4

7. Analysis Procedure

The information to be extracted from selected papers is mostly qualitative. This way, Grounded Theory (GT) procedures should be performed to synthesize the findings using the same terminology, which allows a better categorization. The analysis should follow the GT three-phase process with open, axial, and selective coding. Also, it should aim to answer the research questions defined in Section 2. To that end, we define that the analysis must be exclusively performed using the extracted papers' information contained in the extraction form.

The answers resulting from the analysis procedure will be given in the format of a theoretical model, or at least a set of constructs, which represents the explanations for how architectural elements impact on delivery capability. The model or constructs will be induced and deduced from the synthesis according to interpretative directions that will be given, particularly, by the axial and selective coding.

Induced findings will be presented in the analysis as evidence originated from the primary studies. It is also expected that we can deduce some outcomes, which will be presented as hypotheses in that case. Hypotheses can be particularly deduced from the missing linkings in the Paradigm of (Straus and Corbin 1990).

8. References

ALVES, Nicolli SR et al. Identification and management of technical debt: A systematic mapping study. Information and Software Technology, v. 70, p. 100-121, 2016.

BECK, Kent; GAMMA, Erich. Extreme programming explained: embrace change. Addison-Wesley Professional, 2000.

BOGNER, Justus; WAGNER, Stefan; ZIMMERMANN, Alfred. Automatically measuring the maintainability of service-and microservice-based systems: a literature review. In: Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement. ACM, 2017. p. 107-115.

BOSCH, Jan (Ed.). Continuous Software Engineering. Springer International Publishing AG, 2014.

BRIAND, Lionel C.; DALY, John W.; WÜST, Jürgen. A unified framework for cohesion measurement in object-oriented systems. Empirical Software Engineering, v. 3, n. 1, p. 65-117, 1998.

BRIAND, Lionel C.; DALY, John W.; WÜST, Jürgen K. A unified framework for coupling measurement in object-oriented systems. IEEE Transactions on software Engineering, n. 1, p. 91-121, 1999.

CHEN, Lianping. Continuous delivery: Huge benefits, but challenges too. IEEE Software, v. 32, n. 2, p. 50-54, 2015.

CLAPS, Gerry Gerard; SVENSSON, Richard Berntsson; AURUM, Aybüke. On the journey to continuous deployment: Technical and social challenges along the way. Information and Software technology, v. 57, p. 21-31, 2015.

CODABUX, Zadia; WILLIAMS, Byron. Managing technical debt: An industrial case study. In: Proceedings of the 4th International Workshop on Managing Technical Debt. IEEE Press, 2013. p. 8-15.

DRAGONI, Nicola et al. Microservices: yesterday, today, and tomorrow. In: Present and Ulterior Software Engineering. Springer, Cham, 2017. p. 195-216.

de FRANÇA, Breno B. Nicolau; JERONIMO JUNIOR, Helvio; TRAVASSOS, Guilherme Horta. Characterizing DevOps by Hearing Multiple Voices. In: Proceedings of the 30th Brazilian Symposium on Software Engineering. Maringá, Brazil. ACM, 2016. p. 53-62.

EICK, Stephen G. et al. Does code decay? assessing the evidence from change management data. IEEE Transactions on Software Engineering, v. 27, n. 1, p. 1-12, 2001.

FITZGERALD, Brian; STOL, Klaas-Jan. Continuous software engineering: A roadmap and agenda. Journal of Systems and Software, 2015.

HUMBLE, Jez; FARLEY, David. Continuous delivery: reliable software releases through build, test, and deployment automation. Pearson Education, 2010.

JAKTMAN, Catherine Blake; LEANEY, John; LIU, Ming. Structural analysis of the software architecture—a maintenance assessment case study. In: Software Architecture. Springer, Boston, MA, 1999. p. 455-470.

KITCHENHAM, B. A.; SJØBERG, D. I. K.; DYBÅ, T.; et al. Three empirical studies on the agreement of reviewers about the quality of software engineering experiments. Information and Software Technology, v. 54, n. 8, p. 804–819, 2012.

LAUKKANEN, Eero; ITKONEN, Juha; LASSENIUS, Casper. Problems, causes and solutions when adopting continuous delivery—A systematic literature review. Information and Software Technology, v. 82, p. 55-79, 2017.

LEHMAN, Meir M. Programs, life cycles, and laws of software evolution. Proceedings of the IEEE, v. 68, n. 9, p. 1060-1076, 1980.

LEPPÄNEN, Marko et al. The highways and country roads to continuous deployment. IEEE Software, v. 32, n. 2, p. 64-72, 2015.

MARSCHALL, Matthias. Transforming a six-month release cycle to continuous flow. In: Agile Conference (AGILE), 2007. IEEE, 2007. p. 395-400.

MARTINI, Antonio; BOSCH, Jan; CHAUDRON, Michel. Architecture technical debt: Understanding causes and a qualitative model. In: Software Engineering and Advanced Applications (SEAA), 40th EUROMICRO Conference on. IEEE, 2014. p. 85-92.

MÄKINEN, Simo et al. Improving the delivery cycle: A multiple-case study of the toolchains in Finnish software intensive enterprises. Information and Software Technology, v. 80, p. 175-194, 2016.

MÄNTYLÄ, M. V., ADAMS, B., KHOMH, F., ENGSTRÖM, E., & PETERSEN, K. (2015). On rapid releases and software testing: a case study and a semi-systematic literature review. Empirical Software Engineering, 20(5), 1384-1425.

OLSSON, Helena Holmström; ALAHYARI, Hiva; BOSCH, Jan. Climbing the "Stairway to Heaven" - A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software. In: Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on. IEEE, 2012. p. 392-399.

PARNAS, David Lorge. Software aging. In: Proceedings of the 16th international conference on Software engineering. IEEE Computer Society Press, 1994. p. 279-287.

PETERSEN, Kai. Is Lean Agile and Agile Lean? Modern Software Engineering Concepts and Practices: Advanced Approaches, p. 19, 2010.

PETERSEN, Kai; WOHLIN, Claes. A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. Journal of systems and software, v. 82, n. 9, p. 1479-1490, 2009.

PETERSEN, K., VAKKALANKA, S., & KUZNIARZ, L. Guidelines for conducting systematic mapping studies in software engineering: An update. Information and Software Technology, 64, 1-18, 2015.

POPPENDIECK, Mary; POPPENDIECK, Tom. Lean software development: an agile toolkit. Addison-Wesley Professional, 2003.

RIAZ, Mehwish; SULAYMAN, Muhammad; NAQVI, Husnain. Architectural decay during continuous software evolution and impact of 'design for change' on software architecture. In: International Conference on Advanced Software Engineering and Its Applications. Springer, Berlin, Heidelberg, 2009. p. 119-126.

RODRÍGUEZ, P., HAGHIGHATKHAH, A., LWAKATARE, L. E., TEPPOLA, S., SUOMALAINEN, T., ESKELI, J., KARVONEN, T., KUVAJA, P., VERNER, J.M., OIVO, M. (2017). Continuous deployment of software intensive products and services: A systematic mapping study. Journal of Systems and Software, 123, 263-291.

RUNESON, Per; HÖST, Martin. "Guidelines for conducting and reporting case study research in software engineering." Empirical software engineering 14, no. 2 (2009): 131.

DE SILVA, Lakshitha; BALASUBRAMANIAM, Dharini. Controlling software architecture erosion: A survey. Journal of Systems and Software, v. 85, n. 1, p. 132-151, 2012.

SOLINSKI, Adam; PETERSEN, Kai. Prioritizing agile benefits and limitations in relation to practice usage. Software Quality Journal, p. 1-36. 2014.

TERRA, Ricardo et al. Recommending refactorings to reverse software architecture erosion. In: 2012 16th European Conference on Software Maintenance and Reengineering. IEEE, 2012. p. 335-340.

TOM, Edith; AURUM, AybüKe; VIDGEN, Richard. An exploration of technical debt. Journal of Systems and Software, v. 86, n. 6, p. 1498-1516, 2013.

TORKAR, Richard; MINOVES, Pau; GARRIGÓS, Janina. Adopting free/libre/open source software practices, techniques and methods for industrial use. Journal of the AIS, v. 12, n. 1, 2011.

WOMACK, J., JONES, D.T. Lean Thinking: Banish Waste and Create Wealth in Your Corporation. Productivity Press. 2003.