# CONTINUOUS INTEGRATION / CONTINOUS DELIVERY: REVIEW OF CHALLENGES AND SOLUTIONS

**Ina PAPADHOPULLI and Realb KUSHE**
Faculty of Information Technology, Polytechnic University of Tirana, Albania

---

## ABSTRACT

There is an increasing interest in literature on continuous practices (i.e., continuous integration, delivery, and deployment). For this reason, it is important to systematically review the approaches, tools and challenges related to these practices. In order to offer a "big view" of the continuous software engineering, this systematic literature review lists the strategies proposed to address and solve these challenges. We analyzed 46 relevant papers, filtered from four digital libraries. The implementation of Continuous Software Engineering practices is associated with challenges regarding the software builds, unit tests, integration tests and non-functional ones. Several strategies have been proposed by academics to address these challenges. In addition, tools have been developed to automate each stage of CI/CD pipeline. The progress in the optimization of continuous software engineering practices is inspired by the industrial needs. This Systematic Literature Review (SLR) emphasizes that as in many other engineering problems, there is no optimal continuous software engineering architecture to fulfil all the clients' needs. Selection of the right automation tool for continuous software development project depends on the sort of project.

***Keywords*:** continuous software engineering, continuous integration/continuous deployment, continuous software testing

## 1. INTRODUCTION

The Agile methodology defines the processes used to change software features and accelerate delivery. "DevOps culture" specifies roles and responsibilities of each human actor in a software development project; software developer or IT specialist, to increase their responsiveness. Continuous software engineering focuses on tools used for automation of software defined life-cycles (Doukoure and Mnkandla 2018). Thanks to these

practices, the gap between software development teams and operational ones has been steadily narrowed during the past years (Felidré *et al.*, 2019). The three development activities are continuous integration (CI), continuous delivery (CDE) and continuous deployment (CD). CDE means that the software **can** be deployed to production at any time, whereas CD means that the software is **automatically** deployed to production all the time. The relationship between these concepts is in Figure 1 depicted (Shahin *et al.*, 2017).

CI/CD has the following benefits: i) software of higher quality, ii) faster delivery of features to the customer, iii) easy to be used, simple and flexible to the needs of customers due to the ability to change things more quickly. Javed *et al.*, (2020) recommended to follow various principles and best practices to achieve these benefits:

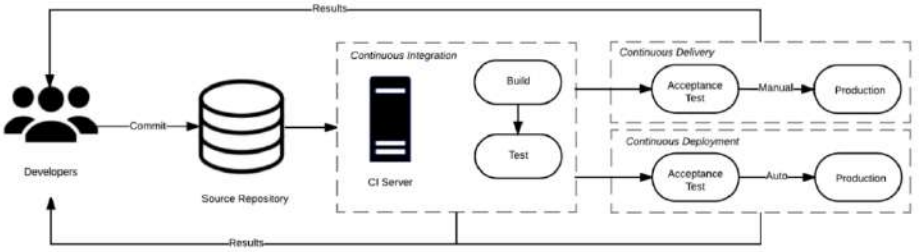


**Fig. 23:** Relationship between CI, CDE and CD (Shahin *et al*., 2017).

More and more strategies (mainly by academic researchers) and tools (mainly by industrial researchers) are being developed in support of each stage of CI/CD pipeline. There are difficulties to transition to CI/CD. Even when the team has successfully introduced the CI/CD culture, living up to its principles and improving the CI/CD practice is also challenging (Ciancarini and Missiroli 2020)

This SLR aims to provide a comprehensive analysis of CSE by balancing the benefits of its applications with their related limitations. Information about the latest research with regard could be found in (Shahin *et al*., 2017).

**Research Questions (RQ):** The study addresses three research questions:

**RQ1:** What are the limitations in the implementation of CI/CD stages?

**RQ2:** What strategies have been proposed to address CI/CD challenges?

**RQ3:** Which are the state-of-the-art tools for designing and implementing the deployment pipeline?

In summary, this paper makes the following contributions:

1. An analysis of the problems that arise during the implementation of CI/CD pipeline grouped by CI/CD stage
2. A list of solutions proposed in the last five years for the problems addressed in RQ1.
3. A comparison of the CI/CD state-of-the-art tools

Section 2 presents the used Review Protocol. Sections 3 - 5 addressed the three research questions. Conclusions are drawn in the end.

## 2 REVIEW PROTOCOL

### 2.1 Study Selection Process:

**Metadata filtering:** Research papers found in digital libraries were filtered based on metadata indicators: title of the paper ("Is this related to CSE?"), author names, date published with respect to inclusion and exclusion criteria as above defined.

**Abstract filtering:** Once the abstract was read, papers on DevOps culture or theoretical part of Agile methodologies were excluded from further reading.

**Content-based filtering:** Only papers answering to the research questions were taken into consideration for this SLR.

In the present investigation, 46 relevant papers were systematically identified and rigorously reviewed. In addition, synthetization of the data extracted from these papers to answer the research questions was made.

**Table 8** The evaluation of research papers during study selection process.

| **Search Engine** | **First-time collected (metadata filtering)** | **Included after filter application related to:** | | | **Total** |
|---|---|---|---|---|---|
| | | Abstract | Content | Repetition of ideas | |
| ***IEEE Xplore*** | *49* | *43* | *40* | *29* | ***29*** |
| ***ACM*** | *23* | *21* | *16* | *9* | ***9*** |
| ***Science Direct*** | *8* | *6* | *5* | *5* | ***5*** |
| ***Scopus*** | *5* | *4* | *3* | *3* | ***3*** |
| **Total** | ***85*** | ***74*** | ***64*** | ***46*** | ***46*** |

## 3. RQ1: WHAT ARE THE LIMITATIONS IN THE IMPLEMENTATION OF CI/CD STAGES?

The challenges are divided into groups in according to the correspondent CI/CD stage.

3.1 **Problems related to Continuous Integration**

**Lack of frequent commits**: (Pinto *et al.* 2018) emphasized that the most common CI problem reported by their survey group was infrequent commits due to time pressure. Felidré *et al.*, (2019) said that "2.36 commits/ weekday" is the lowest threshold value for a software development project to succeed, independently of the project size as based on (Cavalcanti *et al*., 2018).

**Time-consuming builds**: For a large-scale software project, the build can take hours as it includes compilation, unit and acceptance testing (Jin and Servant 2020). Continuous submission of code modification by developers and build latency time creates stalls at CI server build pipeline, and hence developers have to wait long time for the build outcome (Fan, 2019). These builds compete for system resources with other jobs waiting in the processing queue (Bezemer 2017).

**Broken builds:** Builds can be unsuccessful for a variety of reasons (Rebouc *et al*., 2017) points out the gap in the real-time addressing of problematical builds between commercial projects and open-source ones. Commercial projects tend to enter in a "fast-recovery" mode while open-source ones seem to offer a slower but more consolidated solution for the build failure (Avelino *et al*., 2016).

**3.2 Problems related to "Continuous Testing" (CT)**

**3.2.1 Unit tests**

**Writing automated tests is time-consuming:** DiffBlue survey (Zalozhnev, 2017) found that software engineers, among the most expensive talent in any company, spend 20% of their time writing unit tests and an additional 15% of their time writing all other types of tests.

**Manual testing:** DiffBlue survey (Camargo *et al*., 2016) found manual testing as a key bottleneck in a CI/CD pipeline. The reason is that resources are not invested in automated testing. When asked which stage of the DevOps pipeline respondents feel their organization places as its top priority, 51% chose developing, with deploying in second place (24%). Testing fell in last place (11%).

**Non-deterministic automated tests and code:** A considerable amount of disturbance to CI/CD pipeline is caused by non-deterministic (flaky) automated tests. These tests capriciously generate variable results even without changing the isolated tested code (Gallaba 2019). Maintaining flaky tests is costly, especially in large-scale software projects (Diffblue 2021b).

**Poor test quality:** Unreliable tests, high number of test cases, low test coverage and long running tests can impede the deployment pipeline and reduce the confidence of organizations to automatically deploy software on a

continuous basis (Shahin *et al*., 2017). Test-coverage is mainly limited by the use of an old-fashion testing metric: line coverage (Kim *et al*., 2017).

### 3.2.2 Integration Tests

**System heterogeneity**: The complexity of CT stands in its heterogeneity: distributed testing requires the participation of a lot of hardware resources shared between multiple platforms. Test results can be prone to errors while traversing the communication links especially in the master-slave architecture (Aghamohammadi *et al*., 2021).

**Version Control (VI):** A critical point for CI is the Version Control of the shared repository. Updates to the Version Control may trigger instability of the system: widespread file-locking and corrupted copies of the program files have been reported by CI processes in diverse software development projects (Xu *et al*., 2019).

### 3.2.3 Non-functional Testing

**Difficulty to automate performance tests:** Developing performance testing automation scripts is not a trivial task. Automating this process requires strong tool support. A lack of existing tools means that performance testing is normally left out of the scope of CI (Diffblue 2021a).

**Skip security tests:** Security tests in the CI stages are extremely important as they guarantee that none of weak dependencies between entities will process in the rest of the deployment pipeline.

## 3.3 Problems related to "Continuous Deployment"

**Fulfill quality assurance step:** Quality assurance (QA) is the final step before pushing the software to production. The fulfillment of both development and QA constraints is a difficult task (Parnin *et al*., 2017). The implementation of a unified framework for both teams is associated with additional costs of training project's members for its usage.

**Duplicate production environment:** Creating a duplicate production environment (shadow infrastructure) in order to enable software experimentation and accelerate software production is costly (Macho 2017).

**Different Customer Environment:** Shahin *et al*., (2017) said that continuously releasing software product to multiple customers with diverse environments is quite difficult as different deployment configurations for each customer's environment and component's version are needed to be established.

**Maintenance window:** Service deployment, including upgrading to new versions, rolling back to older ones, or introducing fix patches in case of a failed deployment, can be done during a maintenance window while reusing

the infrastructure resources due to the high cost of hardware and its maintenance (Pinto *et al*., 2018)

**Smells in CD configuration files:** Typical configuration files for specialized build tools which depend on the programming languages are usually too complex. This is the reason why there may be many smells in CD pipelines like 'Fake Success', 'Retry Failure', 'Manual Execution' and 'Fuzzy Version' (Javed *et al*., 2020).

## 4. RQ2: WHAT STRATEGIES HAVE BEEN PROPOSED TO ADDRESS CI/CD CHALLENGES?

Several strategies have been proposed for solving the CI/CD challenges. In order to answer to the RQ2, after analyzing the proposed solutions, we have made a mapping between the proposed solution and the challenge it addresses (Table 2).

**Table 2.** Mapping between challenges found in CI/CD pipeline and their proposed solutions. Problem Category (PC): ① (broken builds); ② (long running builds); ③ (Writing automated Unit tests is time-consuming); ④ (Long running unit tests); ⑤ (non-deterministic automated tests); ⑥ (Version Control); ⑦ (Difficulty to automate performance tests); ⑧ (Duplicate production environment); ⑨ (Maintenance window); ⑩ (Smells in CD configuration files);

| Solution | PC | Description of the solution | Ref |
|---|---|---|---|
| **"Filter and flush" architecture** | ① | Records metadata of the previous "failed builds" to reject builds that have crashed in the past. | (Hassan and Wang,2017) |
| **"Taxonomy of broken builds"** | | Broken builds are classified based on their cause and their impact on the project. Most influential builds are tried to repair with highest priority. | (Konersman n *et al*., 2020) |
| **"Exploration of dependencies between builds"** | ② | Proposal to use third-party tools to design annotated graphs that study dependencies between builds. | (Fan 2017) |
| **"Benefit from local spatiality"** | | Builds with similar features are grouped together and a "build agent" examines and extracts their similarities. Builds from different clusters run concurrently. | (Melo and Rocio, 2017) |

| | | | |
|---|---|---|---|
| **Build prediction models** | | Model that uses previous data to predict whether a build will be successful or not without attempting actual build so that developer can get early build outcome result. | (Yang *et al*., 2018) (Fan 2017) |
| **Tool: SmartBuildSkip** | | Use ML to predict the first builds in a sequence of build failures | (Jin and Servant, 2020) |
| **Skip commits** | | Automate the process of determining which commits can be CI skipped through the use of ML techniques | (Singh *et al*., 2019) |
| **Tool: Evosuite** | ③ | Search Based Software Testing tool used to automatically generate unit tests for Java applications | (Francalino *et al*., 2018) |
| **Tool: DiffBlue Cover** | | Tool that uses AI to automatically write suites of unit tests for Java code | (Abdalkareem *et al*., 2021) (Diffblue 2021c) |
| **Testing as a Service"** | ④ | Automated Unit tests are executed parallelly in a distributed cloud infrastructure. | (King *et al*., 2018) |
| **Choose a subset of test to execute** | | Use ML to predict which group of tests should be executed after each change submitted to the CI system. | (Islam and Zibran, 2017) |
| **Postpone re-execution of flaky test** | ⑤ | Non-deterministic tests are marked with a flag and re-executed after all other tests to identify the cause of ambiguity and dependencies that caused the failure | (Deepa *et al*., 2020) (Javed *et al.*, 2020) |
| **Probabilistic approach to detect faulty tests** | | Fault localization can be achieved by creating a Bayesian network model that takes into consideration a broad range of tests metric: assertion count, test size, cyclomatic complexity etc. | (Diffblue 2021c) |
| **Maintenance of VI System** | ⑥ | VI updates should be scheduled at "off-hours". A trade-off should be found between the need for on-time VI updates and the need for the release of stable software products. | (Williams, 2021) (Javed *et al.*, 2020) |
| **PerfCI Tool** | ⑦ | PerfCI - helps developers to easily set up and carry out automated performance testing under CI | (Diffblue, 2021c) |
| **Be fast to deploy but slower to release** | ⑧ | Combination of 'dark launches' and 'feature flags' | (Macho, 2017) |

| | | | |
|---|---|---|---|
| **Blue-Green Deployment Technique** | ⑨ | Usage of Blue-Green Deployment Technique which targets to enable service updates with zero maintenance windows, and thus with no disruption to the end users | (Pinto *et al.*, 2018) |
| **Tool: CD-Linter** | ⑩ | A semantic linter that can automatically identify four different smells in pipeline configuration files on GitLab. | (Javed *et al.*, 2020) |
| **Tool: Xeditor** | | Tool that extracts configuration couplings from Deployment Descriptors, and adopts the coupling rules to validate new / updated files | (Wen *et al.*, 2020) |

## 5. Q3: WHICH ARE THE STATE-OF-THE-ART TOOLS FOR DESIGNING AND IMPLEMENTING THE CI/CD PIPELINE?

Choosing the right CI/CD tool is an essential part that can define the success rate of a software development project. In this SLR, we focused on the tools that automate the whole pipeline since they don't require extra synchronization with other tools.

The comparison is made based on two principles:

1. Research papers which describe the features of the CI/CD tools have made a transparent evaluation of the tool's metric.
2. Since there are different methods to evaluate the tools, we developed a numerical scale to compare the set of tools.

Figure 2 shows the **proposed numerical scale** for the evaluation procedure.

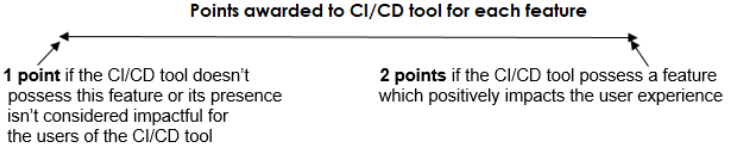


**Fig. 2.** Scoring points evaluation procedure

Features considered:

- Ease of install, ease of upgrade and backup
- If the CI/CD tool is an open-source tool or not
- If the hosting model includes both "On-premise" and "Cloud"
- If the CI/CD tool provides "test parallelization" in distributed environments
- Graphical pipeline view

- Re-usable pipelines

Once the features information was collected from a set of research papers on behalf of 7 CI/CD tools, the tools were compared by using the proposed numerical scale (Table 3).

This score-based evaluation shows that TravisCI tool is the most maturated tool for the implementation of CI/CD pipeline followed by TeamCity and GitLab.

**Table 3**. Comparison of CI/CD tool.

| | Jenkins | TeamCity | Bamboo | CircleCI | GitLab | TravisCI | GoCD |
|---|---|---|---|---|---|---|---|
| **Ease of install** | 2 | 2 | 2 | 1 | 2 | 2 | 1 |
| **Ease of upgrade** | 1 | 2 | 1 | 1 | 1 | 2 | 1 |
| **Ease of backup** | 1 | 2 | 2 | 2 | 2 | 2 | 2 |
| **License of tool** | 2 | 1 | 1 | 2 | 2 | 2 | 2 |
| **Hosting model** | 2 | 1 | 1 | 2 | 2 | 1 | 2 |
| **Test case parallelization** | 2 | 2 | 2 | 2 | 2 | 2 | 1 |
| **Reusable Pipelines for Microservices** | 2 | 2 | 1 | 2 | 1 | 2 | 2 |
| **Graphical Pipeline View** | 1 | 2 | 2 | 1 | 2 | 2 | 2 |
| **Total Points** | **13** | **14** | **12** | **13** | **14** | **15** | **13** |

The features were extracted from (Souza and Silvia, 2017; Hohl *et al*., 2018; Diffblue, 2021c).

## 6. CONCLUSIONS AND FUTURE WORK

CSE practices are being highly adopted by companies, which are transitioning their strategy from developing stand-alone programs to offering software as a service.

This SLR offers a bilateral analysis of both the problems that arise during the implementation of CI/CD pipeline and their counterpart solutions. Each phase of Continuous Development pipeline deals with implementation challenges (Section 3). There exist many proposed solutions to local and isolated problems, but it is hard to implement them in a vector of mixed CI/CD problems. This SLR emphasizes that, as in many other engineering problems, there is no optimal CSE architecture to fulfill all client's needs. Selection of the right automation tool is based in the nature of project.

In the present paper the tools that automate the whole CI/CD pipeline are compared, and our score-based evaluation shows that TravisCI is the most maturated tool.

Based on the number of papers on continuous practices that we found, we can conclude that in the last five years there has been a high interest in this field from academic and industrial researchers.

Several papers (i.e., 10 papers) proposed solutions based on AI techniques as an alternative to deterministic approaches to solve difficult problems related to CI/CD. It seems that the research is focused more on this direction.

Research papers reviewed in this systematic literature review were extensively focused on improving the CSE pipeline. However, we found that there is little work done for the investigation of how do bad DevOps practices actually interfere with Continuous Software Engineering ones.

## REFERENCES

**Abdalkareem R, Mujahid S, Shihab E. 2021.** A Machine Learning Approach to Improve the Detection of CI Skip Commits. International Conference on Software Engineering.

**Aghamohammadi A, Mirian-Hosseinabadi SH, Jalali S. 2021.** Statement frequency coverage: A code coverage criterion for assessing test suite effectiveness. Information and Software Technology.

**Avelino G, Passos P, Hora A, Valente MT. 2016.** A novel approach for estimating truck factors. 24th IEEE International Conference on Program Comprehension, USA, 2016.

**Bezemer C.-P, McIntosh S, Adams B, German D. M, Hassan. A. E. 2017.** An empirical study of unspecified dependencies in make-based build systems. *Empirical Software Engineering* (EMSE), **22(6):** 317–324, 217.

**Camargo A, Salvadori I, Mello S, Siqueira F. 2016.** An architecture to automate performance tests on microservices. In Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services (iiWAS '16). USA.

**Ciancarini P, Missiroli M. 2020.** The Essence of Game Development. IEEE 32nd Conference on Software Engineering Education and Training, Germany, pp.11-14.

**Deepa N, Prabadevi B, Krithika LB, Deepa B. 2020.** An analysis on Version Control Systems. International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), Vellore, Spain, pp. 5-9.

**Diffblue.** https://www.diffblue.com/ [Last accessed: 10 June 2021].

**Diffblue.**https://www.diffblue.com/DevOps/research_papers/2020-devops-and-testing-report/[Last accessed: 1 September 2021].

**Diffblue.**https://www.diffblue.com/Education/research_papers/2019-diffblue-developer-survey/[Last accessed: 10 June 2021].

**Doukoure GAK, Mnkandla E. 2018.** Facilitating management of agile and DevOps activities: Implementation of a data consolidator. International

Conference on Advances in Big Data, Computing and Data Communication Systems, United Kingdom.

**Fan Z. 2019.** A systematic evaluation of problematic tests generated by EvoSuite. In Proceedings of the 41st International Conference on Software Engineering.

**Felidré W, Furtado LB, da Costa DA, Cartaxo B, Pinto G. 2019.** Continuous integration theater. *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*: 1-10. Porto de Galinhas, Recife, Brazil.

**Francalino W, Callado A, Matthews Jucá P. 2018.** Defining and implementing a test automation strategy in an IT Company. In Proceedings of the Euro- American Conference on Telematics and Information Systems (EATIS'18).

**Gallaba K. 2019.** Improving the robustness and efficiency of continuous integration and deployment. International Conference on Software Maintenance and Evolution.

**Hassan F, Wang X. 2017.** Change-aware build prediction model for stall avoidance in continuous integration. In 2017 ACM/IEEE (ESEM), pages 157–162. IEEE.

**Hohl P, Stupperich M, Munch J, Schneider K. 2018.** Combining agile development and software product lines in automotive: Challenges and recommendations. IEEE (ICE/ITMC).

**Islam MR, Zibran MF.2017.** Insights into continuous integration build failures. IEEE/ACM 14th International Conference on Mining Software Repositories (MSR).

**Javed O, Dawes JH, Han M, Franzoni G, Pfeiffer A, Reger G, Binder W. 2020.** PerfCi: A toolchain for automated performance testing during continuous integration of Python projects (ASE).

**Jin X, Servant F. 2020.** A Cost-efficient Approach to Building in Continuous Integration. In Proceedings of the 43rd International Conference on Software Engineering, 2020.

**Kim J, Jeong H, Lee E. 2017.** Failure history data-based test case prioritization for effective regression test. Proceedings of the symposium on applied computing.

**King TM, Santiago D, Phillips J, Clarke PJ. 2018.** Towards a Bayesian network model for predicting cases of flaky automated tests. IEEE International conference on software quality, reliability and security companion.

**Konersmann M, Fitzgerald B, Goedicke M, Olsson H, Bosch J, Krusche S. 2020.** Rapid continuous software engineering - State of the practice and open research questions: SIGSOFT.

**Macho C. 2017.** Preventing and repairing build breakage. IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), pp. 471-475.

**Melo S, Rocio S. 2017.** How to test your concurrent software: an approach for the selection of testing techniques. In Proceedings of the 4th ACM SIGPLAN

International Workshop on Software Engineering for Parallel Systems (SEPS), New York USA, pp.42–43.

**Parnin C, Helms E, Atlee C, Boughton H, Ghattas M, Glover A, Williams L. 2017.** The top 10 adages in continuous deployment. *IEEE Software*, **34(3):** 86–95.

**Pinto G, Castor F, Bonifacio R, Rebouc M. 2018.** Work practices and challenges in continuous integration: A survey with travis CI users. Softw., Pract.Exper.

**Rebouc M, Santos R, Pinto G, Castor F. 2017.** How does contributors' involvement influence the build status of an open-source software project? 14th International Conference on Mining Software Repositories, pages 475–478, USA.

**Shahin M, Ali Babar M, Zhu L. 2017.** Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices, April.

**Singh C, Gaba S, Kaur M, Kaur B. 2019.**Comparison of Different CI/cd tools integrated with cloud platform. *9th International Conference on Cloud Computing*, pp. 7-12.

**Souza R, Silva B. 2017.** Sentiment Analysis of Travis CI Builds. *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)* pp. 459-462.

**Wen Ch, he X, Zhang Y, Meng N. 2020.** Inferring and applying Def-use like configuration couplings in deployment descriptors. *35th IEEE/ACM International Conference on Automated Software Engineering (ASE):* 672-683.

**Williams N. 202).** Towards exhaustive branch coverage with PathCrawler. IEEE/ACM international workshop on automation of software test.

**Xu, X, Cai, Q, Lin J, Pan S, Ren L. 2019.** Enforcing access control in distributed version control systems, IEEE International Conference on Multimedia and Expo (ICME), Shanghai, China, pp. 772-777.

**Yang B, Saller A, Jain S, Tomala-Reyes E, Singh M, Ramnath A. 2018.** Service Discovery based Blue-Green Deployment Technique in Cloud Native Environments. *IEEE International Conference on Services Computing (SCC)*, San Francisco, CA, USA, *pp.* 108–121.

**Zalozhnev AY. 2017.** Big banks systems management software: Architecture. General requirements and functional components. 10th International Conference in Management of Large-Scale System Development, Moscow. 103-108.