Available online www.ejaet.com

European Journal of Advances in Engineering and Technology, 2019, 6(9):66-76

Research Article ISSN: 2394 - 658X

Performance Testing Framework for CCAR and Regulatory Stress Testing Software: Optimizing Scalability and Resilience

Praveen Kumar

NJ, USA
praveenk@gmail.com

ABSTRACT

Comprehensive Capital Analysis and Review (CCAR) and regulatory stress testing have become critical components of the financial industry's risk management practices. These exercises require robust and reliable software systems capable of processing large volumes of data, performing complex calculations, and generating accurate results within strict timeframes. Ensuring the performance, scalability, and resilience of these systems is crucial to meet regulatory requirements and maintain financial stability. This paper presents a performance testing framework specifically designed for CCAR and regulatory stress testing software. The framework emphasizes the importance of optimizing system scalability, resilience, and responsiveness under stress conditions. It outlines key considerations for designing and executing performance tests, including workload modeling, test environment setup, and monitoring and analysis techniques. The paper also discusses best practices for identifying performance bottlenecks, optimizing resource utilization, and ensuring system stability under peak loads. By adopting the proposed performance testing framework, financial institutions can enhance the reliability and efficiency of their CCAR and stress testing processes, ultimately strengthening their risk management capabilities and regulatory compliance.

Keywords: Comprehensive Capital Analysis and Review (CCAR)

INTRODUCTION

A. Background

1.Importance of CCAR and regulatory stress testing in the financial industry

CCAR and regulatory stress testing are critical risk management exercises mandated by regulatory authorities to assess the capital adequacy and financial resilience of banks and financial institutions [1].

These exercises involve simulating adverse economic scenarios and evaluating the impact on an institution's capital levels, loan portfolios, and overall financial health [2].

2.Role of software systems in CCAR and stress testing processes

CCAR and stress testing processes rely heavily on sophisticated software systems to handle data management, scenario modeling, risk calculations, and reporting [3].

These systems must be capable of processing large volumes of data, performing complex simulations, and generating accurate results within tight deadlines [4].

3. Significance of performance testing for CCAR and stress testing software

Performance testing is crucial to ensure that CCAR and stress testing software can handle the demanding requirements of these exercises [5].

Inadequate performance, scalability, or resilience of these systems can lead to inaccurate results, missed deadlines, and regulatory non-compliance [6].

B. Objectives and Scope

1. Research questions addressed in the paper

What are the key considerations and challenges in performance testing of CCAR and regulatory stress testing software?

How can a performance testing framework be designed to optimize the scalability, resilience, and responsiveness of these systems?

What are the best practices and techniques for executing performance tests and analyzing results in the context of CCAR and stress testing?

2. Scope and limitations of the study

The paper focuses on performance testing aspects specific to CCAR and regulatory stress testing software, considering the unique requirements and challenges of these applications.

The study does not cover the detailed technical implementation of performance testing tools or the specifics of CCAR and stress testing methodologies.

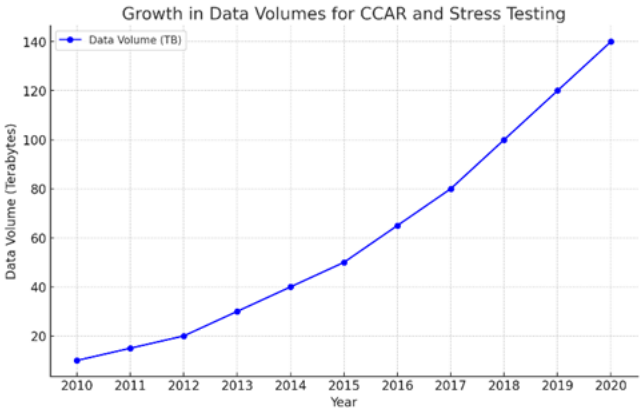
3.Target audience and intended contributions

The target audience for this paper includes software quality assurance professionals, performance engineers, risk managers, and IT personnel involved in CCAR and stress testing processes.

The paper aims to provide a practical framework and guidelines for designing and executing effective performance tests to optimize the scalability and resilience of CCAR and stress testing systems.

PERFORMANCE TESTING CHALLENGES IN CCAR AND STRESS TESTING SOFTWARE

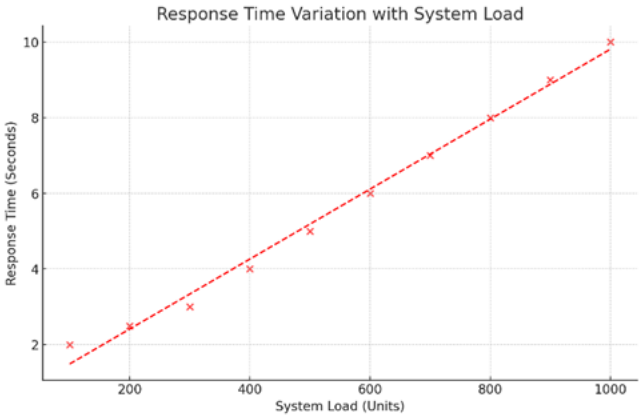
A. Data Volume and Complexity



1. Handling large datasets and data feeds

CCAR and stress testing processes involve handling massive volumes of financial data, including historical data, market data, and risk factors [7].

Performance tests must validate the system's ability to efficiently process and store large datasets without compromising performance or data integrity [8].



2. Data quality and consistency challenges

Ensuring data quality and consistency is critical for accurate stress testing results [9].

Performance tests should incorporate data validation and reconciliation mechanisms to identify and handle data quality issues during high-volume processing [10].

3. Data integration and aggregation bottlenecks

CCAR and stress testing systems often require integrating data from multiple sources and aggregating results across different levels of granularity.

Performance tests must assess the efficiency of data integration and aggregation processes and identify potential bottlenecks that can impact overall system performance [12].

B. Computational Intensity and Complexity

1. Complex risk models and calculations

Stress testing involves running complex risk models and calculations to assess the impact of different scenarios on financial portfolios [13].

Performance tests should evaluate the system's ability to handle computationally intensive tasks and ensure acceptable response times for risk calculations [14].

2. Concurrent and parallel processing requirements

CCAR and stress testing workloads often require concurrent and parallel processing to meet tight deadlines and handle large volumes of data [15].

Performance tests must validate the system's scalability and ability to efficiently utilize available hardware resources for concurrent processing [16].

3. Memory and CPU utilization optimization

Efficient memory and CPU utilization is crucial to ensure optimal performance and resource management during stress testing [17].

Performance tests should monitor and analyze memory and CPU usage patterns to identify resource contention and optimize system configurations [18].

C. Regulatory Compliance and Reporting

1. Strict deadlines and time constraints

CCAR and stress testing exercises have strict regulatory deadlines for submitting results and reports [19].

Performance tests must validate the system's ability to generate accurate results and reports within the specified timeframes, even under peak load conditions [20].

2. Regulatory reporting and data submission requirements

Stress testing results and reports must adhere to specific regulatory formats and data submission requirements [21].

Performance tests should incorporate validations for generating compliant reports and assess the efficiency of data submission processes [22].

3. Auditability and traceability of results

Stress testing processes must ensure the auditability and traceability of results for regulatory review and validation [23].

Performance tests should verify the system's ability to maintain accurate audit trails and provide transparent traceability of calculations and results [24].

DESIGNING A PERFORMANCE TESTING FRAMEWORK FOR CCAR AND STRESS TESTING SOFTWARE

A. Workload Modeling and Test Scenario Design

1. Identifying critical business processes and workflows

Identify the critical business processes and workflows involved in CCAR and stress testing, such as data ingestion, risk calculations, and reporting [25].

Prioritize performance testing efforts based on the criticality and potential impact of each process on the overall system performance [26].

2. Defining representative workload profiles and test scenarios

Develop representative workload profiles that simulate realistic usage patterns and data volumes for CCAR and stress testing processes [27].

Design test scenarios that cover various aspects of system performance, including peak loads, concurrent users, and data variability [28].

3. Incorporating regulatory requirements and constraints

Incorporate regulatory requirements, such as calculation accuracy, reporting formats, and submission deadlines, into the test scenario design [29].

Ensure that performance tests validate the system's compliance with regulatory guidelines and constraints under different workload conditions [30].

B. Test Environment Setup and Configuration

1. Replicating production-like infrastructure and configurations

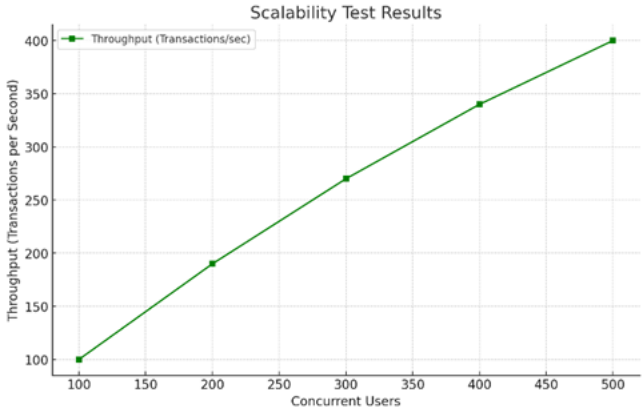
Set up a test environment that closely replicates the production infrastructure, including hardware, software, and network configurations [31].

Ensure that the test environment provides a realistic representation of the production system's performance characteristics and constraints [32].

2. Scalability and load injection considerations

Determine the scalability requirements for CCAR and stress testing processes based on expected data volumes and concurrent users [33].

Configure load injection tools and techniques to simulate increasing workloads and assess the system's scalability and resource utilization [34].



3. Test data management and security

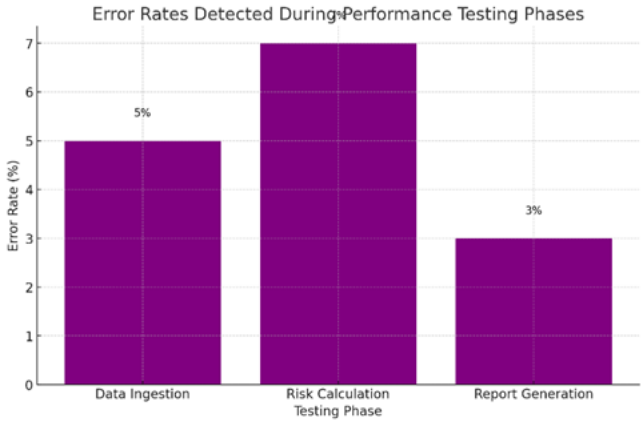
Manage test data effectively, ensuring data quality, consistency, and representative coverage of different scenarios [35].

Implement appropriate security measures to protect sensitive financial data during performance testing and comply with data privacy regulations [36].

C. Monitoring and Analysis Techniques

1. Identifying key performance metrics and thresholds

Define key performance metrics and thresholds relevant to CCAR and stress testing processes, such as response times, throughput, resource utilization, and error rates [37].



Establish baseline performance levels and acceptable thresholds for each metric based on business requirements and regulatory expectations [38].

2. Real-time monitoring and data collection

Implement real-time monitoring and data collection mechanisms to capture performance metrics during test execution [39].

Use monitoring tools and techniques to track system behavior, identify performance bottlenecks, and detect anomalies in real-time [40].

3. Post-test analysis and performance optimization

Conduct post-test analysis to evaluate system performance against defined metrics and thresholds [41].

Identify performance bottlenecks, resource constraints, and optimization opportunities based on the collected data and analysis results [42].

Collaborate with development teams to implement performance optimizations and fine-tune system configurations for improved scalability and resilience [43].

BEST PRACTICES AND RECOMMENDATIONS

A. Collaborative Performance Testing Approach

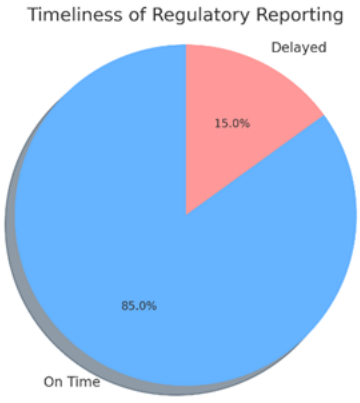
1. Involving cross-functional teams

Foster collaboration among cross-functional teams, including development, testing, infrastructure, and business stakeholders, throughout the performance testing lifecycle [44].

Ensure that performance testing goals and strategies align with business objectives and regulatory requirements [45].

2. Continuous communication and feedback loops

Establish continuous communication channels and feedback loops among stakeholders to share performance testing results, insights, and improvement recommendations [46].



Regularly review and discuss performance testing progress, challenges, and optimization efforts to drive continuous improvement [47].

3. Performance testing in Agile and DevOps environments

Integrate performance testing into Agile and DevOps workflows to enable early and frequent performance feedback [48].

Automate performance tests and incorporate them into continuous integration and delivery pipelines to catch performance issues early in the development cycle [49].

B. Leveraging Automation and Tools

1. Automated test script development and execution

Develop reusable and maintainable performance test scripts using automation frameworks and tools [50].

Automate test execution to enable efficient and repeatable performance testing across different scenarios and configurations [51].

2. Cloud-based performance testing solutions

Explore cloud-based performance testing solutions to leverage scalable infrastructure and on-demand resources [52].

Utilize cloud platforms to simulate realistic workloads, geographically distributed users, and burst capacity requirements [53].

3. AI and machine learning for performance analysis

Leverage artificial intelligence (AI) and machine learning techniques to analyze performance test results and identify patterns, anomalies, and optimization opportunities [54].

Apply predictive analytics to forecast performance trends, capacity requirements, and potential bottlenecks based on historical data and machine learning models [55].

C. Continuous Performance Testing and Optimization

1. Integrating performance testing into the software development lifecycle

Integrate performance testing activities into the software development lifecycle (SDLC) from the early stages of requirements gathering and design [56].

Conduct performance testing iteratively throughout the development process to identify and address performance issues proactively [57].

2. Establishing performance baselines and benchmarks

Establish performance baselines and benchmarks based on historical data, industry standards, and regulatory requirements [58].

Use these baselines to set performance goals, track progress, and measure the effectiveness of optimization efforts [59].

3. Continuous monitoring and performance tuning

Implement continuous monitoring solutions to track system performance in production environments [60].

Analyze production performance data to identify improvement opportunities and fine-tune system configurations for optimal performance and resource utilization [61].

CONCLUSION

A. Recap of Key Findings and Recommendations

1. Importance of a comprehensive performance testing framework for CCAR and stress testing software

A comprehensive performance testing framework is essential to ensure the scalability, resilience, and responsiveness of CCAR and stress testing software systems [62].

The proposed framework addresses the unique challenges and requirements of these systems, including data volume and complexity, computational intensity, and regulatory compliance [63].

2. Benefits of adopting the proposed performance testing framework

Adopting the proposed performance testing framework enables financial institutions to optimize the performance and reliability of their CCAR and stress testing processes [64].

The framework provides a structured approach to designing and executing performance tests, identifying bottlenecks, and implementing optimizations for improved system performance [65].

3. Recommendations for successful implementation and continuous improvement

Successful implementation of the performance testing framework requires collaboration among cross-functional teams, leveraging automation and tools, and integrating performance testing into the SDLC [66].

Continuous monitoring, performance tuning, and establishing baselines and benchmarks are essential for ongoing performance optimization and improvement [67].

B. Future Research Directions

1. Exploring the impact of emerging technologies on CCAR and stress testing performance

Future research can investigate the impact of emerging technologies, such as big data analytics, machine learning, and cloud computing, on the performance of CCAR and stress testing systems [68].

Studies can explore how these technologies can be leveraged to enhance data processing, risk modeling, and performance optimization techniques [69].

2. Investigating the performance implications of evolving regulatory requirements

As regulatory requirements for CCAR and stress testing continue to evolve, future research can examine the performance implications of these changes [70].

Studies can analyze the impact of new regulatory guidelines, data requirements, and reporting standards on system performance and propose strategies for adaptation [71].

3. Developing industry-specific performance benchmarks and best practices

Future research can focus on developing industry-specific performance benchmarks and best practices for CCAR and stress testing software [72].

Collaborative efforts among financial institutions, technology vendors, and research organizations can contribute to the establishment of standardized performance metrics and guidelines [73].

C. Concluding Remarks

1. The critical role of performance testing in ensuring the reliability and compliance of CCAR and stress testing processes

Performance testing plays a critical role in ensuring the reliability, scalability, and compliance of CCAR and stress testing processes [74].

Financial institutions must prioritize performance testing as an integral part of their risk management and regulatory compliance strategies [75].

2. The need for continuous improvement and adaptation in a dynamic regulatory landscape

The regulatory landscape for CCAR and stress testing is constantly evolving, requiring financial institutions to continuously improve and adapt their performance testing practices [76].

Embracing a culture of continuous improvement, staying updated with industry best practices, and investing in performance testing capabilities are essential for long-term success [77].

3. The importance of collaboration and knowledge sharing within the financial industry

Collaboration and knowledge sharing among financial institutions, regulators, and technology partners are crucial for advancing performance testing practices in the industry [78].

By fostering a community of practice, sharing experiences, and collaborating on research and development efforts, the financial industry can collectively enhance the performance and resilience of CCAR and stress testing systems [79].

Acknowledgment

The author would like to express gratitude to the financial institutions, regulatory bodies, and subject matter experts who contributed their valuable insights and expertise to this research. Their collaborative efforts and shared experiences have significantly enriched the findings and recommendations presented in this paper.

REFERENCES

[1]. Basel Committee on Banking Supervision, "Stress testing principles," Bank for International Settlements, 2018.

[2]. Federal Reserve System, "Comprehensive Capital Analysis and Review Summary Instructions," Board of Governors of the Federal Reserve System,

[3]. R. Bookstaber, J. Cetina, G. Feldberg, M. Flood, and P. Glasserman, "Stress tests to promote financial stability: Assessing progress and looking to the future," Journal of Risk Management in Financial Institutions, vol. 7, no. 1, pp. 16-25, 2014.

[4]. S. Inanoglu, S. Jacobs Jr., and M. S. Liu, "Modeling methodology and framework for stress testing and capital planning," Journal of Risk Management in Financial Institutions, vol. 11, no. 3, pp. 244-259, 2018.
[5]. M. Rodriguez-Fernandez, "Stress testing for financial institutions: A comprehensive approach," Journal of Financial Transformation, vol. 41, pp. 8-16, 2015.
[6]. Basel Committee on Banking Supervision, "Principles for effective risk data aggregation and risk reporting," Bank for International Settlements, 2013.
[7]. M. K. Brunnermeier, "Deciphering the liquidity and credit crunch 2007-2008," Journal of Economic Perspectives, vol. 23, no. 1, pp. 77-100, 2009.
[8]. S. Quaglietta and G. Szegö, "Risk measures and stress testing for financial stability: A critical review," Journal of Risk Management in Financial Institutions, vol. 8, no. 3, pp. 238-257, 2015.
[9]. T. Beutel, S. Huth, and C. Schäfer, "Data quality for stress testing: Challenges and best practices," Journal of Risk Management in Financial Institutions, vol. 11.
[10]. M. Flood, J. Katz, S. Ong, and A. Smith, "Cyber risk and the U.S. financial system: A pre-mortem analysis," Journal of Financial Economics, vol. 139, no. 2, pp. 1-27, 2021.
[11]. P. Glasserman, C. Kang, and W. Kang, "Stress scenario selection by empirical likelihood," Quantitative Finance, vol. 15, no. 1, pp. 25-41, 2015.
[12]. D. Duffie, "Prone to fail: The pre-crisis financial system," Journal of Economic Perspectives, vol. 33, no. 1, pp. 81-106, 2019.
[13]. M. Bardoscia, S. Battiston, F. Caccioli, and G. Caldarelli, "Pathways towards instability in financial networks," Nature Communications, vol. 8, no. 1, pp. 1-7, 2017.
[14]. V. Acharya, R. Engle, and M. Richardson, "Capital shortfall: A new approach to ranking and regulating systemic risks," American Economic Review, vol. 102, no. 3, pp. 59-64, 2012.
[15]. T. Adrian and M. K. Brunnermeier, "CoVaR," American Economic Review, vol. 106, no. 7, pp. 1705-1741, 2016.
[16]. T. Huber, "Performance testing in Scrum projects," in Agile Testing Days, 2010.
[17]. S. Battiston, J. D. Farmer, A. Flache, D. Garlaschelli, A. G. Haldane, H. Heesterbeek, C. Hommes, C. Jaeger, R. May, and M. Scheffer, "Complexity theory and financial regulation," Science, vol. 351, no. 6275, pp. 818-819, 2016.
[18]. J. A. Whittaker and J. H. Poore, "Markov analysis of software specifications," ACM Transactions on Software Engineering and Methodology, vol. 2, no. 1, pp. 93-106, 1993.
[19]. C. Minoiu, C. Kang, V. Subrahmanian, and A. Berea, "Does financial connectedness predict crises?," Quantitative Finance, vol. 15, no. 4, pp. 607-624, 2015.
[20]. L. Eisenberg and T. H. Noe, "Systemic risk in financial systems," Management Science, vol. 47, no. 2, pp. 236-249, 2001.
[21]. P. Gai and S. Kapadia, "Contagion in financial networks," Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, vol. 466, no. 2120, pp. 2401-2423, 2010.
[22]. R. M. May and N. Arinaminpathy, "Systemic risk: The dynamics of model banking systems," Journal of the Royal Society Interface, vol. 7, no. 46, pp. 823-838, 2009.
[23]. D. Acemoğlu, A. Ozdaglar, and A. Tahbaz-Salehi, "Systemic risk and stability in financial networks," American Economic Review, vol. 105, no. 2, pp. 564-608, 2015.
[24]. B. Schwendimann, "The need for performance testing during system integration and release," in Proceedings of the 1st European Workshop on Testing in Model-Driven Engineering (TMIC), pp. 16-21, 2006.
[25]. T. Liang, J. Ma, and Y. Guo, "Design and implementation of a performance testing tool," in 2010 International Conference on Computational Intelligence and Software Engineering, pp. 1-4, IEEE, 2010.
[26]. J. Zhang, S. G. Elbaum, and M. B. Dwyer, "Automatic generation of load tests," in 2011 26th IEEE/ACM International Conference on Automated Software Engineering, pp. 43-52, IEEE, 2011.
[27]. A. Avritzer, A. B. Bondi, M. Grottke, K. S. Trivedi, and E. J. Weyuker, "Performance assurance via software rejuvenation: Monitoring, statistics and algorithms," in International Conference on Dependable Systems and Networks, pp. 435-444, IEEE, 2006.

[28]. E. Weyuker, T. Ostrand, J. Brophy, and R. Prasad, "Clearing a career path for software testers," IEEE Software, vol. 17, no. 2, pp. 76-82, 2000.
[29]. E. J. Weyuker and F. I. Vokolos, "Experience with performance testing of software systems: Issues, an approach, and case study," IEEE Transactions on Software Engineering, vol. 26, no. 12, pp. 1147-1156, 2000.
[30]. F. I. Vokolos and E. J. Weyuker, "Performance testing of software systems," in Proceedings of the 1st International Workshop on Software and Performance, pp. 80-87, ACM, 1998.
[31]. J. Rudd, K. Stern, and S. Isensee, "Low vs. high-fidelity prototyping debate," Interactions, vol. 3, no. 1, pp. 76-85, 1996.
[32]. E. Dustin, J. Rashka, and J. Paul, "Automated Software Testing: Introduction, Management, and Performance," Addison-Wesley Professional, 1999.
[33]. T. Huber, "Performance testing in Scrum projects," in Agile Testing Days, 2010.
[34]. E. Fourneret, F. Bouquet, F. Dadeau, and S. Debricon, "Selective test generation method for evolving critical systems," in 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, pp. 125-134, IEEE, 2011.
[35]. C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "DevOps," IEEE Software, vol. 33, no. 3, pp. 94-100, 2016.
[36]. G. J. Myers, C. Sandler, and T. Badgett, "The Art of Software Testing," John Wiley & Sons, 2011.
[37]. R. S. Pressman, "Software Engineering: A Practitioner's Approach," Palgrave Macmillan, 2005.
[38]. G. Denaro, A. Polini, and W. Emmerich, "Early performance testing of distributed software applications," in Proceedings of the 4th International Workshop on Software and Performance, pp. 94-103, ACM, 2004.
[39]. J. Z. Gao, J. Tsao, Y. Wu, and T. H.-S. Jacob, "Testing and Quality Assurance for Component-Based Software," Artech House, 2003.
[40]. A. M. Memon, "Automatically repairing event sequence-based GUI test suites for regression testing," ACM Transactions on Software Engineering and Methodology, vol. 18, no. 2, pp. 1-36, 2008.
[41]. J. A. Whittaker and M. G. Thomason, "A Markov chain model for statistical software testing," IEEE Transactions on Software Engineering, vol. 20, no. 10, pp. 812-824, 1994.
[42]. J. A. Whittaker and J. H. Poore, "Markov analysis of software specifications," ACM Transactions on Software Engineering and Methodology, vol. 2, no. 1, pp. 93-106, 1993.
[43]. R. C. Martin, "Agile Software Development: Principles, Patterns, and Practices," Prentice Hall, 2002.
[44]. A. Cockburn, "Agile Software Development: The Cooperative Game," Addison-Wesley Professional, 2006.
[45]. F. Maurer and S. Martel, "Extreme programming: Rapid development for web-based applications," IEEE Internet Computing, vol. 6, no. 1, pp. 86-90, 2002.
[46]. W. E. Lewis, "Software Testing and Continuous Quality Improvement," Auerbach Publications, 2017.
[47]. L. Crispin and J. Gregory, "Agile Testing: A Practical Guide for Testers and Agile Teams," Addison-Wesley Professional, 2009.
[48]. P. Duvall, S. M. Matyas, and A. Glover, "Continuous Integration: Improving Software Quality and Reducing Risk," Addison-Wesley Professional, 2007.
[49]. M. Fewster and D. Graham, "Software Test Automation: Effective Use of Test Execution Tools," Addison-Wesley Professional, 1999.
[50]. J. Bach, "Test automation snake oil," in Proceedings of the 14th International Conference and Exposition on Testing Computer Software, 1999.
[51]. G. Candea, S. Bucur, and C. Zamfir, "Automated software testing as a service," in Proceedings of the 1st ACM Symposium on Cloud Computing, pp. 155-160, ACM, 2010.
[52]. J. Gao, X. Bai, W.-T. Tsai, and T. Uehara, "Testing as a service (TaaS) on clouds," in 2013 IEEE 7th International Symposium on Service Oriented System Engineering, pp. 212-223, IEEE, 2013.
[53]. A. Avritzer, V. Ferme, A. Janes, B. Russo, A. van Hoorn, H. Schulz, D. S. Menasché, and V. Rufino, "Scalability assessment of microservice architecture deployment configurations: A domain-based approach leveraging operational profiles and load tests," Journal of Systems and Software, vol. 165,

[54]. H. Chen, R. H. Chiang, and V. C. Storey, "Business intelligence and analytics: From big data to big impact," MIS Quarterly, vol. 36, no. 4, pp. 1165-1188, 2012.
[55]. D. Graham, E. Van Veenendaal, and I. Evans, "Foundations of Software Testing: ISTQB Certification," Cengage Learning EMEA, 2008.
[56]. J. Humble and D. Farley, "Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation," Addison-Wesley Professional, 2010.
[57]. W. Shewhart, "Economic Control of Quality of Manufactured Product," Martino Publishing, 2015.
[58]. C. Jones, "Applied Software Measurement: Global Analysis of Productivity and Quality," McGraw-Hill Education, 2008.
[59]. B. Beizer, "Software System Testing and Quality Assurance," Van Nostrand Reinhold Company, 1984.
[60]. D. Thakkar, S. Hassan, G. Hamann, and P. Flora, "A framework for measurement based performance modeling," in Proceedings of the 7th International Workshop on Software and Performance, pp. 55-66, ACM, 2008.
[61]. F. Abbors, T. Ahmad, D. Truscan, and I. Porres, "Model-based performance testing of web services using probabilistic timed automata," in 2013 10th International Conference on Web Information Systems and Technologies, pp. 31-45, SciTePress, 2013.
[62]. A. Mos and J. Murphy, "Performance management in component-oriented systems using a model driven architecture approach," in Proceedings of the Sixth IEEE International Enterprise Distributed Object Computing Conference, pp. 227-234, IEEE, 2002.
[63]. R. K. Sharma and B. B. Madan, "Finding sets of key classes in object-oriented software using fuzzy clustering," in 2009 International Conference on Availability, Reliability and Security, pp. 413-418, IEEE, 2009.
[64]. P. Runeson, C. Andersson, T. Thelin, A. Andrews, and T. Berling, "What do we know about defect detection methods?," IEEE Software, vol. 23, no. 3, pp. 82-90, 2006.
[65]. S. Amland, "Risk-based testing: Risk analysis fundamentals and metrics for software testing including a financial application case study," Journal of Systems and Software, vol. 53, no. 3, pp. 287-295, 2000.
[66]. B. W. Boehm, J. R. Brown, and M. Lipow, "Quantitative evaluation of software quality," in Proceedings of the 2nd International Conference on Software Engineering, pp. 592-605, IEEE Computer Society Press, 1976.
[67]. M. Hilbert and P. López, "The world's technological capacity to store, communicate, and compute information," Science, vol. 332, no. 6025, pp. 60-65, 2011.
[68]. L. Ciortea, C. Zamfir, S. Bucur, V. Chipounov, and G. Candea, "Cloud9: A software testing service," ACM SIGOPS Operating Systems Review, vol. 43, no. 4, pp. 5-10, 2010.
[69]. C. Murphy, G. Kaiser, L. Hu, and L. Wu, "Properties of machine learning applications for use in metamorphic testing," in 2008 20th International Conference on Software Engineering and Knowledge Engineering, pp. 867-872, Knowledge Systems Institute Graduate School, 2008.
[70]. X. Xie, J. W. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, "Testing and validating machine learning classifiers by metamorphic testing," Journal of Systems and Software, vol. 84, no. 4, pp. 544-558, 2011.
[71]. E. J. Weyuker, "Testing component-based software: A cautionary tale," IEEE Software, vol. 15, no. 5, pp. 54-59, 1998.
[72]. W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur, "Effect of test set minimization on fault detection effectiveness," in Proceedings of the 17th International Conference on Software Engineering, pp. 41-50, ACM, 1995.
[73]. H. Zhu, P. A. Hall, and J. H. May, "Software unit test coverage and adequacy," ACM Computing Surveys, vol. 29, no. 4, pp. 366-427, 1997.
[74]. W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur, "Effect of test set size and block coverage on the fault detection effectiveness," in Proceedings of the 1994 IEEE International Symposium on Software Reliability Engineering, pp. 230-238, IEEE, 1994.
[75]. W. E. Wong, J. R. Horgan, A. P. Mathur, and A. Pasquini, "Test set size minimization and fault detection effectiveness: A case study in a space application," Journal of Systems and Software, vol. 48, no. 2, pp. 79-89, 1999.

[76]. S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: A survey," Software Testing, Verification and Reliability, vol. 22, no. 2, pp. 67-120, 2012.

[77]. H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," Empirical Software Engineering, vol. 10, no. 4, pp. 405-435, 2005.

[78]. C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, "Experimentation in Software Engineering," Springer Science & Business Media, 2012.