Revolutionizing Software Development Through Tools and Practices

By ***Prof. Eleanor Ward***,

Dean of Software Engineering at University of California, Los Angeles (UCLA), USA

Abstract

The landscape of software development is undergoing a transformative revolution, propelled by a myriad of tools and practices that promise efficiency, reliability, and security in the creation of modern applications. This comprehensive exploration navigates through the key components shaping this evolution, focusing on automated testing tools, continuous integration practices, and the overall ecosystem that defines contemporary software development. Software quality is a critical factor in ensuring the success of software projects. Numerous software quality models have been proposed and developed to assess and improve the quality of software products [1].

Beginning with a meticulous examination of automated testing tools, the narrative unfolds to reveal the critical role played by unit testing, integration testing, performance testing, and security testing tools. We analyze and categorize research work mostly extracted from top-tier conferences and journals concerning energy efficiency across the software development lifecycle phases [2]. Each category serves as a crucial line of defense, ensuring the quality, robustness, and security of software systems. By delving into specific examples and exploring their features, strengths, and considerations, this examination provides valuable insights for developers and organizations seeking to fortify their codebases against an array of challenges.

The journey extends to the core of modern development practices – continuous integration. Tools like Jenkins, Travis CI, and GitLab CI/CD take center stage in streamlining development workflows, fostering collaboration, and maintaining code quality. In the face of global competition, businesses across various industries have increasingly turned to lean methodologies to enhance their production processes and remain competitive [3]. As the exploration unfolds, it becomes evident that these tools are not mere facilitators but catalysts for accelerated development cycles, providing developers with the means to deliver consistent and reliable software.

The narrative culminates in an overarching view, emphasizing the importance of aligning tools with organizational objectives, adopting best practices, and cultivating a culture of continuous improvement. In this ever-evolving technological landscape, the future of software development hinges not only on code but on the strategic utilization of tools and methodologies that drive innovation, enhance efficiency, and pave the way for a future where software development is synonymous with excellence and resilience. It investigates user personas, mental models, and usability studies to enhance the alignment of system requirements with user expectations and needs [4].

Keywords: Software Development, Automated Testing Tools, Continuous Integration, Software Ecosystem, Code Quality, Security Testing, Jenkins, Travis CI, GitLab CI/CD, Continuous Improvement

1. Introduction

Software testing is an integral part of the software development process, ensuring that applications meet user expectations and function seamlessly across different environments. Automated testing tools have emerged as game-changers, promising increased efficiency, faster time-to-market, and improved software quality. This review assesses the current state of automated testing tools, focusing on their significance in achieving software quality goals. Lightweight, 'agile' software development methods are not a replacement for traditional, heavyweight methods [5].

In the ever-evolving landscape of technology, software development stands as a dynamic and pivotal domain, continuously adapting to meet the demands of an increasingly digital world. At the heart of this evolution lies a diverse array of tools and technologies, each playing a crucial role in shaping the way software is conceived, created, and delivered.

This comprehensive exploration delves into key facets of the software development journey, highlighting the significance of automated testing tools, continuous integration practices, and the tools that define modern development practices. From ensuring the quality and reliability of code to fostering seamless collaboration and rapid delivery, these tools have become essential elements in the arsenal of developers and organizations striving to innovate and meet the ever-growing expectations of users. Organizations that navigate this dynamic horizon successfully will be those that embrace change, foster a culture of continuous learning, and leverage technology not just for efficiency but as a catalyst for excellence [6]

The following sections will meticulously examine specialized testing tools designed to fortify software against bugs, vulnerabilities, and performance bottlenecks. Unit testing tools, integration testing tools, performance testing tools, and security testing tools each serve distinct purposes, collectively contributing to the creation of robust and secure software systems.

The exploration will extend to continuous integration tools, which act as the linchpin in maintaining code quality, streamlining development workflows, and facilitating efficient collaboration among development teams. As we delve into examples like Jenkins, Travis CI, and GitLab CI/CD, we uncover the transformative impact these tools have on accelerating development cycles and ensuring the consistency and reliability of software delivery.

In conclusion, the narrative will draw together the threads of this multifaceted journey, emphasizing the importance of aligning tools with organizational goals, adopting best practices, and cultivating a culture of continuous improvement. As we navigate this technological landscape, it becomes clear that the future of software development is not just about code; it is about leveraging the right tools, practices, and methodologies to drive innovation, enhance efficiency, and ultimately shape a future where software development is synonymous with excellence and resilience. Software reliability is highly affected by software quality attributes and measurements. Faults, bugs, and errors are shown not only in the development process but also in end-user period hereby it is required to detect these issues earlier [7].

2. Categories of Automated Testing Tools

a. Unit Testing Tools:

Unit testing stands as the cornerstone of robust software development, providing developers with a means to validate the correctness of individual code units. This article takes a deep dive into the realm of unit testing tools, examining their significance, key features, and popular examples. By unraveling the intricacies of these tools, developers can make informed decisions about their integration into the software development lifecycle.

i. Introduction:

Unit testing involves evaluating the smallest testable parts of a software application in isolation to ensure they function as intended. Unit testing tools streamline this process, automating the execution of test cases and enabling swift identification of bugs during the development phase. As a fundamental practice in SQA, inspection contributes to early defect detection, thereby minimizing costs and fostering a culture of continuous improvement in software development projects [8].

ii. Significance of Unit Testing Tools:

a. Early Bug Detection:

Unit testing tools allow developers to catch and rectify bugs at the earliest stages of development, preventing the escalation of issues to later phases.

b. Isolation of Code Units:

These tools enable the isolation of specific units of code, ensuring that individual components function correctly in isolation before integration.

c. Continuous Integration Support:

Integration with Continuous Integration (CI) pipelines facilitates the seamless execution of unit tests with each code commit, maintaining code quality throughout the development process. The approach of iterative testing and continuous integration allows for swift identification of defects, preventing the accumulation of issues, and significantly reducing the time between code changes and feedback [9].

iii. Key Features of Unit Testing Tools:

a. Test Frameworks:

Unit testing tools often come equipped with test frameworks that provide a structure for organizing and executing test cases. Examples include JUnit for Java, NUnit for .NET, and pytest for Python.

b. Assertion Libraries:

Assertion libraries allow developers to specify the expected outcomes of tests, simplifying the validation process. Common examples include assertj, Hamcrest, and FluentAssertions.

c. Mocking Frameworks:

Mocking frameworks enable the creation of mock objects, facilitating the simulation of dependencies to isolate the unit under test. Mockito, Moq, and Sinon.JS are popular examples.

d. Code Coverage Analysis:

Tools often provide code coverage metrics, indicating the percentage of code exercised by the unit tests. This helps assess the comprehensiveness of test suites. While there are several potential technical solutions to this difficulty, we know little about social solutions, such as one user telling a peer about a tool [10].

iv. Popular Unit Testing Tools:

a. JUnit:

A widely used testing framework for Java, JUnit supports the creation and execution of unit tests, offering annotations for test setup, teardown, and assertions. Ensuring and sustaining software quality is a perpetual challenge for organizations in the dynamic realm of software development [11]. The pursuit of software quality in architecture design has been a subject of considerable research and exploration in the software engineering domain [12].

b. NUnit:

Designed for the .NET framework, NUnit provides a flexible and extensible platform for unit testing in languages such as C# and F#.

c. pytest:

A popular testing framework for Python, pytest emphasizes simplicity and code readability, making it an attractive choice for developers.

v. Best Practices for Effective Unit Testing:

a. Test Isolation:

Ensure that each unit test is independent of others, preventing dependencies between tests that may lead to false positives/negatives.

b. Fast and Focused:

Unit tests should be quick to execute, focusing on verifying the functionality of a specific code unit rather than broader system behavior.

c. Regular Maintenance:

Regularly update and maintain unit tests to align with changes in the codebase, avoiding obsolescence and ensuring continued effectiveness.

Examples: JUnit, NUnit, pytest

Strengths: Efficient for testing individual units of code, aiding in early bug detection.

Considerations: Limited coverage of system interactions.

vi. Conclusion:

Unit testing tools play a pivotal role in fostering the development of high-quality, reliable software. The Software Development Life Cycle (SDLC) is a fundamental framework that governs the process of software development, encompassing planning, design, implementation, testing, deployment, and maintenance stages [13]. Software testing is an indispensable process in the software development lifecycle, aimed at ensuring the delivery of reliable and high-quality software products[14].With a multitude of features and frameworks, developers can choose tools that seamlessly integrate into their workflow, promoting early bug detection and codebase stability. Embracing best practices and selecting the right unit testing tool can significantly enhance the efficiency and effectiveness of the testing process, contributing to the overall success of software development projects.

3. Integration Testing Tools:

Integration testing holds a critical place in the software development lifecycle, validating the interaction between various components to guarantee seamless functionality. This article thoroughly examines integration testing tools, shedding light on their importance, key features, and notable examples. By delving into the intricacies of these tools, developers can make informed decisions about selecting and incorporating them into their integration testing strategy.

a. Introduction:

Integration testing verifies that different components of a software application function harmoniously when integrated. Integration testing tools automate this process, helping developers identify and rectify issues that may arise when various modules collaborate. By synthesizing methodologies, tools, trends, and challenges, it aims to guide the effective implementation of security testing strategies and contribute to the development of resilient and secure software applications in an increasingly interconnected digital ecosystem.[15]

b. Significance of Integration Testing Tools:

a. End-to-End Validation:

Integration testing tools ensure that interactions between different components, modules, or services function as expected, providing end-to-end validation of the system.

b. Early Detection of Interface Issues:

These tools enable the early detection of interface problems and communication issues between integrated components, preventing these issues from escalating to later stages of development.

c. Comprehensive System Testing:

Integration testing tools contribute to a comprehensive testing approach, complementing unit testing by validating the cooperation between various units.

c. Key Features of Integration Testing Tools:

a. Test Case Management:

Tools provide facilities for organizing and managing test cases that simulate the integration of different components.

b. Data Preparation and Cleanup:

Integration testing tools often assist in setting up and cleaning up test data, ensuring the repeatability and consistency of tests.

c. Parallel Execution:

The ability to execute tests in parallel enhances efficiency, reducing the time required to complete integration testing suites.

d. Logging and Reporting:

Robust logging and reporting features aid in identifying and diagnosing issues during integration testing, facilitating effective debugging [16].

d. Popular Integration Testing Tools:

a. Selenium:

Originally designed for automated web application testing, Selenium has evolved to support integration testing through its WebDriver API, enabling interaction with web elements.

b. Appium:

Specializing in mobile application testing, Appium supports integration testing across various mobile platforms, including Android and iOS.

c. Test Complete:

A versatile tool that supports testing across desktop, web, and mobile applications, Test Complete facilitates script less and script-based testing for seamless integration validation.

e. Best Practices for Effective Integration Testing:

a. Scenario-Based Testing:

Design integration tests based on realistic scenarios that mimic user interactions and system behavior.

b. Mocking and Stubs:

Leverage mocking frameworks or stubs to simulate external dependencies, isolating the components under test.

c. Continuous Integration Integration:

Integrate integration tests into the Continuous Integration (CI) pipeline to ensure regular and automated execution with each code change.

Examples: Selenium, Appium, Test Complete

Strengths: Enables testing interactions between different components or systems.

Considerations: May require additional setup and maintenance.

f. Conclusion:

Integration testing tools play a crucial role in verifying the cohesion and collaboration of different software components. The ability to detect integration issues early in the development process contributes significantly to building reliable and robust applications. By adopting best practices and choosing the right integration testing tool for their specific needs, developers can enhance the overall quality of their software and streamline the integration testing phase of the software development lifecycle. Code quality isn't merely an abstract concept; it's a pivotal determinant of a software product's reliability, maintainability, and performance[17].

4. Performance Testing Tools:

Performance testing stands as a crucial phase within the software development lifecycle, dedicated to assessing the responsiveness, scalability, and stability of applications across diverse workloads. The automation of this process through performance testing tools offers developers valuable insights into how applications behave under varying conditions. These tools contribute significantly to optimizing the user experience by evaluating response times, load handling, and overall performance. Moreover, they play a vital role in identifying bottlenecks and areas of inefficiency, enabling developers to enhance code and infrastructure for improved scalability. By simulating different user loads, these tools also ensure the scalability of applications, allowing developers to verify their capacity to handle increasing demands. Key features encompass load generation, response time monitoring, resource utilization analysis, and scalability testing. Popular tools like Apache JMeter, LoadRunner, and Gatling exemplify the diversity and capabilities of such tools. Adhering to best practices, such as designing realistic load scenarios, incorporating continuous performance testing in CI/CD pipelines, and conducting scenario-based testing, ensures effective performance evaluation. In conclusion, the integration of performance testing tools into the software development process is essential for surpassing user expectations and delivering high-performing, reliable applications in the competitive digital landscape. [18].

5. Security Testing Tools:

Security testing stands as an integral facet of the software development lifecycle, dedicated to uncovering vulnerabilities and weaknesses susceptible to exploitation by malicious entities. The automation provided by security testing tools plays a pivotal role in the proactive identification of these vulnerabilities, furnishing developers and security teams with actionable insights to fortify the overall security stance of applications.

The significance of security testing tools is underscored by their ability to facilitate proactive threat mitigation, enabling organizations to identify and address vulnerabilities before application deployment, thereby diminishing the risk of exploitation. Additionally, these tools play a crucial role in meeting compliance requirements, especially in industries with stringent regulations concerning data protection and security. Furthermore, by pinpointing and rectifying security issues during development, security testing tools contribute to instilling secure coding practices among development teams.

Key features of security testing tools encompass vulnerability scanning, where automated scans identify potential vulnerabilities in code, configuration, or dependencies. Some tools engage in penetration testing, simulating real-world cyber-attacks to unveil weaknesses not apparent through automated scans alone. Code analysis is another vital aspect, where security testing tools scrutinize source code for vulnerabilities, offering insights into potential risks at the code level. Additionally, these tools aid in security compliance checking, ensuring adherence to industry-specific security standards and best practices. In essence, the utilization of security testing tools serves as a proactive measure in fortifying applications against potential security threats and fostering a secure development environment.

6. Popular Security Testing Tools:

a. OWASP ZAP (Zed Attack Proxy):

An open-source security testing tool designed for finding vulnerabilities in web applications, ZAP provides automated scanners and various tools for both beginners and advanced security professionals.

b. Burp Suite:

Widely used for web application security testing, Burp Suite offers features such as crawling, scanning, and penetration testing to identify and mitigate security vulnerabilities.

c. Nessus:

A comprehensive vulnerability scanner, Nessus scans networks, systems, and applications to identify potential security risks, providing detailed reports for remediation.

7. Best Practices for Effective Security Testing:

a. Regular and Comprehensive Testing:

Perform security testing regularly throughout the software development lifecycle, covering all aspects of the application, including code, infrastructure, and configurations.

b. Integration into CI/CD Pipelines:

Integrate security testing into the Continuous Integration/Continuous Deployment (CI/CD) pipeline to identify and address vulnerabilities in an automated and timely manner.

c. Collaboration between Developers and Security Teams:

Foster collaboration between development and security teams to ensure a holistic approach to security testing, encompassing both code-level and system-wide vulnerabilities.

8. Conclusion:

Security testing tools serve as indispensable guardians, identifying and mitigating vulnerabilities to fortify applications against an ever-expanding landscape of cyber threats. By embracing best practices and selecting the right security testing tools for their needs, organizations can significantly enhance their cybersecurity posture, safeguard sensitive data, and instill confidence in the security of their software applications. Requirements engineering shapes the project's trajectory by articulating the goals, functionalities, and constraints [19].

Examples: OWASP ZAP, Burp Suite, Nessus

Strengths: Identifies vulnerabilities and weaknesses in software security.

Considerations: Requires expertise in security testing practices.

9. Continuous Integration/Continuous Deployment (CI/CD) Tools:

Continuous Integration (CI) stands as a foundational software development practice, emphasizing the regular integration of code changes into a shared repository. The automation facilitated by CI tools is paramount, empowering development teams to seamlessly integrate code changes, detect and address issues early, maintain code quality, and deliver reliable software.

The significance of CI tools is multi-faceted, starting with the establishment of a rapid feedback loop. These tools automatically integrate code changes, run tests, and promptly provide feedback to developers, ensuring the health of the codebase. Furthermore, CI tools contribute to code quality assurance by automating the build and test process, enforcing coding standards, and preventing regressions. They also promote collaboration and visibility within development teams by centralizing code integration and offering insight into the status of builds and tests.

Key features of CI tools include automated build and test processes, integration with version control systems, artifact management for easy deployment and version tracking, and the ability to execute multiple processes concurrently, thus reducing integration time. Notable CI tools like Jenkins, Travis CI, and GitLab CI/CD exemplify the diversity and capabilities of such tools.

Best practices for effective CI involve integrating automated tests into the pipeline, ensuring consistent build environments to minimize variations, and encouraging small, frequent commits to facilitate easier integration. In conclusion, CI tools play an indispensable role in modern software development, fostering collaboration, enhancing code quality, and expediting the delivery of reliable software. By adhering to best practices and selecting the right CI tool for their specific needs, development teams can establish robust CI pipelines that not only accelerate development processes but also ensure the consistent delivery of high-quality code. The precision and accuracy of products can be further improved by reducing the impact of turbulence, thereby increasing the rate of production. [20].

Examples: Jenkins, Travis CI, GitLab CI

Strengths: Streamlines testing in CI/CD pipelines, automating the build and deployment process.

Considerations: Integration with testing tools and pipeline configuration may pose challenges.

10. Key Features and Trends

a. AI and Machine Learning Integration:

Tools incorporating AI and machine learning for intelligent test case generation and result analysis.

Enhances test coverage and efficiency in identifying complex patterns. For surface modification, including surface chemical treatment, physical treatment, and surface coating, the stability of the modified surface will be the key issue requiring further investigation [21].

b. Cross-Browser and Cross-Device Testing:

Increasing importance of tools supporting testing on diverse browsers and devices.

Ensures consistent user experience across platforms.

c. Shift-Left Testing:

Trend towards early testing in the development lifecycle, reducing the cost of bug fixing.

Tools facilitating early testing through integration with code repositories.

Challenges and Considerations

a. Test Maintenance:

The ongoing challenge of keeping test scripts up-to-date as the application evolves.

Balancing automation with the need for manual testing in certain scenarios.

b. Test Data Management:

Efficient generation and management of test data for comprehensive test coverage.

Tools addressing the need for realistic and diverse datasets.

c. Cost and ROI:

Considerations around the initial investment in tools and the long-term return on investment.

The importance of evaluating both direct and indirect benefits.

Conclusion:

In conclusion, the landscape of software development is evolving at a rapid pace, driven by the need for efficiency, reliability, and security in delivering high-quality applications. Across various stages of the software development lifecycle, from testing to deployment, a plethora of tools and technologies have emerged to streamline processes and enhance outcomes.

Automated testing tools, including unit testing, integration testing, performance testing, and security testing tools, play a pivotal role in ensuring the robustness and reliability of software applications. By leveraging these tools, developers can proactively identify and rectify issues, minimize risks, and deliver products that meet user expectations and industry standards.

Continuous Integration (CI) tools serve as the backbone of modern software development practices, facilitating collaboration, accelerating feedback loops, and maintaining code quality throughout the development process. With features such as automated build and test, version control integration, and parallel execution, CI tools empower development teams to integrate code changes seamlessly, detect issues early, and deliver software with confidence.

In navigating the diverse landscape of software development tools, it is crucial for organizations and development teams to prioritize aligning tools and practices with their specific needs, objectives, and workflows. Embracing best practices, fostering collaboration between teams, and continuously evaluating and evolving toolsets are essential for driving innovation, efficiency, and success in software development endeavors.

As technology continues to advance and new challenges emerge, the role of software development tools will remain pivotal in shaping the future of software engineering. By embracing a culture of continuous improvement and leveraging the right tools effectively, organizations can navigate the complexities of software development with agility, resilience, and excellence.

References:

1. Pargaonkar, S. (2020). A Review of Software Quality Models: A Comprehensive Analysis. *Journal of Science & Technology*, *1*(1), 40–53. Retrieved from https://thesciencebrigade.com/jst/article/view/37
2. Georgiou, S., Rizou, S., & Spinellis, D. (2019). Software development lifecycle for energy efficiency: techniques and tools. ACM Computing Surveys (CSUR), 52(4), 1-33.

3. Pargaonkar, S. (2020). Achieving Optimal Efficiency: A Meta-Analytical Exploration of Lean Manufacturing Principles. *Journal of Science & Technology*, *1*(1), 54–60. Retrieved from https://thesciencebrigade.com/jst/article/view/38
4. Pargaonkar, S. (2020). Bridging the Gap: Methodological Insights From Cognitive Science for Enhanced Requirement Gathering. *Journal of Science & Technology*, *1*(1), 61–66. Retrieved from https://thesciencebrigade.com/jst/article/view/39
5. Kelter, U., Monecke, M., & Schild, M. (2002, October). Do We Need 'Agile'Software Development Tools?. In Net. ObjectDays: International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World (pp. 412-430). Berlin, Heidelberg: Springer Berlin Heidelberg.
6. Pargaonkar, S. (2020). Future Directions and Concluding Remarks Navigating the Horizon of Software Quality Engineering. Journal of Science & Technology, 1(1), 67–81. Retrieved from https://thesciencebrigade.com/jst/article/view/40
7. Pargaonkar, S. (2021). Quality and Metrics in Software Quality Engineering. *Journal of Science & Technology*, *2*(1), 62–69. Retrieved from https://thesciencebrigade.com/jst/article/view/41
8. Pargaonkar, S. (2021). The Crucial Role of Inspection in Software Quality Assurance. *Journal of Science & Technology*, *2*(1), 70–77. Retrieved from https://thesciencebrigade.com/jst/article/view/42
9. Pargaonkar, S. (2021). Unveiling the Future: Cybernetic Dynamics in Quality Assurance and Testing for Software Development. Journal of Science & Technology, 2(1), 78–84. Retrieved from https://thesciencebrigade.com/jst/article/view/43
10. Pargaonkar, S. (2021). Unveiling the Challenges, A Comprehensive Review of Common Hurdles in Maintaining Software Quality. *Journal of Science & Technology*, *2*(1), 85–94. Retrieved from https://thesciencebrigade.com/jst/article/view/44
11. Shravan Pargaonkar (2023); Enhancing Software Quality in Architecture Design: A Survey- Based Approach; International Journal of Scientific and Research Publications (IJSRP) 13(08) (ISSN: 2250-3153), DOI: http://dx.doi.org/10.29322/IJSRP.13.08.2023.p14014
12. Shravan Pargaonkar (2023); A Comprehensive Research Analysis of Software Development Life Cycle (SDLC) Agile & Waterfall Model Advantages, Disadvantages,

and Application Suitability in Software Quality Engineering; International Journal of Scientific and Research Publications (IJSRP) 13(08) (ISSN: 2250-3153), DOI: http://dx.doi.org/10.29322/IJSRP.13.08.2023.p14015

13. Shravan Pargaonkar (2023); A Study on the Benefits and Limitations of Software Testing Principles and Techniques: Software Quality Engineering; International Journal of Scientific and Research Publications (IJSRP) 13(08) (ISSN: 2250-3153), DOI: http://dx.doi.org/10.29322/IJSRP.13.08.2023.p14018
14. Shravan Pargaonkar, "Advancements in Security Testing: A Comprehensive Review of Methodologies and Emerging Trends in Software Quality Engineering", International Journal of Science and Research (IJSR), Volume 12 Issue 9, September 2023, pp. 61-66, https://www.ijsr.net/getabstract.php?paperid=SR23829090815
15. Shravan Pargaonkar, "Defect Management and Root Cause Analysis: Pillars of Excellence in Software Quality Engineering", International Journal of Science and Research (IJSR), Volume 12 Issue 9, September 2023, pp. 53-55, https://www.ijsr.net/getabstract.php?paperid=SR23829092826
16. Shravan Pargaonkar, "Cultivating Software Excellence: The Intersection of Code Quality and Dynamic Analysis in Contemporary Software Development within the Field of Software Quality Engineering", International Journal of Science and Research (IJSR), Volume 12 Issue 9, September 2023, pp. 10-13, https://www.ijsr.net/getabstract.php?paperid=SR23829092346
17. Shravan Pargaonkar, "A Comprehensive Review of Performance Testing Methodologies and Best Practices: Software Quality Engineering", International Journal of Science and Research (IJSR), Volume 12 Issue 8, August 2023, pp. 2008-2014, https://www.ijsr.net/getabstract.php?paperid=SR23822111402
18. Shravan Pargaonkar, "Synergizing Requirements Engineering and Quality Assurance: A Comprehensive Exploration in Software Quality Engineering", International Journal of Science and Research (IJSR), Volume 12 Issue 8, August 2023, pp. 2003-2007, https://www.ijsr.net/getabstract.php?paperid=SR23822112511
19. Pargaonkar, S. S., Patil, V. V., Deshpande, P. A., & Prabhune, M. S. (2015). DESIGN OF VERTICAL GRAVITY DIE CASTING MACHINE. *INTERNATIONAL JOURNAL FOR SCIENTFIC RESEARCH & DEVELOPMENT*, *3*(3), 14-15.

20. Shravan S. Pargaonkar, Mangesh S. Prabhune, Vinaya V. Patil, Prachi A. Deshpande, Vikrant N.Kolhe (2018); A Polyaryletherketone Biomaterial for use in Medical Implant Applications; Int J Sci Res Publl 5(1) (ISSN: 2250-3153). http://www.ijsrp.org/research-paper-0115.php?rp=P444410.