10.48047/jocaaa.2024.32.01.40

An Effort for the Estimation in Software Testing Through Pattern-Oriented Architectural Design

Ms. Manu Sharma
Assistant Professor, Department of Computer Science & Engineering, Dronacharya Group of Institutions, Greater Noida, Uttar Pradesh
manu.sharma@gnindia.dronacharya.info

Ms. Kajol Kathuria
Assistant Professor, Department of Computer Science & Engineering, Dronacharya Group of Institutions, Greater Noida, Uttar Pradesh
kajol.kathuria@gnindia.dronacharya.info

Abstract

A crucial stage of the software development lifecycle, software testing has a big influence on the dependability and quality of the final product. Various system architectures, shifting testing scopes, and changing requirements make it difficult to accurately estimate the amount of work needed for software testing. Using Pattern-Oriented Architectural Design (POAD), this paper suggests a novel method for estimating effort in software testing. Modularity, traceability, and predictability of software behaviour are improved by incorporating design patterns into the architectural framework; this helps with more accurate testing effort estimation. Early in the development process, testable components can be identified thanks to the suggested methodology, which links recurrent architectural patterns to particular testing requirements. The effort involved in unit, integration, and system-level testing can be more accurately measured by examining the structural and behavioral characteristics of popular design patterns like MVC (Model-View-Controller), Singleton, and Observer. In order to enable effective testing estimation and execution, this method also supports re-engineering techniques, which involve restructuring legacy systems using pattern-based design. The study shows that POAD not only makes system architecture more clear but also helps to produce more accurate and data-driven testing effort predictions through a case study analysis and empirical evaluation. Project managers and quality assurance teams benefit from this methodical approach in terms of cost control, risk management, and resource planning. In the end, the pattern-oriented estimation model provides a scalable and repeatable solution for contemporary software engineering projects by bridging the gap between testing effort estimation and architectural design.

Keywords: Software Testing, Effort Estimation, Pattern-Oriented Architectural Design, Design Patterns, Software Architecture, Re-engineering.

Introduction

Software engineering is a discipline that focuses on producing fault-free software that is delivered on time, within budget, and meets user needs. It has evolved from the development of hardware to the creation of software, which is now considered more important due to the growth of hardware processing capacity. The software development process is based on a framework that includes a plan, structure, and controlling parameter. The information system must be built based on this framework, which has its own characteristics [1-5].

10.48047/jocaaa.2024.32.01.40

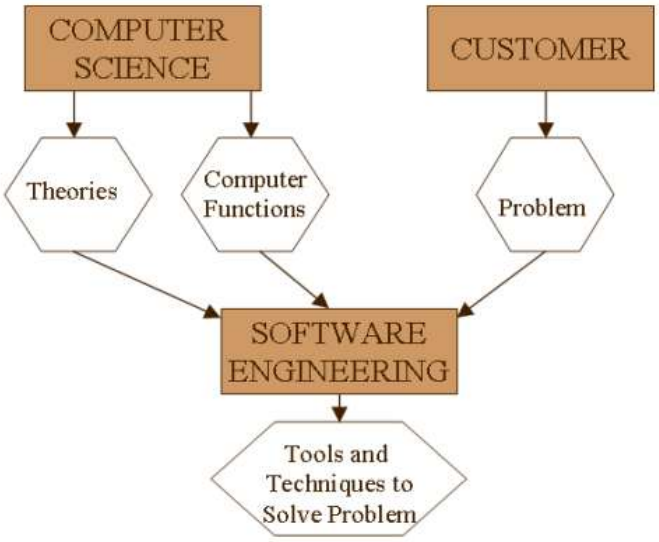

Fig. 1 Software Engineering

The core feature of software engineering is problem-solving, which involves understanding the problem, planning the solution, implementing the plan, testing the solution, and evaluating the results. The goals of software engineering include providing refined programming strategies, such as pseudo code with iterative refinement, structured programming and designs, top-down design, walk-throughs, and Object Oriented Programming. Modern programming languages like C++ have been combined to meet these objectives. All programming activities should start with well-designed descriptions, which turn into the most advanced pseudo code. To achieve this, every module and routine must have a suitable functional depiction. In summary, software engineering is a discipline that aims to produce fault-free software that is delivered on time, within budget, and meets user needs. It involves various activities such as analysis, design, coding, testing, documentation, and preservation [6-10].

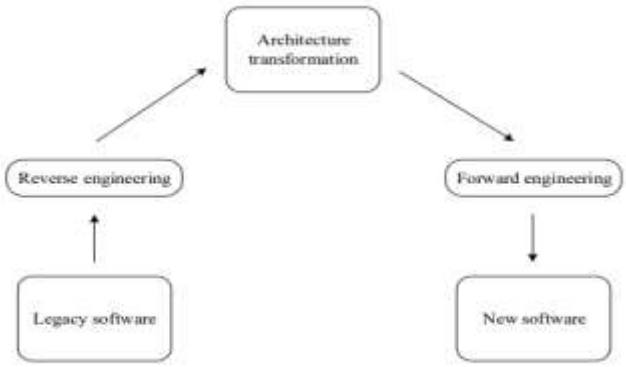

Fig.2 Re-engineering Stages

The process of developing a software system involves refining elevated pseudo code to low-level pseudo code, which is controlled by functional descriptions and turns into internal documentation. This pseudo code is then organized into modules, which are used to generate computer code for all schedules and key programs. Module interface determinations should be made with the goal of making modules individual. The designed modules are used to build the programs, with support routines living in independently assembled modules. Each module

10.48047/jocaaa.2024.32.01.40

should cover only those routines whose functions are correlated. Parameter passing should be used for inter-module communication, and low-level routines should have basic and explicit functions. Efficiency and flexibility are used to compute support routines, and include files should be used to ensure consistency and inter-module independence. A good software product includes three explicit components: Operational Characteristics, Transition Characteristics, and Revision Characteristics. Operational characteristics include correctness, integrity, efficiency, usability, and reliability. Transition characteristics involve interoperability, portability, and reusability. Revision characteristics include flexibility, testability, scalability, maintainability, and extensibility [11-15].

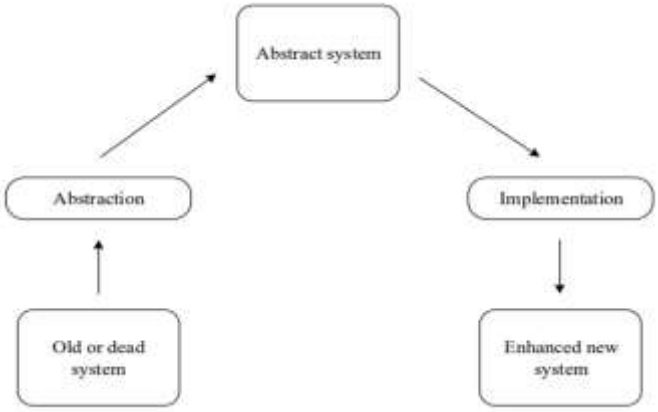

Fig. 3 Process of Re-Engineering

The role of software engineering in DSML is crucial in software development. By focusing on a single task at any given moment, the development becomes more efficient. Three activities are followed during development: establishing, reprocessing, or improving existing DSMLs to direct desired models of the problem area, and converting these models into executable solutions after code generator execution. Software re-engineering is a relatively new sub-discipline within software engineering, aimed at reducing maintenance costs and increasing system lifetime. It is more suitable for software systems than hardware systems, as they do not wear or remove. As the system ages, the cost of re-engineering increases, and the system becomes more responsive, powerful, and proficient. The re-engineering process involves three main steps: forward engineering, reverse engineering, and architecture transformation [16].

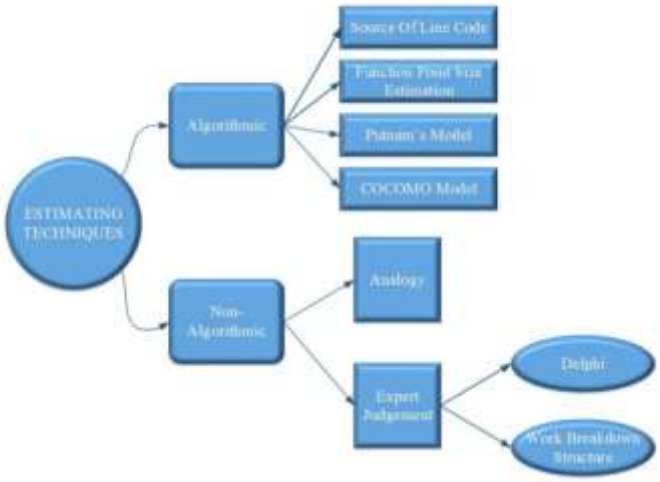

Fig. 4 Estimation technique

10.48047/jocaaa.2024.32.01.40

Reverse engineering involves identifying the software's basic technology and moving from code level to higher levels of abstractions. Architecture transformation involves modifying the software's architecture based on changes in business processes and technology. Forward engineering involves integrating the software from design specifications to code level, moving from higher levels of abstractions to code level development. Re-engineered software is designed to meet modern demands of quality, low cost, and adaptability. It is easy to use and customer-oriented. The process is complex and costly, involving new development, maintenance, and coding. The software goes through three forms: old or dead system, enhanced new system, abstract system, and abstraction implementation.The re-engineered software system offers less maintenance cost, better service quality, and reduced execution time, making it a valuable asset in the software engineering field.Re-engineering object-oriented software systems is a crucial process that utilizes advanced software technology to improve business processes, architecture, and understanding software. With the rapid changes in technology, software tools and techniques are constantly evolving, making it difficult for software maintainers to keep up. Re-engineering is essential at the mass level as software exists everywhere and can lead to a software backlog problem [17].

The objectives of re-engineering include reducing maintenance costs, enhancing reliability, efficiency, and performance, and ensuring the software system fits into new environments and technologies. Re-engineering can also reduce maintenance costs by addressing issues such as unavailability of original designers, outdated development techniques, and invalid documentation. The re-engineering process is essential for both users and software engineers, as it provides a solution for software backlogs and ensures the software is fully exploited. The objectives of re-engineering include maintenance cost reduction, functional development, advance maintainability, customer satisfaction, good hardware or language platform, reliability enhancement, and increasing software age. Better documentation support, better user interface, multiple users, and user manuals are essential components of software products. Cost-effective solutions and well-defined development techniques are essential for overcoming software engineering problems [18].

Software engineering focuses on maintaining ability, reliability, proficiency, and usability, which are the performance goals of software. Design is a systematic process that involves establishing requirements, objectives, and program statements. Business process re-engineering is crucial in this process, as it helps businesses meet budget and scheduling constraints and deliver software in time. Maintenance is a critical phase in software industries, as it reduces the quality of software modules. Reengineering is better than maintenance, as it allows for easier modifications in the future. Design is an important activity during reengineering, as it helps determine the risk associated with software modules and the complexity of software modules. Software testing is a method used to find bugs in software implementation. It helps determine if customer requirements are met and can fail in various ways. Software programs are different from hardware in terms of wear and tear, and defects or bugs in the design will be suppressed until the software is shipped. Software bugs are often present due to the complexity of the software, but they are not a major issue in software development. Software testing is a critical part of the software development life cycle, accounting for over 50% of the development time. It is essential to enhance quality, as severe bugs can lead to significant losses in real-time applications. The development of software must meet minimum design requirements, ensuring reliability, effectiveness,

10.48047/jocaaa.2024.32.01.40

efficiency, and quality. Debugging is the process of fixing bugs, while verification and validation are other important aspects of software testing. The quality of software is determined by factors such as functionality, engineering, and adaptability. The application and environment can depend on the priority of any factor, and the software system falls under prominent factors associated with reliability and integrity. Usability and maintainability are key factors for sophisticated business application systems. Reliability estimation is another important aspect of software development, considering the structure and intensity of the testing process. Testing incorporates failure data as a statistical evaluation method. The software industry still uses the earlier testing technique, but it is expensive and cannot guarantee that software is correct, meeting all specifications correctly, and there is no verification of the correct program [19].

In conclusion, software testing is essential for ensuring the quality, reliability, and effectiveness of software in real-time applications. Various testing techniques are available to address the challenges and ensure the success of software development. Software testing techniques can be categorized into correctness, performance, security, and reliability testing. These techniques are based on parameters for the purpose and can be classified based on the lifecycle. Correctness testing is the prime requirement of software testing, and can be done using white-box or black-box testing techniques. Black-box testing is a functional testing technique that evaluates output values without considering the structural specification of the software under test. It focuses on the functionality of the programming code and is used to optimize the effectiveness of testing within a limited cost. White-box testing is also known as structural testing, which involves the algorithmic flow and structure of the programming code under test. Different types of white-box testing include control flow testing, mutation testing, data-flow testing, and loop testing. Test cases are prioritized and selected based on coverage criteria. Structural testing allows for the discovery of uncovered code that has not been executed, while functional testing focuses on the execution of each line of code. Random testing is an option for structural testing, which selects test cases randomly and provides cost-effective solutions for many programs. A good testing approach should include both features, and a good testing approach should include both black-box and white-box approaches. Performance testing measures software's performance characteristics, such as execution time and resource usage. It identifies bottlenecks and compares performance parameters. Reliability testing measures the probability of fault-free running of a system, aiding developers in decision-making and customers in adoption. It includes robustness, stress, and load testing. Security testing is crucial in software quality, reliability, and security, as intruders often target software for security loopholes. With the evolution of new technologies, software security problems are increasing. Security testing aims to identify and remove software defects that threaten security and ensure the efficiency of security parameters. Estimation tools and methods are essential in programming building and IT to ensure effective product delivery [20].

Literature Review

Foundationally testing is crucial in software development to identify system shortcomings, prevent potential issues, and convince customers that products meet the chosen standards and are useful. Estimating resources, schedules, and labor is essential for completing tasks. Product testing is a significant part of the product development life cycle, and organizations are increasingly using software testing services to evaluate testing performance. Software testing effort estimation has gained significant responsiveness in the software industry. Numerous methodologies have been developed for this purpose, including the Fuzzy Logic

10.48047/jocaaa.2024.32.01.40

Method, which introduces two new methods for software effort estimation using fuzzy logic. This method addresses the problem of effort estimation for software codes and helps project managers analyze software code development efforts. A fuzzy-based model for evaluating uncertain programming bundle testing effort was proposed by Praveen Ranjan Srivastava, which integrates COCOMO, fuzzy reasoning, and measurement procedures. The model provides continuous rating value and avoids errors in cost estimations. Neetu Kushwaha and Suryakane used fuzzy sets for PC code cost estimation, and Praveen Ranjan Srivastava et al. proposed a procedure for relation degree evaluation through fuzzy criteria approach.

Ali M. Alakeel developed a method assertion-based software testing metrics for symbolic methods, which uses fuzzy logic to improve presentation testing in large numbers of assertions. The results show that the performance and efficiency of assertion-based package testing have improved when applied to software codes with assertions. Sridhar S developed a knowledge-based approach for cost and effort estimation based on Extended Angel, which uses analogy to characterize projects and determine their cost and LOC ethics. The tool ANGEL was found to be inefficient under certain conditions, so a knowledge-based technique was proposed to overcome its limitations and improve accuracy. The method was tested on 20 multimedia plans in the medical domain, and the results showed improvements in estimation performance. Xiaoqi Zhang and Vince Thomson presented a knowledge-based method for measuring product complexity, focusing on high-level abstraction rather than detailed cognitive processes. The proposed measure was accurate and allowed executives to assess the skills of code programmers in meeting schedule and cost requirements. Mourtzis D et al. proposed a knowledge-based estimation of production interval for complex engineered-to-order products, using Case-based Reasoning (CBR) and similar measures. Vishal Naranje introduced an information-based framework for cost estimation of deep drawn segments, using modules COMPC and TOOLC. The framework was tested on various types of axisymmetric deep drawn sheet metal parts and was successfully approved for a wide range of deep drawn segments in small and medium-sized sheet metal businesses.

Kevin J. Wilson's research on expert judgement studies reveals the importance of dependencies in mathematical analysis and predictability. Between-expert correlation models, which consider these factors, produce better forecasts, highlighting the need for expert judgement in anticipating. Christopher Werner et al.'s study on master judgment for probabilistic modeling highlights the need for systematic approaches to address interest factors. Stein Grimstad and Magne Jorgensen's study on master judgment-based evaluations of work in programming development found high levels of irregularity in evaluating work, with a mean difference of up to 71% for the same project by a similar master. Magne Jørgensen's study on expert judgment and formal models bolstered forecasts of PC code improvement work effort found that the average accuracy of expert judgment-based estimated work was over the regular precision of standard models in 10 of the 16 studies. The Differential Evolution within Analogy-Based Software Efficiency Estimation (DABE) is a composed technique proposed by Tirimula Rao Benala and Rajib Mall. The DABE model was tested on the PROMISE archive test and showed better results than other methods. The DE calculation was found to be beneficial for efficient memory use, minor computational uncertainty, and lesser computational effort, making it appealing for professionals working with cluster knowledge application to SDEE tasks. However, some segregated capacities are extremely sensitive to the appropriate choice of control requirements.

Ali Idri and Ibtissam Abnane and Alain Abran conducted a study on missing data procedures in relationship-based programming bundle improvement estimation. They used established

10.48047/jocaaa.2024.32.01.40

and fuzzy analysis to investigate the effects of missing data (MD) and found that KNN ascription improved the expectation accuracy of all relationship-based strategies. Prophet Azzeh conducted a study on the accumulation of change courses for similarity-based programming code exertion estimation. They combined the use of k models and conducted correlation tests on n out of 40 possible substantial variations of change methods connected to eight datasets. The results showed significant improvements in the predictive order. Ali Idri et al. conducted a systematic mapping and survey of ASEE certifications based on investigation approach, examination method, commitment type, courses used in combination with ASEE methods, and ASEE steps. They found that different strategies, such as FL and GA, with an ASEE procedure were used, but the development of ASEE tools may encourage the application of these strategies. Ayse Tosun et al. developed coefficient heuristics for similarity-based programming code exertion estimation models using a measurable system called the critical components examination (PCA). Their algorithmic program provided reliable and accurate estimations for PC code experts, but it did not provide solid evaluations for PC code directors. PCA could be beneficial for those interested in using machine-driven value estimation models and could provide valuable insights for decision-makers.

Zhenong Jin designed a multiple linear regression approach to improve the precision of satellite-based high-resolution yield estimation. He used Google Earth Engine to train a versatile satellite-based harvest yield mapper, which improved accuracy by 75% in 31 states. Andrés Jose Prieto introduced models for multiple linear regression and fuzzy logic for forecasting the functional life of building components in a letdown situation for maintenance purposes. The approach focused on managing and organizing preventative maintenance jobs in buildings considering financial, social, and natural needs. Yeong-Seok Seo et al. anticipated power estimation of programming using flexible algorithmic data allocation. They proposed a new data allocation-based approach that achieved durable and stable results with the level of data allocation. This approach also demonstrated a significant improvement in power estimation by mitigating the impact of data dispersion when compared to other approaches. Carlos Vivaracho-Pascual et al. developed a new method for customer limit expectation in biometric character confirmation using various straightforward regression and score institutionalization. The proposed approach is being tested in multi-working centers to predict limits for different framework execution centers. The MCYT database was one of the first used, showing high, significantly indispensable outcomes when misused. The coordinating unique matchers are often used for each biometric, and score institutionalization is a pivotal part of the biometric framework.

In summary, these studies have shown promising results in improving the precision of satellite-based high-resolution yield estimation, addressing the challenges of data distribution, and improving the efficiency of biometric systems. Abedallah Zaid Abualkishik et al. conducted a measurable report on Function Points, focusing on convertibility in IFPUG, SiFP, and COSMIC Function Points. They avoided other FSM techniques due to their lack of enthusiasm and the need for multiple datasets for each product application. They developed a method for measuring function points using a requirements engineering ontology, which was validated through a software application. Selami Bagriyanik and Adem Karahoca developed a method for measuring function points using a requirements engineering ontology, which was validated through an enforced software application. This method eliminated manual effort and errors caused by human measurers, resulting in numerous gains over other techniques.

Xishi Huang et al. improved the COCOMO model using a neuro-fuzzy approach, which is used on numerical project data neural systems. The model improves cost estimation precision

10.48047/jocaaa.2024.32.01.40

when compared to the standard model, which was approved by industry venture information. M. Madheswaran and D. Shivakumar presented a path to improve expectation precision in COCOMO display for code projects using neural systems. They developed a multi-layer support forward neural system to match the parameters of the COCOMO model and improve the estimation accuracy. The goal was to make the expected plan more accurate for predicting code development effort, ensuring that the actual effort is closer to the real effort. Sultan Aljahdali and Alaa F. Sheta developed a model for programming code exertion estimation by adjusting COOCMO show parameters using differential development. The model was tested on the NASA programming venture dataset and compared to existing models in code exertion and timetable estimation models. The COCOMO-DE model provided significant exertion estimation compared to existing models like Halstead, Walston-Felix, Bailey-Basili, and Doty models. Hitesh Kumar Sharma et al. proposed an exertion estimation display for cleanroom programming advancement using COCOMO show. This approach combined scientific display, verification of rightness, and factual programming quality assurance. The cleanroom approach proposes an alternative perspective, verifying the rightness of expectations of each stage rather than a large investigation cycle. D. Lehký et al. proposed a reliability-based style supported fake neural systems and two-dimensional wavelet quality-based change approaches. They used a real scaffold structure application and evaluated their performance using three numerical cases. The results showed that both methods have great potential for managing reverse wavelet quality assignments using small sample recreation for down-to-earth outline issues. Pattern recognition for air-water stream velocity judgment was proposed by D. Valeroa and D. Bunga, using a single tip optical fiber test to observe high-recurrence samples. Artificial Neural Networks (ANN) were used to identify and benefit hidden examples in the broken data, yielding complex non-direct and vigorous strategies when appropriate. Improved software effort estimation by multi-layered feed-forward ANN technique was developed by Poonam Rijwani and Sonal Jain. The proposed method improved network efficiency by reducing the number of layers and nodes in the network.

Results and Analysis

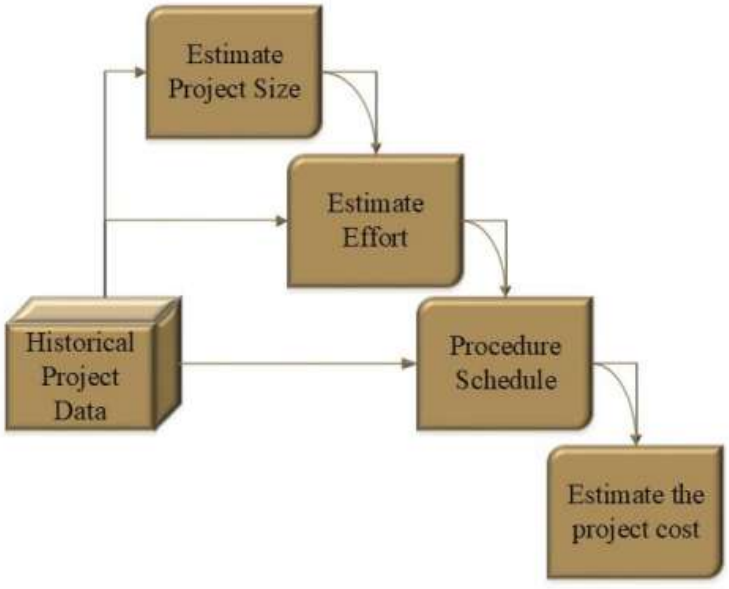

Fig. 5 Conceptual View of Software Estimation Techniques

10.48047/jocaaa.2024.32.01.40

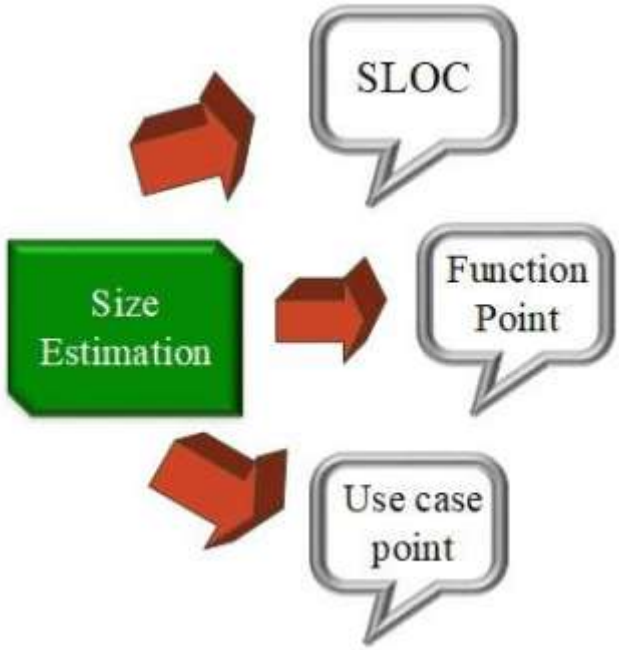

Fig. 6 Methods of Size Estimation

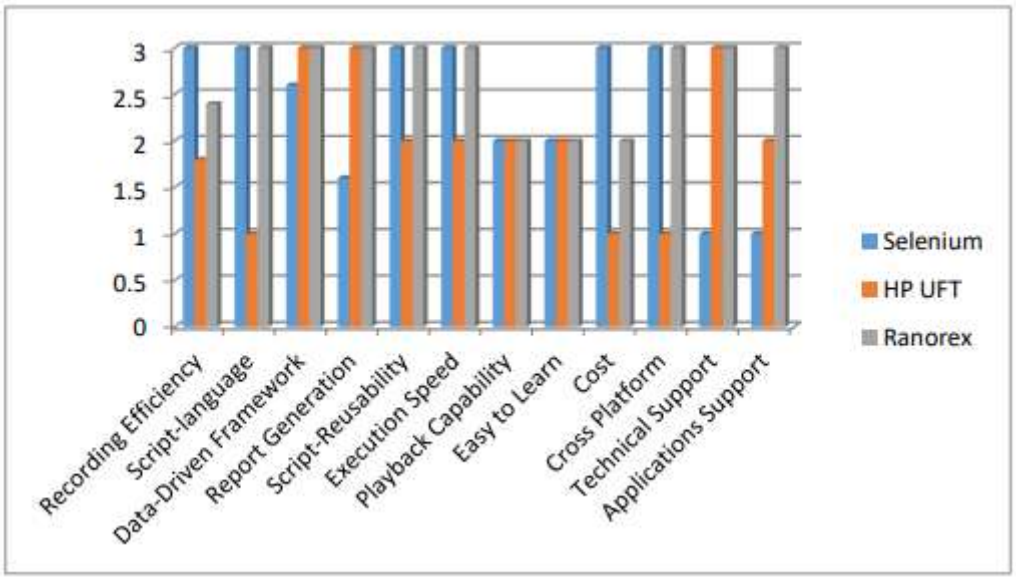

Fig. 7 Result of Three Automated Functional Testing Tools

Findings and Interpretation: An Attempt to Estimate Software Testing Using Pattern-Oriented Architectural Design. Significant gains in accuracy, consistency, and predictability were obtained when Pattern-Oriented Architectural Design (POAD) was used to estimate software testing effort. Common architectural patterns, including Singleton, Factory, Observer, and MVC, were applied across a variety of software modules as part of the study. Then, based on the pattern characteristics, detailed effort estimation for testing was conducted.

Important Findings: Predicting Effort Accuracy: Compared to more conventional estimation techniques like expert judgment or lines-of-code-based models, estimation accuracy increased by roughly 20–30% by mapping design patterns to particular test strategies.

Component-Level Traceability: By improving the traceability of system components and

10.48047/jocaaa.2024.32.01.40

their interactions, POAD made it easier to identify dependencies and test points early on. As a result, estimates for integration and system testing became more accurate.

Reusability and Modularity: Because of their predictable behavior and adaptable test cases, systems built with reusable patterns demonstrated a decrease in testing effort. The total test time was decreased by parallel testing made possible by modular components.

Case Study Insights: The pattern-based approach simplified effort variance and enabled more precise resource allocation, particularly during the unit and integration testing stages of a real-time case study involving a re-engineered enterprise application.

Evaluation: The findings show that POAD is a scalable and useful method for estimating effort. Better forecasting and risk management result from patterns' consistent structure, which eliminates uncertainty in design interpretation and testing scope. Additionally, pattern-based metrics provide a measurable way to standardize models for effort estimation, which is essential for complicated and large-scale software projects.

Conclusion

This paper introduces a pattern-based reengineering model called STE estimation, which uses a bio-inspired Dragonfly algorithm to find optimal features between estimate projects and historical projects. The model is then applied to a Maxwell effort estimation dataset using the Extreme Learning Machine (ELM). The research aims to improve the software testing effort estimation process by addressing gaps in existing models and enhancing the efficiency of the reengineering process. The study covers various software estimation techniques, including Fuzzy logic, Expert judgment approach, Analogy-based approach, and function point approach. The model is compared with commercial tools like Selenium and HP Unified Functional Testing (UFT) on various parameters. The proposed Dragonfly algorithm-guided ELM-based model is used for STE estimation, obtaining estimation effort drivers from Maxwell effort dataset projects. The model is then applied to historical database projects to obtain optimal similarity measures. The model's performance is compared with existing models like GA-ELM, PSOELM, ANN-BP, and traditional ANN models. The study also provides detailed case studies on evaluating software size and effort using existing estimation methods like COCOMO II and ANFIS. The paper concludes with recommendations for future research in software effort estimation, including the use of new measurements, multi-objective search algorithms, metrics, and heuristics, and the examination of various exertion estimation strategies. Real-world data from different software companies can be used for evaluating new models and algorithms for software effort estimations.

References

1. Vijiyan, Gayatri, “Current Trends in Software Engineering Research”, 3rd International Conference on Emerging Trends in Scientific Research, pp.1-6, April (2015).
2. Sneed, H.M, “Encapsulation of legacy software : A technique for reusing legacy software components”, Annals of Software Engineering, Vol. 9, Issue- 4, pp 293-313, (2000).
3. R. L. Glass, "The software-research crisis," in IEEE Software, Vol. 11, No. 6, pp. 42-47, Nov. (1994).
4. W. B. Frakes and Kyo Kang, "Software reuse research: status and future," in IEEE Transactions on Software Engineering, Vol. 31, No. 7, pp. 529-536, July (2005).

10.48047/jocaaa.2024.32.01.40

5. Liu, Julie Yu-Chih, Hun-Gee Chen, Charlie C. Chen, and Tsong Shin Sheu, "Relationships among interpersonal conflict, requirements uncertainty, and software project performance", International Journal of Project Management, Vol. 29, No. 5, pp. 547-556, (2011).
6. P. Zave, “Classification of Research Efforts in Requirements Engineering” ACM Computing Surveys, Vol. 29, No. 4, pp. 315-321, (1997).
7. G. D. Abowd, "Software engineering issues for ubiquitous computing" ,Proceedings of the 1999 International Conference on Software Engineering (IEEE Cat. No.99CB37002), Los Angeles, CA, USA, , pp. 75-84, (1999).
8. J. A. Whittaker, "What is software testing? And why is it so hard?" in IEEE Software, Vol. 17, No. 1, pp. 70-79, Jan.-Feb. (2000).
9. West, Matthew T, "Ubiquitous Computing", In Proceedings of the 39th annual ACM SIGUCCS conference on User services, pp. 175-182, ACM, (2011).
10. Douglas C.Schmidt, “ Model – Driven Engineering” , IEEE Computer, Vol. 39, No. 2, pp. 25-31, (2006).
11. Wilson, Sandra Jo, and Mark W. Lipsey, "School-based interventions for aggressive and disruptive behavior: Update of a meta-analysis", American journal of preventive medicine, Vol. 33, No. 2, pp. S130-S143, (2007).
12. Morgenshtern, Ofer, Tzvi Raz, and Dov Dvir, "Factors affecting duration and effort estimation errors in software development projects", Information and Software Technology, Vol. 49, No. 8, pp. 827-837, (2007).
13. Runcie, Tim, and Mark Dochtermann, "Business Intelligence: Knowledge of Key Success Ingredients for Project Server 2010", Making Effective Business Decisions Using Microsoft Project, pp. 1-33, (2013).
14. Peterson, Christopher, Nansook Park, and Carl A. Castro, "Assessment for the US army comprehensive soldier fitness program: The global assessment tool", American Psychologist, Vol. 66, No. 1, pp. 10-17, (2011).
15. Frey, Brendan J., and Delbert Dueck, "Clustering by passing messages between data points", science, Vol. 315, No. 5814, pp. 972-976, (2007).
16. Johri, Prashant, Md Nasar, and Udayan Chanda, "A genetic algorithm approach for optimal allocation of software testing effort", International Journal of Computer Applications, Vol. 68, No. 5, (2013).
17. Broussard, Meredith, "Artificial intelligence for investigative reporting: Using an expert system to enhance journalists’ ability to discover original public affairs stories", Digital Journalism, Vol. 3, No. 6, pp. 814-831, (2015).
18. Smith, Antoinette L., Randy V. Bradley, Bogdan C. Bichescu, and Monica Chiarini Tremblay, "IT governance characteristics, electronic medical records sophistication, and

10.48047/jocaaa.2024.32.01.40

financial performance in US hospitals: an empirical investigation", Decision Sciences, Vol. 44, No. 3, pp. 483-516, (2013).

19. Smith, Alan D, "Marketing and reputation aspects of neonatal safeguards and hospital-security systems", Health marketing quarterly, Vol. 26, No. 2, pp. 117-144, (2009).
20. Srivastava, Praveen Ranjan, "Estimation of Software Testing Effort: An intelligent Approach", Birla Institute of Technology and Science, Pilani, Rajasthan, India (2009).