Original Article

Advances in software engineering: research boundaries and innovations

Sevda Nasseri Ahmadabad[1]*

[1]Bachelor's degree, computer software engineering, Islamic Azad University, Tabriz branch, Iran.

Correspondence: Sevda Nasseri Ahmadabad, Bachelor's degree, computer software engineering, Islamic Azad University, Tabriz branch, Iran. sevdanaseri1401@gmail.com

ABSTRACT

Like an imperceptible and suspended flood in the depths of the oceans of knowledge and technology, software engineering is known as one of the fundamental and dynamic sciences in today's world, which faces boundless diversity and complexity. Software and related technologies reach new dimensions at a tremendous speed and push the boundaries of research and innovation in this field. The present study aims to investigate the progress of software engineering along with its research boundaries and innovations. The research method is a descriptive-analytical methodology based on library resources. In this study, we cover various topics such as complex software architecture, artificial intelligence-based software development, software modeling and analysis, software security and related ethical issues, and new software development and management methods. Also, by providing innovative and inspiring content and patterns in the research and development process of this field, we provide new tools and perspectives for interested audiences.

Keywords: Software engineering, Agile development, Machine learning, Artificial intelligence

Introduction

Software engineering research, as a significant and outstanding field in computer science and information technology, plays a vital and irreplaceable role in the development and progress of the software industry. Basically, the studies in this field investigate and analyze the many challenges related to the development, testing, quality assurance, management, and evolution of software. Our objective in this research is to provide a comprehensive and in-depth review of software engineering research in such a way that the interested reader gets to know the challenges in the complex and changing world of software and sees them as opportunities for further progress.

In this study, we will explain the basics and main concepts of software engineering and then explore the details of research in this field. We introduce various research methods so that researchers and students can use them in their research.

One of our objectives in this study is to promote a deeper understanding of software engineering research and increase readers' awareness of important and recent issues in this field. We invite you to accompany us on this journey to the novelties and innovations of the world of software technology. We hope that this study will be considered as a valid and useful resource for software engineering research and will help you to improve your skills and knowledge.

Principles and methods of agile development

Principles of agile development

1. Continuous interaction with the customer: Agile development emphasizes continuous interaction with the customer. The software development team should involve the customer in the development process and use customer feedback for improvement (1).

2. Frequent changes in requirements: Agile development has the ability to adapt to frequent changes in requirements. In fact, the project is constantly changing and meeting the requirements of the customer.

3. Iterative development and testing: In agile development, teams iteratively develop new versions of software and run frequent tests to ensure that software quality is maintained.

4. Agile configuration: Cross-functional teams in agile development consist of people with diverse skills and expertise. These teams are collaboratively involved in software development.

Agile development methodologies

1. Scrum

Scrum is one of the most well-known agile development methodologies focused on continuous interaction with the customer, team accountability, and distribution of tasks in short time periods.

2. Kanban

Kanban as a workflow management system helps teams to manage tasks (in order of priority) and resource availability.

3. Extreme programming (XP)

XP is an agile development methodology focused on code quality, automated tests, and pairwise development.

4. Feature-Driven Development (FDD)

FDD is an agile development method focused on features and dividing work into specific time periods.

These principles and methods help development teams to improve their projects and achieve desired results quickly.

Agility in large-scale projects

Agile development is known as a popular method of software development in small and medium projects. Can this approach achieve productivity in large-scale projects with greater complexity? (2) The answer to this question may be associated with various conflicts. However, agile development in large-scale projects is also possible and effective with its own conditions and challenges. Several issues in large-scale projects may challenge agile development, such as the following:

1. Inflexibility in structure: Large projects may have complex structures and processes that increase the difficulty of agile development. These structures may reduce the flexibility of the development team.

2. Resource management: Resource management in large projects is inherently challenging. A large number of existing members and teams can make managing interactions more complex and the project more difficult.

3. Communication: Communication between teams and different members becomes more complex in large projects. This may cause more conflicts and ambiguities.

Nevertheless, agile development in large-scale projects still brings several benefits, such as the following:

1. Continuous interaction with the customer: Focusing on continuous interaction with the customer in large projects can lead to the improvement of needs and response to changes.

2. Flexibility: Agile development still allows teams to adapt to changes and drive development to improve the product.

3. Risk reduction: Iterative processes and tests reduce the risks in large projects.

4. Knowledge sharing: Agile development allows teams to share knowledge and experiences and benefit from continuous learning.

In summary, agile development in large-scale projects can be effective and successful provided that the challenges are properly managed and attention is paid to the benefits and principles of agile development. Flexibility and continuous interaction with the customer are the main strengths of this approach in large projects.

Machine learning and artificial intelligence in software engineering

Machine learning is one of the main branches of artificial intelligence and is based on algorithms and mathematical models that enable computers to be more efficient and make better decisions. These techniques and methods are used in software engineering to resolve various problems including error detection, system performance prediction, optimization, and complex decision-making.

Software quality is one of the critical subjects in software development. Evaluation and prediction of software quality ensures the correct and optimal performance of the software and helps to reduce errors and functional defects. Software quality can help improve the performance of software developers and reduce the cost and time spent on fixing problems. This is of significant importance in software engineering. Ultimately, the ability to predict software quality analysis enables developers to make continuous improvements to their software quality and improve user experience.

Artificial intelligence

Artificial intelligence-based software testing helps ensure software quality and stability. It can also help developers detect and respond to software problems and errors earlier. This section is dedicated to developing more advanced technologies for software testing and improving existing methods in order to achieve higher quality in developed software.

In today's world, technologies based on artificial intelligence and software engineering are developing rapidly and penetrating various fields such as artificial intelligence systems, automation, data analysis, etc. These developments impose new crises and ethical challenges in the path of technological development. Many topics are studied and investigated in the field of artificial intelligence, such as ethical decisions about self-driving cars and work automation, face recognition and people's privacy, the abilities of artificial intelligence to destroy human jobs, and relying on algorithms for various choices and decisions. Consequently, it is important to understand the ethical considerations in the field of artificial intelligence and advanced technologies. The reason is that technologies should be developed in such a way that they serve the interests of humans and society and do not impose negative effects. This section provides technological experts and software developers with ethical guidelines for design decisions and implementation of advanced technologies (3).

Understanding software security

Software security is one of the fundamental and vital matters in the digital age. With the expansion of the use of software and online communication, attention has been focused on software security. Understanding software security helps developers and development teams consider:

1. Detection and understanding of security threats: It is very important to understand security threats and methods of attacks on software. This knowledge helps developers protect their software programs against these threats.

2. Implementing security techniques: Understanding security principles and techniques can help developers take advantage of encryption, authentication, and access control tools and methods.

3. Risk management: Understanding software security helps managers and decision-makers to properly assess security risks and make appropriate decisions about security policies and strategies.

4. Responding to attacks: In the event of security attacks, understanding software security helps security operations teams respond quickly and effectively to security incidents and mitigate the resulting negative impact.

Analysis and reduction of vulnerability

By definition, vulnerabilities are weaknesses in the core of software that can be exploited by attackers to gain access to systems and sensitive information. Therefore, analyzing and mitigating vulnerabilities is very important to prevent security threats.

The following matters should be considered in the process of analyzing and reducing vulnerabilities (4):

1. Identification of vulnerabilities: First of all, the vulnerabilities in the software should be identified. This can be achieved through source code reviews, security tests, and the use of security analysis tools.

2. Prioritization of vulnerabilities: After identification, vulnerabilities should be sorted according to their importance and destructive effects. This prioritization helps security teams focus on the most threatening vulnerabilities.

3. Determining solutions and making decisions: After prioritizing the vulnerabilities, appropriate solutions should be applied to reduce or eliminate them. This can include changes to the source code, adding security layers, and updating the software.

4. Testing and validation: After implementing the solutions, security tests and security analysis should ensure that vulnerabilities are properly fixed and the software is protected against security attacks.

Analyzing and reducing vulnerabilities is an important phase in the software development process and maintaining the security of information and systems. By relying on the right techniques and specialized tools, this phase can help discover and fix vulnerabilities and help improve software security, and protect valuable information.

Safe coding practices

Safe coding methods include the following (5):

1. Valid inputs: Make sure your software accepts user inputs correctly and securely and can prevent malicious code injection. This can be achieved by validating and filtering inputs.

2. Access control: Limiting access to sensitive code and protecting critical resources will deny internal and external attacks.

3. Avoiding obsolete and vulnerable codes: Using safe codes and research related to vulnerable technologies helps to reduce known vulnerabilities in software codes.

4. Logging causes and events: Logging and analyzing security causes and events helps security teams to quickly respond to attacks and analyze security problems.

5. Security testing: regularly conducting security tests to discover and fix vulnerabilities and security weaknesses in software codes.

6. Training: Training developers on secure coding principles and updating them on new security threats and common security issues.

Secure coding methods can help developers strengthen the security of software source codes and protect information and systems against security threats. These principles and methodologies allow developers to consider security in the early stages of software development and prevent security problems from occurring.

Unknown threats and future directions

Software security is a dynamic and complex field that is constantly changing. Security threats are constantly evolving and changing and will appear in new and unknown forms in the future. This section focuses on unknown threats and predictions about future directions in software security. Analyzing and predicting unknown threats and future directions is a vital matter in software security science. These analyses can help security teams and developers cultivate the best solutions and strategies to deal with future threats. For example, future analytics can help detect and predict malicious attacks, develop new defense techniques, and design advanced security systems (6).

Innovations in software testing

Software testing is one of the fundamental steps in the software development process. Considering the complexity and variety of software, the need for innovative methods and improvement of the test process is inevitable. This section deals with matters related to this field as follows (7):

1. New tools for software testing: Introducing new tools for software testing helps developers to perform better and more comprehensive tests.

2. Innovative testing methods: explanation about innovative methods in software testing such as automated testing, performance testing, security testing, etc.

3. User experience improvement techniques: how to utilize user tests and user experience (UX) improvement processes in software development.

4. Experiences and case studies: introducing case studies in the field of projects and companies benefiting from innovations in software testing that have achieved favorable results.

Innovations in software testing enable developers and test teams to improve the testing process and deliver better, more stable software. These concepts and innovations can help developers identify and fix security problems and software bugs and improve the quality and security of software.

Automatic test and continuous test

Automated testing means using computer tools and scripts to run tests and evaluate software quality and performance. Continuous testing is a type of automatic testing that is continuously implemented in the software development process (8).

These two concepts are very important in software development. The following are discussed in this section:

1. Principles of automated testing: explaining the principles and approaches related to performing effective automated testing. These principles include choosing the right tests, creating strong test scripts, and creating the right test platforms.

2. Benefits of continuous testing: explaining the benefits and advantages of continuous automated testing during the software development process. The resulting benefits include rapid identification of problems and bugs, improved code quality, increased software reliability, and increased development speed.
3. Automated test tools: introducing tools and frameworks for running automated tests. These tools can help developers run automated tests more effectively.
4. Success in continuous testing: a case study of companies and projects using continuous testing that has achieved significant success and positive results.

Automated testing and continuous testing are the most important elements in helping developers cultivate high-quality and more stable software. These principles and methods can help software development teams focus on software testing and improving software performance and security.

Model-based test

Model-based testing means using mathematical and conceptual models to evaluate and test software. In this methodology, software models are created to accurately represent the features, functionality, and behavior of the software. Then these models are used to perform tests and evaluate the software (9).

The advantages of model-based testing include the following:

1. High accuracy: the use of accurate and mathematical models leads to higher accuracy in software evaluation.
2. Coverage tests: More comprehensive coverage tests can cover all software components.
3. Easy analysis and interpretation: Mathematical models can help to analyze and interpret test results and provide a more accurate understanding of software problems and bugs.
4. Risk management: the use of model-based testing can facilitate the improvement of risks associated with software development.

This section covers topics such as the principles of creating different types of software models (mathematical, conceptual models, etc.), model-based testing methods, and practical examples of the use of model-based testing in software development.

Model-based testing is an advanced method in software testing that helps developers evaluate and improve their software more accurately. This method facilitates the improvement of the software quality and helps to increase the trust in the software and reduce the risks related to the development.

AI-powered testing

Artificial intelligence (AI) refers to the use of algorithms, models, and data-driven technologies to perform human-like intelligence tasks. In the software testing process, artificial intelligence can be used as a powerful tool to improve software tests and evaluations. This section deals with the following:

1. Automatic detection of bugs: using artificial intelligence to automatically detect bugs and software problems. This includes detecting code errors, security breaches, and performance issues.
2. Automated tests: Using artificial intelligence techniques to create and run automated tests.
3. Improving tests: using artificial intelligence to improve test coverage, increase the accuracy of tests, and detect specific cases in software.
4. Test data analysis: using artificial intelligence techniques to analyze test data and extract useful information.

AI-powered testing helps developers to focus on software testing and evaluation and improve the testing process. Artificial intelligence can help diagnose and fix software problems, reduce testing time and cost, and improve software quality. Therefore, the use of artificial intelligence in software testing is considered one of the most important innovative and efficient methods in this field.

Blockchain and emerging applications

Blockchain is a technology based on digital principles and cryptography that allows the recording and verification of transactions or assets between people without the need for intermediaries. This section explains the basics and architecture of blockchain, transaction verification algorithms, and various applications of blockchain in different industries.

Smart contracts are programs defined on the basis of contractual terms in the blockchain that execute their function. These programs are compiled and executed by cryptography and contract logic. This section explains how to create and use emerging applications, their advantages and challenges (10).

Blockchain is a digital system recording transactions in the form of consecutive blocks. These blocks are interconnected and record transactions securely without the need for intermediaries. The details of this technology are given below:

1. Blockchain structure: Blockchain consists of a chain of consecutive blocks. Each block stores a number of transactions and is related to its previous and next block. This structure provides the possibility of ensuring the authenticity of transactions and the possibility of tracking their history.
2. Cryptography and security: Blockchain uses cryptographic algorithms for security. Transactions are encrypted in blocks and reject unauthorized access.
3. Verification of transactions: Blockchain uses a network of computers or nodes to verify transactions. Transactions are confirmed by network nodes and then added to the block.
4. Applications: Blockchain has many applications in various industries, such as digital money transfer (such as Bitcoin), tracking products in the supply chain, recording digital assets (such as smart contracts), and more.
3. The intersection of blockchain and emerging applications: This section deals with how to use emerging applications in the blockchain and establishes a connection between these two concepts. Emerging applications can be used as execution logic in blockchain contracts and execute the transactions performed under the specified conditions.

Smart contracts and emerging applications

Smart contracts are contracts executed by artificial intelligence and automatic algorithms. These types of contracts are governed by computer codes and artificial intelligence algorithms. Also, transactions and actions are executed automatically.

Research in the field of human-computer interaction (HCI) and software design

This section explains about human-computer interaction, research related to human-computer interaction, and approaches and issues related to software design. The details of this field are provided below (11):

1. Human-computer interaction (HCI): This section deals with the study and research in the field of human-computer interaction. Human-computer interaction examines how humans interact with computer systems and software. Research in this field is dedicated to improving user experience, increasing productivity, and creating better user interfaces.

2. Paying attention to the user's requirements: the research conducted on recognizing the requirements, preferences, and problems of users in using software and user interfaces are examined. This can help improve software design and increase user satisfaction.

3. Interaction between technology and humans: The research conducted on the impact of technology on the lives of humans and society is examined. This can help to better understand the impact of technology in modern societies and create solutions to manage its positive and negative effects.

4. Advances in software design: The research conducted in the field of software design deals with how to improve software design methods and tools, software architectures, and optimization of the software development process.

5. Software development focused on user experience (UX): Research in the field of UX focuses on improving the user experience and designing user interfaces. These studies help to create enjoyable and easily reached software for users.

Research in the field of human-computer interaction and software design aims to improve the user experience, increase productivity, and improve the quality of software. These studies help to create better software products and improve the performance of computer systems.

Research methodologies in the field of HCI

This section describes the methods and techniques used in human-computer interaction research. The details of this issue are given below:

1. Introduction of HCI research methodologies: This section introduces various research methodologies in the field of human-computer interaction, including quantitative methodologies such as statistical analysis, surveys, and controlled experiments, and qualitative methods such as interviews, observations, and content analysis.

2. Comparative research: Comparative research methodologies are used to compare two or more situations or different versions of a software or user interface. This approach can help compare user experiences and possible design improvements.

3. Experimental research: introducing techniques and models related to experimental research investigating and evaluating user experience. This research utilizes user tests, surveys, and empirical measurements.

4. Case studies: introducing case studies and projects benefiting from HCI research methodologies. These studies are considered as models and useful examples for other research in the field of human-computer interaction.

5. The Impact of HCI research: A comprehensive description of the impact of human-computer interaction research on software development and user experience improvement. This can help improve software performance and increase user satisfaction.

HCI research methods are of great importance in developing software and creating improved user interfaces. These methods help researchers and designers make the best decisions for designing and improving software and user experience.

Innovations in human-computer interaction and user experience

This section describes recent innovations and changes in the field of human-computer interaction (HCI) and user experience. These changes and innovations have an important impact on the way software and user interfaces are designed and used. Details are given below:

1. Developments in human-computer interaction: Important developments in the field of human-computer interaction are examined in this section. These developments include changes in user interfaces, methods of user interaction with computers, and other aspects of HCI.

2. Innovations in user experience: Comprehensive explanations about innovations and changes in user experience (UX). These innovations include changes in the design of user interfaces, adding new features, and improving the user experience.

3. The impact of innovations: Investigating the impact of the realized innovations on user satisfaction, improving software efficiency, and improving the quality of user experience. Also, comprehensive explanations about changes in research and design methods in response to these developments are also included.

4. Applications of innovations: introducing applications and projects benefiting from these innovations in human-computer interaction and user experience. These cases are considered useful models and examples for future research and development in this field.

The innovations achieved in human-computer interaction and user experience improve the quality and performance of software and user interfaces and play a vital role in the development of new technologies. These changes and innovations are usually quickly adopted in the IT industry and have a direct impact on people's user experience.

Challenges and open research issues in the field of deep learning and neural networks

Challenges and open issues in the field of deep learning and neural networks are considered one of the most challenging fields in artificial intelligence and machine learning. Due to the limitations and complexities in this field, many different cases are continuously researched. Some of these challenges and research issues are mentioned below:

1. Magnitude and complexity of models: One of the main challenges in deep learning is the creation and training of deep and complex models including a large number of layers and parameters. Management of many parameters and effective and fast implementation of training algorithms is one of the critical issues.

2. Learning on sparse data: In many domains, data is limited, and training deep models on sparse data is challenging. Finding ways to improve the performance of models with sparse data is a critical research topic.
3. Overfitting: This matter is one of the most important challenges in deep learning. Overfitting occurs if the model is trained on frequent training data but fails to generalize to new data. Controlling overfitting and increasing the generalization ability of models is an important research issue.
4. Understanding the performance of models: Due to the complexity and high number of layers, deep neural networks are often referred to as "black boxes" and it is challenging to accurately understand their performance. Research to improve the understanding and clarity of the models is still ongoing.
5. Reinforcement learning: The learning of deep models in the form of a reinforcement method in control and decision-making issues is considered a challenging research problem. Generalizing reinforcement learning to deep neural network architectures and performing complex learning on these models is always of interest.
6. Unsupervised learning: Training deep models without labeling data is one of the important issues in deep learning. Research is ongoing on developing methods for unsupervised learning and converting unlabeled data into examples that can be used in deep learning.

Due to their complexity, deep learning and neural networks are always facing important and challenging issues. Continuous efforts to solve these challenges and advance research in this field help the development of deep learning-based technologies and play a very important role in the evolution of artificial intelligence.

Conclusion

This study titled " Advances in Software Engineering: Research Boundaries and Innovations" took an in-depth and analytical look at the current and future state of software engineering. This study aims to examine the development of technology in this field and have a new look at the future of software engineering. In this study, we covered various topics such as complex software architecture, artificial intelligence-based software development, software modeling and analysis, software security and related ethical issues, and new software development and management methods. Also, by providing innovative and inspiring content and patterns in the research and development process of this field, we provided new tools and perspectives for interested audiences. Finally, we hope that this study will provide new insights to researchers, students, software industry professionals, and all those interested in software engineering topics and will be useful for conducting new research and innovations in this field. The development path of software engineering has always been full of challenges and attractive opportunities. We hope that this study will provide our readers with the necessary tools for new experiences in this direction.

Acknowledgments: None

Conflict of interest: None

Financial support: None

Ethics statement: None

References

1. Roger SP, Bruce RM. Software engineering: a practitioner's approach. 2015.
2. Martin RC. Agile software development: principles, patterns, and practices. Prentice Hall PTR; 2003 Sep 1.
3. Harrington P. Machine learning in action. Simon and Schuster; 2012 Apr 3.
4. Kim G, Humble J, Debois P, Willis J, Forsgren N. The DevOps handbook: How to create world-class agility, reliability, & security in technology organizations. IT Revolution; 2021 Nov 30. - Why: This handbook provides a comprehensive look at the principles and practices of DevOps, including continuous integration and continuous delivery (CI/CD).
5. McGraw G. Software security. IEEE Security & Privacy. 2004 Aug 2;2(2):80-3.
6. Desikan S, Ramesh G. Software testing: principles and practice. Pearson Education India; 2006.
7. Imran B. MASTERING BLOCKCHAIN: distributed ledger technology, decentralization, and smart contracts explained, ; distributed ledger. [Place of publication not identified] PACKT Publishing; 2018.
8. Shneiderman B, Plaisant C. Designing the user interface: strategies for effective human-computer interaction. Pearson Education India; 2010.
9. Gamma E, Helm R, Johnson R, Vlissides J. Design Patterns. Patterns.
10. Warren J, Marz N. Big Data: Principles and best practices of scalable realtime data systems. Simon and Schuster; 2015 Apr 29.
11. Tavani HT. Ethics and technology: Controversies, questions, and strategies for ethical computing. Wiley Publishing; 2012 Dec 3.