10.48047/jocaaa.2024.33.08.179

A REVIEW OF CHANGE MANAGEMENT IN COMPONENT-BASED SOFTWARE ENGINEERING: CURRENT PRACTICES AND FUTURE DIRECTIONS

Navnit Kumar Manglam[1], Dr. I B Lal[2]

[1]Research Scholar, Department of Computer Application, B.R.A Bihar University, Muzaffarpur Bihar
[2]Head of Department, Department of Computer Application, L N Mishra College of Business Management, Muzaffarpur
Email: iblal.lnmcbm@gmail.com

Received: 18.07.2024, Revised: 22.08.2024, Accepted: 20.09.2024

Abstract

Component-Based Software Engineering (CBSE) has emerged as a viable approach to software development that emphasizes building systems by assembling pre-existing software components. While CBSE offers significant advantages in terms of reusability, maintainability, and time-to-market, it introduces unique challenges for change management. This review paper examines the current state of research on change management in CBSE, synthesizing findings from existing literature and identifying gaps for future research. The review reveals that while substantial research exists on component models, development methodologies, and technical aspects of CBSE, there is limited focus on systematic approaches to managing changes in component-based systems. Key challenges identified include dependency management, version compatibility, interface stability, and organizational barriers to effective change management. The paper concludes by proposing directions for future research, emphasizing the need for integrated frameworks that address both technical and organizational aspects of change management in component-based software systems.

1. INTRODUCTION

Software systems have become increasingly complex and pervasive, serving as the backbone of critical infrastructure across virtually all sectors of society and industry. As software complexity grows, traditional development approaches have proven insufficient to meet the demands for rapid development, high quality, and adaptability to changing requirements. Component-Based Software Engineering (CBSE) emerged as a paradigm that addresses these challenges by focusing on building systems through the composition of independently developed and deployed software components (Szyperski, 1998).

The adoption of CBSE has been driven by its promise of increased reusability, improved maintainability, reduced time-to-market, and enhanced quality through the use of pre-tested components (Bakshi & Singh, 2013). However, the component-based approach introduces unique challenges for change management – the process of planning, implementing, and evaluating changes to software systems in a controlled and systematic manner.

Change management is particularly critical in CBSE for several reasons. First, changes to one component can potentially impact multiple other components due to

10.48047/jocaaa.2024.33.08.179

dependencies. Second, components often evolve independently, creating version compatibility challenges. Third, the distributed nature of component development, where components may be developed by different teams or organizations, complicates coordination of changes. Fourth, the integration of commercial off-the-shelf (COTS) components, open-source components, and in-house components creates a heterogeneous environment with varying change processes and lifecycles.

Despite these challenges, research on change management in CBSE remains fragmented, with limited attention to the development of comprehensive frameworks and methodologies. This review paper aims to address this gap by synthesizing existing research, identifying current practices, and highlighting areas for future investigation.

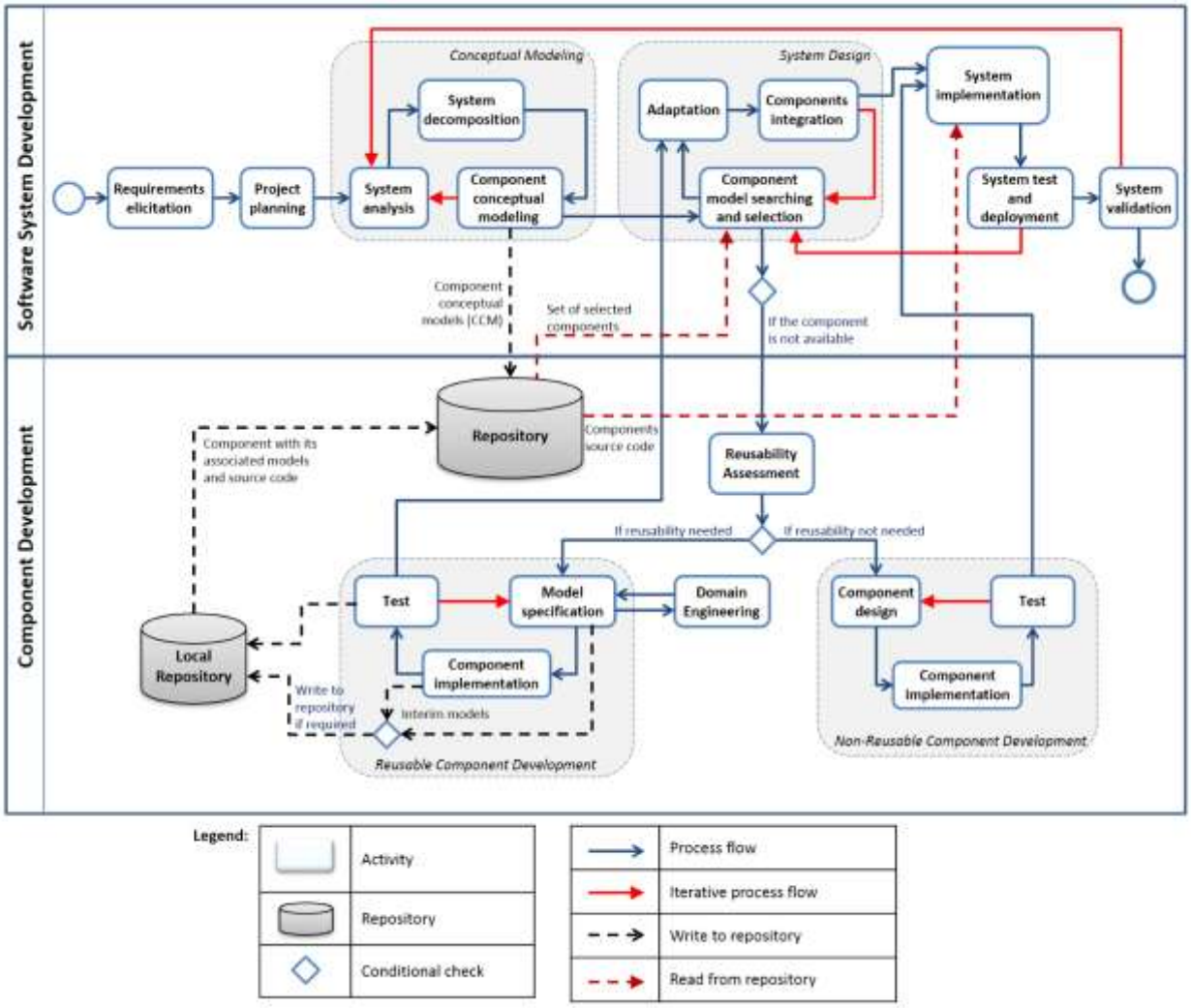

Figure 1. CompoMDD process model.

2. BACKGROUND: COMPONENT-BASED SOFTWARE ENGINEERING

2.1 Definition and Core Principles

Component-Based Software Engineering is defined as an approach to software development that emphasizes the design, construction, and deployment of software systems using reusable components. Szyperski (1998) provides a widely accepted definition of a software component as "a unit of composition with contractually specified interfaces and explicit context

10.48047/jocaaa.2024.33.08.179

dependencies only. A software component can be deployed independently and is subject to composition by third parties."

Table 1. Analysis of existing component-based process models.

Models	V [5]	Y [18]	W [19]	X [20]
Components can be deployed independently	Yes	Yes	Yes	Yes
Components can be composed with other components	Yes	Yes	Yes	Yes
Components can interact with other components	Yes	Yes	Yes	Yes
Separation between component development	Yes	Yes	Yes	Yes
and system development activities				
Adaptation step to modify the components	Yes	Yes	Yes	Yes
Component verified and validated before it is stored	Yes	No	Yes	Yes
in repository				
Use of domain engineering	No	Yes	Yes	Yes
Embedding another software development approach	No	No	No	No
Considering the potential reuse of components	No	No	No	No

The core principles of CBSE include:

1. **Component Reusability**: Components are designed to be reused across different applications and contexts.
2. **Component Independence**: Components can be deployed independently and composed without modification.
3. **Component Interoperability**: Components can work together through well-defined interfaces.
4. **Component Encapsulation**: Components hide their internal implementation details, exposing only necessary interfaces.
5. **Component Substitutability**: Components with the same interfaces can be substituted for one another.

These principles form the foundation of CBSE and guide the development of component models, frameworks, and methodologies.

10.48047/jocaaa.2024.33.08.179

2.2 Component Models and Technologies

Component models provide the technical infrastructure for component development, deployment, and composition. They define how components are structured, how they interact, and how they are managed at runtime. Several component models and technologies have emerged over the years, including:

1. **COM/DCOM**: Microsoft's Component Object Model and Distributed Component Object Model
2. **CORBA**: Common Object Request Broker Architecture
3. **JavaBeans/EJB**: Java's component models for desktop and enterprise applications
4. **OSGi**: A dynamic component model for Java
5. **Web Components**: Standards-based components for web applications

Emmerich and Kaveh (2002) provide a comparative analysis of these component technologies, highlighting their respective strengths and limitations. The choice of component model significantly influences the approach to change management, as different models offer varying levels of support for dynamic updates, version compatibility, and dependency management.

2.3 Component-Based Development Process

The component-based development process differs from traditional software development approaches in several key aspects. Crnkovic et al. (2006) describe a component-based development process that includes:

1. **Requirements Analysis**: Identifying system requirements and potential components.
2. **Component Identification**: Determining which components need to be developed, reused, or purchased.
3. **Component Selection/Acquisition**: Selecting or acquiring components that meet the requirements.
4. **Component Development**: Developing new components when required.
5. **System Integration**: Integrating components to build the complete system.
6. **System Testing**: Testing the integrated system.
7. **System Deployment**: Deploying the system in its operational environment.
8. **System Maintenance and Evolution**: Maintaining and evolving the system over time.

This process must be supported by appropriate tools, methodologies, and organizational structures to be effective. Capretz (2005) and Lau and Taweel (2011) have proposed specific lifecycle models for component-based software development, with the Y-model and W-model respectively, emphasizing different aspects of the development process.

3. CHANGE MANAGEMENT IN SOFTWARE ENGINEERING

Before examining change management specifically in the context of CBSE, it is important to understand the general principles and practices of software change management.

3.1 Definition and Scope

Software change management encompasses the processes, tools, and techniques used to manage changes to software systems throughout their lifecycle. It includes activities such as:

10.48047/jocaaa.2024.33.08.179

1. **Change Request Management**: Capturing, evaluating, and prioritizing change requests.
2. **Impact Analysis**: Assessing the potential impact of changes on the system.
3. **Change Implementation**: Planning and implementing changes in a controlled manner.
4. **Version Control**: Managing different versions of software artifacts.
5. **Configuration Management**: Identifying, tracking, and controlling changes to the system's configuration.
6. **Release Management**: Planning and coordinating software releases.

Effective change management is critical for maintaining software quality, controlling costs, and ensuring that changes align with organizational goals and user needs.

3.2 Traditional Change Management Approaches

Traditional approaches to software change management have evolved from simple, ad-hoc methods to more formalized processes. Key approaches include:

1. **Process-Oriented Approaches**: These approaches focus on defining and following standardized processes for managing changes, often based on frameworks such as ITIL (Information Technology Infrastructure Library) or CMMI (Capability Maturity Model Integration).
2. **Tool-Based Approaches**: These approaches emphasize the use of tools for version control, issue tracking, and configuration management to support the change management process.
3. **Risk-Based Approaches**: These approaches prioritize changes based on their potential risk to the system, focusing resources on high-risk changes.
4. **Agile Change Management**: This approach adapts change management to align with agile development methodologies, emphasizing flexibility, collaboration, and iterative implementation of changes.

Traditional change management approaches have typically been developed for monolithic software systems, where the entire system is developed, deployed, and maintained as a single unit. These approaches may not fully address the unique challenges of component-based systems.

4. UNIQUE CHALLENGES FOR CHANGE MANAGEMENT IN CBSE

Component-based software systems present unique challenges for change management that are not fully addressed by traditional approaches. These challenges stem from the fundamental characteristics of CBSE, such as independent component evolution, explicit dependencies, and distributed development.

4.1 Dependency Management

Component-based systems are characterized by complex dependency relationships between components. Changes to one component can potentially impact other components that depend on it, creating ripple effects throughout the system. As Vale et al. (2016) note, managing these dependencies becomes increasingly challenging as the number of components grows.
Key challenges in dependency management include:

10.48047/jocaaa.2024.33.08.179

1. **Identifying Dependencies**: Determining which components depend on a changed component, both directly and indirectly.
2. **Assessing Dependency Impact**: Evaluating how changes to a component will affect dependent components.
3. **Managing Transitive Dependencies**: Handling chains of dependencies where changes propagate through multiple levels.
4. **Resolving Dependency Conflicts**: Addressing situations where different components require different versions of the same dependency.

These challenges are exacerbated in systems that combine components from different sources (in-house, open-source, commercial), as these components may follow different evolution paths and have different change management processes.

4.2 Version Compatibility

Components in a component-based system often evolve independently, leading to version compatibility challenges. Szyperski (1998) highlights the "DLL hell" problem as an early manifestation of version compatibility issues in component-based systems.

Version compatibility challenges include:

1. **Interface Compatibility**: Ensuring that new versions of components maintain compatibility with their interfaces.
2. **Behavioral Compatibility**: Ensuring that new versions of components maintain consistent behavior even if their implementation changes.
3. **Version Selection**: Determining which version of a component to use when multiple versions are available.
4. **Version Coordination**: Coordinating the release of new versions of interdependent components.

These challenges require careful version management strategies, including semantic versioning, backward compatibility policies, and version resolution mechanisms.

4.3 Interface Stability

Interfaces play a central role in component-based systems, defining the contracts between components. Changes to interfaces can have significant impacts on the system, potentially breaking existing integrations.

Challenges related to interface stability include:

1. **Interface Design**: Designing interfaces that are flexible enough to accommodate future changes.
2. **Interface Evolution**: Evolving interfaces in a way that minimizes impact on existing clients.
3. **Interface Versioning**: Managing different versions of interfaces as they evolve.
4. **Interface Documentation**: Maintaining accurate documentation of interfaces and their changes.

Allen and Garlan (1997) emphasize the importance of formal specifications of architectural connections, which can help address some of these challenges by providing a rigorous basis for reasoning about interface changes.

10.48047/jocaaa.2024.33.08.179

4.4 Testing Complexity

Testing component-based systems presents unique challenges due to the distributed nature of component development and the potential for complex interactions between components.

Testing challenges include:

1. **Component-Level Testing**: Testing individual components in isolation.
2. **Integration Testing**: Testing the interactions between components.
3. **Regression Testing**: Ensuring that changes to one component do not break existing functionality in other components.
4. **Test Coverage**: Achieving adequate test coverage across all components and their interactions.

Javed et al. (2007) propose model-driven approaches to automated test case generation, which can help address some of these challenges by systematically generating tests based on component models.

4.5 Organizational Challenges

Beyond technical challenges, change management in component-based systems also faces organizational challenges, particularly when components are developed by different teams or organizations.

Organizational challenges include:

1. **Cross-Team Coordination**: Coordinating changes across teams responsible for different components.
2. **Change Communication**: Communicating changes effectively to all stakeholders.
3. **Change Prioritization**: Balancing the priorities of different stakeholders when planning changes.
4. **Governance**: Establishing effective governance structures for managing changes across components.

These organizational challenges require not only technical solutions but also appropriate processes, communication channels, and organizational structures.

5. CURRENT APPROACHES TO CHANGE MANAGEMENT IN CBSE

Research on change management specifically in the context of CBSE remains limited, but several approaches have been proposed that address aspects of the challenge.

5.1 Model-Driven Approaches

Model-driven engineering (MDE) has been proposed as an approach to managing complexity in component-based systems, including the management of changes. MDE involves creating high-level models of the system and its components, and using these models to drive the development, deployment, and evolution of the system.

Several researchers have explored the application of MDE to component-based systems:

- Atkinson et al. (2001) propose KobrA, a component-based model-driven architecture that emphasizes the use of models throughout the development lifecycle.
- Phung-Khac et al. (2008) present a model-driven approach for developing component-based adaptive distributed applications, which can help manage changes in dynamic environments.

10.48047/jocaaa.2024.33.08.179

- Kainz et al. (2010) describe a model-to-metamodel transformation approach for component-based systems, which can facilitate the evolution of component models.
- Liu et al. (2010) introduce rCOS, a formal model-driven approach for component-based development that provides a theoretical foundation for reasoning about changes.

These model-driven approaches offer several advantages for change management, including:

1. Higher-level abstraction, which can make changes easier to understand and implement.
2. Automated transformation between models, which can reduce the manual effort required for changes.
3. Formal foundations, which can help reason about the impact of changes.
4. Traceability between models, which can help identify the artifacts affected by changes.

However, these approaches also face challenges, including the complexity of maintaining consistent models, the need for specialized tools and expertise, and the difficulty of integrating with existing development processes.

5.2 Component Adaptation Approaches

Component adaptation approaches focus on modifying or wrapping components to accommodate changes without directly modifying the original components. This can be particularly useful when dealing with third-party components that cannot be modified directly.

Becker et al. (2006) present an engineering approach to component adaptation, which involves:

1. Identifying adaptation needs based on mismatches between components.
2. Selecting appropriate adaptation techniques.
3. Implementing adaptations using adapters, wrappers, or other mechanisms.
4. Validating the adapted components.

Brogi et al. (2004) propose metrics for measuring component adaptation, which can help evaluate the effectiveness of adaptation strategies and guide the selection of adaptation approaches.

Component adaptation approaches offer a flexible way to manage changes in component-based systems, particularly when components come from different sources. However, they can introduce additional complexity, performance overhead, and maintenance challenges.

5.3 Component Repository and Discovery Approaches

Component repositories provide a centralized location for storing, managing, and discovering components. They can play a critical role in change management by facilitating the identification of alternative components, tracking component versions, and maintaining component metadata.

Chatterjee and Rathi (2014) propose an approach for automating component repository search, which can help identify suitable components when changes require the replacement or addition of components.

10.48047/jocaaa.2024.33.08.179

Baker et al. (2006) present search-based approaches to component selection and prioritization, which can help navigate the potentially large space of available components when making changes.

Component repository and discovery approaches can support change management by:

1. Providing visibility into available components and their capabilities.
2. Tracking the evolution of components over time.
3. Facilitating the identification of alternative components when needed.
4. Maintaining metadata about component dependencies, compatibility, and quality attributes.

However, these approaches also face challenges, including the need for standardized component descriptions, the difficulty of assessing component quality, and the potential for repository fragmentation.

5.4 Lifecycle Models for Component-Based Systems

Specialized lifecycle models for component-based systems can provide a framework for managing changes throughout the system's life.

Capretz (2005) proposes the Y model, which emphasizes the parallel activities of domain engineering (creating reusable components) and application engineering (using components to build systems).

Lau and Taweel (2011) introduce the W model, which adds explicit verification and validation steps for components, helping ensure that changes maintain component quality.

Tomar and Gill (2010) present the X component-based model, which includes additional steps for component verification and validation.

These lifecycle models can support change management by:

1. Providing a structured process for planning and implementing changes.
2. Emphasizing the importance of verification and validation in maintaining system quality.
3. Distinguishing between changes to individual components and changes to the overall system.
4. Accounting for the parallel evolution of components and the systems that use them.

However, these models may require significant adaptation to fit specific organizational contexts and may not fully address all aspects of change management.

6. RESEARCH GAPS AND FUTURE DIRECTIONS

Despite the advances in component-based software engineering and the recognition of the importance of change management, several significant gaps remain in the research. These gaps represent opportunities for future research to enhance the state of practice in change management for component-based systems.

6.1 Integrated Change Management Frameworks

While various approaches address specific aspects of change management in CBSE, there is a lack of comprehensive, integrated frameworks that address all dimensions of the challenge. Future research should focus on developing holistic frameworks that integrate technical,

10.48047/jocaaa.2024.33.08.179

organizational, and process aspects of change management specifically tailored for component-based systems.

These frameworks should:

1. Provide end-to-end processes for managing changes from request to implementation.
2. Address both planned changes (e.g., feature additions) and unplanned changes (e.g., defect fixes).
3. Consider the unique characteristics of different types of components (in-house, open-source, commercial).
4. Integrate with existing development processes and tools.

6.2 Empirical Studies of Change Management Practices

There is a need for more empirical studies that examine how organizations actually manage changes in component-based systems. These studies could provide valuable insights into effective practices, common challenges, and potential improvements.

Areas for empirical investigation include:

1. How organizations coordinate changes across teams responsible for different components.
2. How they manage the dependencies between components during changes.
3. How they handle version compatibility challenges.
4. What tools and processes they use to support change management.

6.3 Impact Analysis Techniques

While some research has addressed impact analysis for component-based systems, there is a need for more sophisticated techniques that can accurately predict the effects of changes in complex, highly interconnected component ecosystems.

Future research in this area could explore:

1. Machine learning approaches to predicting change impacts based on historical data.
2. Visualization techniques to help developers understand potential ripple effects.
3. Formal methods for reasoning about the impact of changes across component boundaries.
4. Lightweight impact analysis techniques suitable for agile development environments.

6.4 Integration with Agile and DevOps Approaches

As agile development and DevOps practices become increasingly prevalent, there is a need for research on how to integrate change management for component-based systems with these approaches.

Alfraihi and Lano (2017) have examined the integration of agile development and model-driven development, but more research is needed specifically on change management in this context.

Areas for investigation include:

1. How to maintain component quality and compatibility in fast-paced agile environments.
2. How to apply DevOps practices (e.g., continuous integration, continuous delivery) to component-based systems.

10.48047/jocaaa.2024.33.08.179

3. How to balance the need for rapid changes with the need for stability in component interfaces.

6.5 Change Management for Emerging Component Paradigms

As new component paradigms emerge, such as microservices, serverless functions, and cloud-native components, there is a need for research on how to adapt change management approaches to these contexts.

These emerging paradigms present new challenges and opportunities for change management, including:

1. More frequent and fine-grained changes.
2. Dynamic composition and deployment of components.
3. Greater distribution and heterogeneity of components.
4. New dependency patterns and failure modes.

Research in this area could help organizations navigate the transition to these new paradigms while maintaining effective change management practices.

7. CONCLUSION

Change management in component-based software engineering represents a critical challenge that requires dedicated attention from researchers and practitioners. While CBSE offers significant advantages in terms of reusability, maintainability, and time-to-market, it also introduces unique challenges for managing changes due to component dependencies, independent evolution, and distributed development.

This review has highlighted several approaches to addressing these challenges, including model-driven approaches, component adaptation techniques, repository and discovery mechanisms, and specialized lifecycle models. However, significant gaps remain, particularly in the development of integrated frameworks, empirical understanding of current practices, advanced impact analysis techniques, integration with agile and DevOps approaches, and adaptation to emerging component paradigms.

Future research in these areas has the potential to significantly enhance the state of practice in change management for component-based systems, ultimately leading to more adaptable, maintainable, and resilient software systems. As software continues to grow in complexity and importance, effective change management will be increasingly critical for organizations seeking to leverage the benefits of component-based software engineering while managing the associated challenges.

REFERENCES

1. Szyperski, C. Component Software: Beyond Object-Oriented Programming; ACM Press/Addison-Wesley Publishing Co.: New York, NY, USA, 1998. [Google Scholar]
2. Sommerville, I. Software Engineering, 10th ed.; Pearson: London, UK, 2016. [Google Scholar]
3. Emmerich, W.; Kaveh, N. Component Technologies: Java Beans, COM, CORBA, RMI, EJB and the CORBA Component Model. In Proceedings of the 24th International Conference on Software Engineering, ICSE ’02, Orlando, FL, USA, 25

10.48047/jocaaa.2024.33.08.179

May 2002; ACM: New York, NY, USA, 2002; pp. 691–692. [Google Scholar] [CrossRef]

4. Khemakhem, S.; Drira, K.; Jmaiel, M. Chapter 8: Description, classification and discovery approaches for software components: A comparative study. In Modern Software Engineering Concepts and Practices: Advanced Approaches; Information Science Reference: Warszawy, Poland, 2010; pp. 196–219. [Google Scholar] [CrossRef]
5. Crnkovic, I.; Chaudron, M.; Larsson, S. Component-Based Development Process and Component Lifecycle. In Proceedings of the International Conference on Software Engineering Advances, Tahiti, France, 29 October–3 November 2006; pp. 44–60. [Google Scholar] [CrossRef] [Green Version]
6. Bakshi, A.; Singh, R. Component Based Development in Software Engineering. Int. J. Recent Technol. Eng. (IJRTE) 2013, 2, 48–52. [Google Scholar]
7. Chatterjee, R.; Rathi, H. A prolific approach towards automating component repository search. In Proceedings of the Seventh International Conference on Contemporary Computing (IC3), Noida, India, 7–9 August 2014; pp. 547–552. [Google Scholar]
8. Baker, P.; Harman, M.; Steinhofel, K.; Skaliotis, A. Search Based Approaches to Component Selection and Prioritization for the Next Release Problem. In Proceedings of the 22nd IEEE International Conference on Software Maintenance, ICSM ’06, Chicago, IL, USA, 11–14 September 2006; IEEE Computer Society: Washington, DC, USA, 2006; pp. 176–185. [Google Scholar] [CrossRef]
9. Becker, S.; Brogi, A.; Gorton, I.; Overhage, S.; Romanovsky, A.; Tivoli, M. Towards an Engineering Approach to Component Adaptation. In Proceedings of the International Conference on Architecting Systems with Trustworthy Components; Springer: Berlin/Heidelberg, Germany, 2006; pp. 193–215. [Google Scholar]
10. Brogi, A.; Canal, C.; Pimentel, E. Measuring Component Adaptation. In Coordination Models and Languages; De Nicola, R., Ferrari, G.L., Meredith, G., Eds.; Springer: Berlin/Heidelberg, Germany, 2004; pp. 71–86. [Google Scholar]
11. Çetinkaya, D.; Verbraeck, A.; Seck, M.D. Model Continuity in Discrete Event Simulation: A Framework for Model-Driven Development of Simulation Models. ACM Trans. Model. Comput. Simul. 2015, 25, 17:1–17:24. [Google Scholar] [CrossRef]
12. Syriani, E.; Gray, J.; Vangheluwe, H. Domain Engineering: Product Lines, Languages, and Conceptual Models. In Domain Engineering: Product Lines, Languages, and Conceptual Models; Chapter Modeling a Model Transformation Language; Springer: Berlin/Heidelberg, Germany, 2013; pp. 211–237. [Google Scholar] [CrossRef]
13. Badampudi, D.; Wohlin, C.; Petersen, K. Software component decision-making: In-house, OSS, COTS or outsourcing—A systematic literature review. J. Syst. Softw. 2016, 121, 105–124. [Google Scholar] [CrossRef]
14. Alfraihi, H.; Lano, K. The Integration of Agile Development and Model Driven Development—A Systematic Literature Review. In Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development,

10.48047/jocaaa.2024.33.08.179

Porto, Portugal, 19–21 February 2017; INSTICC, SciTePress: Setúbal, Portugal, 2017; pp. 451–458. [Google Scholar] [CrossRef]

15. Alrubaee, A.U. A Component Based Model Driven Software Development Framework for Web-Based Applications. Master's Thesis, Software Engineering MSc Program, Graduate School of Natural and Applied Sciences, Atilim University, Ankara, Turkey, 2017. [Google Scholar]
16. Brambilla, M.; Cabot, J.; Wimmer, M. Model-Driven Software Engineering in Practice, 2nd ed.; Morgan & Claypool Publishers: Mountain View, CA, USA, 2017. [Google Scholar]
17. Brown, A.W.; Conallen, J.; Tropeano, D. Introduction: Models, Modeling, and Model-Driven Architecture (MDA). In Model-Driven Software Development; Springer: Berlin/Heidelberg, Germany, 2005; pp. 1–16. [Google Scholar] [CrossRef]
18. Capretz, L. Y: A new component-based software life cycle model. J. Comput. Sci. Sci. Publ. 2005, 1, 76–82. [Google Scholar] [CrossRef] [Green Version]
19. Lau, K.K.; Taweel, F.M.; Tran, C.M. The W Model for Component-Based Software Development. In Proceedings of the 37th EUROMICRO Conference on Software Engineering and Advanced Applications, Oulu, Finland, 30 August–2 September 2011; pp. 47–50. [Google Scholar]
20. Tomar, P.; Gill, N.S. Verification amp; Validation of components with new X Component-Based Model. In Proceedings of the 2010 2nd International Conference on Software Technology and Engineering, San Juan, PR, USA, 3–5 October 2010; Volume 2, pp. V2-365–V2-371. [Google Scholar]
21. Vale, T.; Crnkovic, I.; de Almeida, E.S.; da Mota Silveira Neto, P.A.; Cavalcanti, Y.C.; de Lemos Meira, S.R. Twenty-eight years of component-based software engineering. J. Syst. Softw. 2016, 111, 128–148. [Google Scholar] [CrossRef]
22. Rodrigues da Silva, A. Model-driven Engineering. Comput. Lang. Syst. Struct. 2015, 43, 139–155. [Google Scholar] [CrossRef] [Green Version]
23. Kapteijns, T.; Jansen, S.; Brinkkemper, S.; Houet, H.; Barendse, R. A Comparative Case Study of Model Driven Development vs Traditional Development: The Tortoise or the Hare. In Proceedings of the 4th European Workshop on from Code Centric to Model Centric Software Engineering: Practices, Implications and ROI, Enschede, The Netherlands, 24 June 2009. [Google Scholar]
24. Bucchiarone, A.; Cabot, J.; Paige, R.F.; Pierantonio, A. Grand challenges in model-driven engineering: An analysis of the state of the research. Softw. Syst. Model. 2020, 19, 5–13. [Google Scholar] [CrossRef] [Green Version]
25. Mussbacher, G.; Amyot, D.; Breu, R.; Bruel, J.M.; Cheng, B.H.C.; Collet, P.; Combemale, B.; France, R.B.; Heldal, R.; Hill, J.; et al. The Relevance of Model-Driven Engineering Thirty Years from Now. In Model-Driven Engineering Languages and Systems; Dingel, J., Schulte, W., Ramos, I., Abrahão, S., Insfran, E., Eds.; Springer International Publishing: Berlin/Heidelberg, Germany, 2014; pp. 183–200. [Google Scholar]
26. Bocciarelli, P.; D'Ambrogio, A.; Giglio, A.; Paglia, E. Model-Driven Distributed Simulation Engineering. In Proceedings of the 2019 Winter Simulation Conference

10.48047/jocaaa.2024.33.08.179

(WSC), National Harbor, MD, USA, 8–11 December 2019; pp. 75–89. [Google Scholar]

27. Bocciarelli, P.; D'Ambrogio, A.; Falcone, A.; Garro, A.; Giglio, A. A model-driven approach to enable the simulation of complex systems on distributed architectures. SIMULATION 2019, 95, 1185–1211. [Google Scholar] [CrossRef]
28. Atkinson, C.; Paech, B.; Reinhold, J.; Sander, T. Developing and applying component-based model-driven architectures in KobrA. In Proceedings of the Fifth IEEE International Enterprise Distributed Object Computing Conference, Seattle, WA, USA, 4–7 September 2001; pp. 212–223. [Google Scholar]
29. Weiss, K.A.; Ong, E.C.; Leveson, N.G. Reusable specification components for model-driven development. In Proceedings of the International Conference on System Engineering (INCOSE'03), Portland, OR, USA, 3–10 May 2003. [Google Scholar]
30. Phung-Khac, A.; Beugnard, A.; Gilliot, J.M.; Segarra, M.T. Model-driven Development of Component-based Adaptive Distributed Applications. In Proceedings of the 2008 ACM Symposium on Applied Computing; ACM: New York, NY, USA, 2008; pp. 2186–2191. [Google Scholar] [CrossRef] [Green Version]
31. Kainz, G.; Buckl, C.; Sommer, S.; Knoll, A. Model-to-Metamodel Transformation for the Development of Component-Based Systems. In Model Driven Engineering Languages and Systems; Petriu, D.C., Rouquette, N., Haugen, Ø., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 391–405. [Google Scholar]
32. Liu, Z.; Morisset, C.; Stolz, V. rCOS: Theory and Tool for Component-Based Model Driven Development. In Fundamentals of Software Engineering; Arbab, F., Sirjani, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 62–80. [Google Scholar]
33. Clemente, P.J.; Hernández, J.; Conejero, J.M.; Ortiz, G. Managing crosscutting concerns in component based systems using a model driven development approach. J. Syst. Softw. 2011, 84, 1032–1053. [Google Scholar] [CrossRef]
34. Kathayat, S.B.; Le, H.N.; Bræk, R. A Model-Driven Framework for Component-Based Development. In SDL 2011: Integrating System and Software Modeling; Ober, I., Ober, I., Eds.; Springer: Berlin/Heidelberg, Germany, 2011; pp. 154–167. [Google Scholar]
35. Agustin, J.L.H.; del Barco, P.C. A model-driven approach to develop high performance web applications. J. Syst. Softw. 2013, 86, 3013–3023. [Google Scholar] [CrossRef]
36. Mizuno, T.; Matsumoto, K.; Mori, N. Applying Component-Based Technologies to Model-Driven Software Development. Electron. Commun. Jpn. 2015, 98, 24–31. [Google Scholar] [CrossRef]
37. Ciccozzi, F.; Carlson, J.; Pelliccione, P.; Tivoli, M. Editorial to theme issue on model-driven engineering of component-based software systems. Softw. Syst. Model. 2017. [Google Scholar] [CrossRef] [Green Version]
38. Allen, R.; Garlan, D. A Formal Basis for Architectural Connection. ACM Trans. Softw. Eng. Methodol. (TOSEM) 1997, 6, 213–249. [Google Scholar] [CrossRef] [Green Version]
39. Javed, A.Z.; Strooper, P.; Watson, G. Automated Generation of Test Cases Using Model-Driven Architecture. In Proceedings of the Workshop on Automation of

10.48047/jocaaa.2024.33.08.179

Software Test (AST'07), Minneapolis, MN, USA, 26 May 2007. [Google Scholar] [CrossRef]

40. Shiva, S.G.; Shala, L.A. Software Reuse: Research and Practice. In Proceedings of the 4th International Conference on Information Technology, Las Vegas, NV, USA, 2–4 April 2007; pp. 603–609. [Google Scholar]
41. Sametinger, J. Software Engineering with Reusable Components; Springer: Berlin/Heidelberg, Germany, 1997. [Google Scholar]
42. Desouza, K.C.; Awazu, Y.; Tiwana, A. Four Dynamics for Bringing Use Back into Software Reuse. Commun. ACM 2006, 49, 96–100. [Google Scholar] [CrossRef]
43. Councill, B.; Heineman, G.T. Definition of a Software Component and Its Elements. In Component-Based Software Engineering; Heineman, G.T., Councill, W.T., Eds.; Addison-Wesley Longman Publishing Co., Inc.: Boston, MA, USA, 2001; pp. 5–19. [Google Scholar]
44. Emerson, M.; Neema, S.; Sztipanovits, J. Metamodeling Languages and Metaprogrammable Tools. In Handbook of Real-Time and Embedded Systems; Lee, I., Leung, J.Y.T., Son, S.H., Eds.; Taylor & Francis Group: Abingdon, UK, 2008; Chapter 33; pp. 1–16. [Google Scholar]
45. Institute, P.M. A Guide to the Project Management Body of Knowledge (PMBOK® Guide), 6th ed.; Project Management Institute, Inc.: Newtown Square, PA, USA, 2017. [Google Scholar]
46. Acerbis, R.; Bongio, A.; Brambilla, M.; Butti, S.; Ceri, S.; Fraternali, P. Web Applications Design and Development with WebML and WebRatio 5.0. In Objects, Components, Models and Patterns; Paige, R.F., Meyer, B., Eds.; Springer: Berlin/Heidelberg, Germany, 2008; pp. 392–411. [Google Scholar]
47. Kroiss, C.; Koch, N.; Knapp, A. UWE4JSF: A Model-Driven Generation Approach for Web Applications. In Web Engineering; Gaedke, M., Grossniklaus, M., Díaz, O., Eds.; Springer: Berlin/Heidelberg, Germany, 2009; pp. 493–496. [Google Scholar]