TPM Vol. 32, No. S8, 2025
ISSN: 1972-6325
https://www.tpmap.org/

Open Access

PROACTIVE COMPONENT LIFE CYCLE MODEL (PCLCM)

SANDEEP CHOPRA
RESEARCH SCHOLAR, MDU10635, MAYA DEVI UNIVERSITY, DEHRADUN

RAM BHAWAN SINGH
DEAN, COMPUTER SCIENCE & APPLICATIONS, MAYA DEVI UNIVERSITY, DEHRADUN

ABSTRACT
Information technology has become essential for both routine jobs and intricate scientific undertakings in today's quickly changing digital landscape. Software development issues including high prices, time limits, and rapid change are getting worse as demands for quicker delivery and more adaptable solutions rise. Component-Based Software Engineering (CBSE) is a strategy approach that uses reusable, modular components to improve scalability, lower costs, and simplify development. In order to meet the changing needs of contemporary IT settings, this article presents an improved life-cycle model for CBSE. The suggested model provides reliable, usable software that is suited to quickly evolving business needs by combining automated testing, risk and quality certification, and a proactive feedback loop. This ensures dependable performance, high maintainability, and effective resource use within time and financial constraints.

Keywords: Component Based Life Cycle Model, Software Development Architecture , COTS, and Component-Based Development (CBD), Component-Based Software Engineering(CBSE)

1. INTRODUCTION

Component-based development has been a fundamental paradigm in the IT industry over the last 20 years. This approach focuses on putting software together from modular, separately created elements called components, each of which is intended to provide a particular function with a clearly defined interface. This method's popularity is directly related to the increasing need for quick application delivery, improved maintainability, and adaptability to changing business needs.

By reusing pre-built, extensively tested modules, component-based software engineering (CBSE) helps businesses create scalable and reliable solutions. Because teams can concentrate on integrating current assets rather than creating systems from the ground up, this results in quicker development cycles, easier maintenance, and significant cost savings. Additionally, CBSE facilitates parallel development initiatives and promotes cooperation among development teams, which shortens time-to-market and raises overall quality[1].

The modular design of CBSE offers a distinct benefit as applications grow more complex: individual parts can be scaled, updated, or replaced separately without compromising the system's overall integrity. This flexibility creates a solid basis for innovation in contemporary software engineering while also making routine maintenance easier[2].

2. State of Art

The field of Component-Based Software Engineering (CBSE) encompasses a wide variety of models developed to address the growing complexity and demands of modern software projects. These models provide structured approaches for assembling, integrating, and maintaining software systems using reusable components, improving development speed, quality, and adaptability.

Among the most widely recognized models is the Twin Peaks Model[13], which promotes the parallel refinement of requirements and architecture throughout project development, supporting flexible and evolutionary design.

The X Model [14]centers on the principle of building systems from reusable, testable modules, thereby encouraging reliability and efficient delivery.

The Y Model[15] introduces overlapping and iterative phases such as domain engineering, framework creation, system assembly, archiving, and maintenance. This approach emphasizes adaptability and reuse, making it well-suited to large-scale and complex solutions.

The Umbrella Model [16]divides the development life cycle into three primary stages: design (component selection from repositories), integration (joining components via defined interfaces), and runtime initialization. At each stage, rigorous testing and verification are applied to ensure consistency and reliability.

In the COTS[20] Model, developers integrate commercially available, third-party components built using diverse programming languages and platforms. This flexibility facilitates quick assembly of systems tailored to specific needs, often reducing development time and cost.

The V Model[17] adapts sequential development and integrates thorough testing practices across design and implementation stages, while the W Model extends the V approach by further distinguishing between design and deployment phases. Notably, the W Model highlights verification and validation at both component and system levels. The Knot Model[19] focuses on continual risk analysis and feedback within iterative cycles, making it well-suited for complex environments requiring ongoing evaluation and quality improvement.
These models together reflect the industry's shift towards greater modularity, reusability, and adaptability, optimizing the software engineering process for speed, quality, and cost-effectiveness in contemporary IT environments.
The Elena model [23] is a novel component-based software engineering (CBSE) life cycle framework designed to enhance flexibility and adaptability in software development. It centers on the rapid assembly of systems by selecting and integrating reusable components through clearly defined interfaces. What distinguishes the Elena model is its focus on both static (unchanging) and dynamic (evolving) requirements: it supports updating requirements according to new market trends and emphasizes stepwise validation of both the product and its components. The model incorporates customer feedback at multiple stages to ensure the final product meets current and future needs, making it suitable for developing small to large-scale software solutions in a modern, user-focused, and iterative manner.

3. 'Proactive Component Life Cycle Model '

A contemporary software engineering methodology called the Proactive Component Life Cycle Model (PCLCM) was created to optimize component-based development's quality, efficiency, and agility. By adding certification-oriented and feedback-driven phases that cater to changing business requirements and technology breakthroughs, this model expands on the conventional CBSE life cycles[5].

Core Features of the Model

3.1 Requirement Categorization:

At the initial stage, requirements are divided into 'Persistent' and 'Evolving' categories, recognizing which needs are long-term and which may change with stakeholder feedback and market trends.

3.2 Continuous Market Trend Analysis:

• In a setting that is changing quickly, requirements and components are kept current and competitive through regular changes derived from market study.

• The Proactive Component Life Cycle Model (PCLCM)'s Milestone-Based Customer Review is separated into two distinct sections, Persistent needs and Evolving Requirements, each of which is adapted to the type of needs. Every component guarantees that client participation corresponds with the stability or adaptability required at various phases of software development.

1. Persistent Requirements Review

Persistent requirements are those that remain stable and unchanged throughout the software lifecycle. For these:
• Early and Thorough Validation: Since these needs are the cornerstone of the system's essential features, there is a great deal of customer engagement at the first milestones to finalize and freeze these requirements.
• Minimal Repetition: After being decided upon, persistent requirements are reviewed less frequently to prevent needless delays and to make sure that integration and development adhere to the predetermined standards.
• Emphasis on Formal Acceptance: To ensure that the system's unchangeable components are implemented correctly, the customer review seeks formal approval at crucial junctures like requirements formulation and pre-release testing.
• Risk Mitigation: By preventing expensive rework later on, early validation helps reduce the risks associated with misunderstanding or misinterpretation of stable requirements.

2. Evolving Requirements Review

Requirements that are subject to change over time as a result of user input, market conditions, or technical advancements are known as evolving requirements.
• Iterative and Adaptive Feedback: At several pre-established milestones, such as prototype releases, beta tests, or following significant feature increments, iterative or cyclical customer engagement takes place.
• Focused Refinements: The goal of reviews is to collect targeted input in order to improve features, modify functionalities, and fit the system to the evolving business landscape.
• Fast Feedback Assimilation: Contains tools for promptly incorporating user feedback to change requirements and impact later stages of development.
• Innovation Promotion: By enabling the system to adapt in real time to usage and new trends, this dynamic review process promotes innovation.
A balanced, customer-centric development strategy is achieved by PCLCM's split approach to customer review, which guarantees that while essential core requirements stay constant and are thoroughly validated, flexible areas change in response to real user needs and market situations.
Promotion of Innovation: By enabling the system to change in response to current usage and new trends, this dynamic review process promotes innovation.

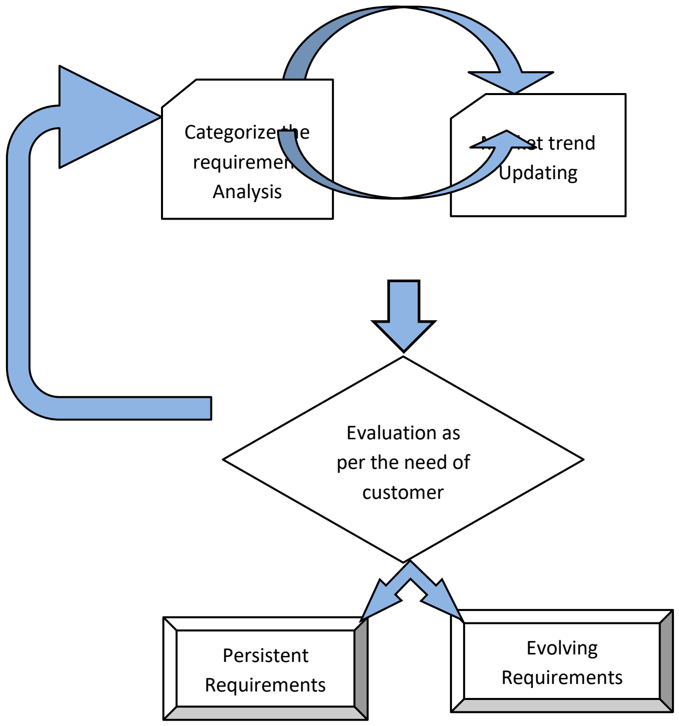

Categorize the requirement Analysis
Market trend Updating
Evaluation as per the need of customer
Persistent Requirements
Evolving Requirements

Figure 1- Steps involved in requirement analysis

3.3 Milestone-Based Customer Review:
Prioritizing customer involvement at significant stages streamlines review procedures, cuts down on needless repetition, and collects targeted feedback for improved alignment.
3.4 Comprehensive Documentation:
Every step produces accurate documentation (such as SRS and architecture definitions), improving traceability and simplifying audits, upgrades, and maintenance in the future.
3.5 Component Selection and Certification:
Components are selected from repositories, designed anew, modified, or outsourced according to project needs. Dedicated risk analysis and certification steps ensure high standards and compatibility. A module in software engineering can be described as a collection of connection points (ports) and operational attributes. This can be mathematically represented as:
$M=(P,OA)$ where M stands for the module, P denotes the set of ports (interfaces) such that $P=\{P_1,P_2,...,P_n\}$and OA represents the set of operational attributes, i.e., $OA=\{OA_1,OA_2,...,OA_k\}$[6].
3.6 Integration and Automated Testing:
The Proactive Component Life Cycle Model's Integration and Automated Testing maximizes speed and reliability by combining static and dynamic testing techniques. Following the assembly of all parts via well-defined interfaces, the software is put through two types of automated testing:
• Static testing: Prior to actual execution, source code, design documentation, and architectural specifications must be reviewed. Tools are used to analyse code, verify that standards are being followed, and find mistakes early in the lifecycle, including inconsistent interfaces or security flaws. Static testing helps maintain the integrity of linked modules by identifying flaws that could otherwise go undetected until later.
• Dynamic Testing: This technique validates real-time interactions between components by automating the execution of test cases once the integrated system is operational. Regression, system, and integration testing are all covered by this method, which makes sure that each module works properly both separately and in concert with the others. Dynamic tests reveal runtime problems, validate real behaviour, and offer input for ongoing development.

The PCLCM reduces manual testing labour and produces a more reliable, production-ready solution by combining static analysis and dynamic test automation to assure robust error detection, high dependability, and quick defect resolution.

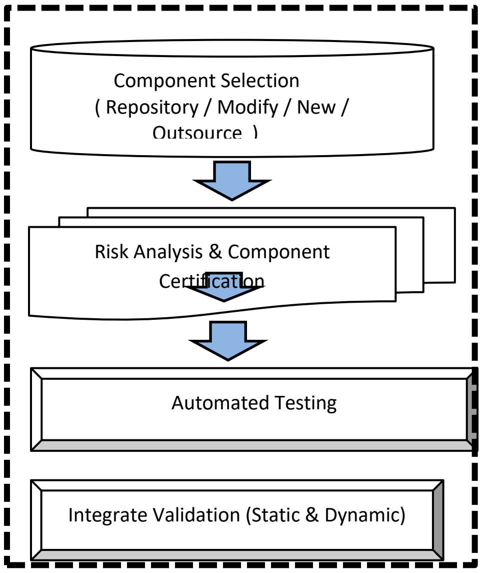

Component Selection
(Repository / Modify / New / Outsource)
Risk Analysis & Component Certification
Automated Testing
Integrate Validation (Static & Dynamic)

Figure 2- Selection of Component Integration and Validation

3.7 Unified Validation and Feedback:
Unified validation and feedback in the Proactive Component Life Cycle Model means that both persistent (fixed) and evolving (changing) requirements are assessed through a single, coherent process that harmoniously blends static analysis and dynamic testing, supported by input from technical teams and end users.
How Unified Validation Works
Integrated Requirement Review: Walkthroughs, checklists, and prototypes are used to verify both sorts of requirements—those that are expected to change and those that are expected to stay the same. This guarantees that all requirements are known and practical before development starts.
Concurrent Validation, Static and Dynamic:
Automated consistency analysis and document reviews are examples of static validation approaches that verify requirements and design artifacts for completeness, clarity, and compliance.
Creating prototypes, executing test cases, and modeling system behavior are all part of dynamic validation, which verifies that requirements are satisfied under operational circumstances..
Feedback from the entire organization:
Validation makes use of input from users, project team members, quality assurance, and business owners. By identifying ambiguities or gaps early on, this collaborative review reduces the need for rework and miscommunication.
Loops for Customer Feedback:
To ensure that software not only satisfies static criteria but also successfully adjusts to changing demands, user feedback is methodically gathered and integrated following each significant milestone or version. Feedback is utilized for real-time improvement in the event that requirements change or become new.
Final Sign-off:
The unified process culminates in comprehensive re-testing and formal stakeholder review, ensuring all requirements are satisfied and the system is ready for deployment. This approach delivers thorough validation for both foundational and flexible features, enabling continuous improvement and alignment of the software with organizational and customer objectives throughout its life cycle.
3.8 Final Delivery and Maintenance:
The model supports robust delivery, with maintenance strategies embedded to allow easy future upgrades and ongoing support.
Agility: Designed to quickly respond to changing requirements and market conditions without major disruptions.
Quality and Reliability: Emphasizes risk analysis, automated testing, and certification to consistently deliver high-quality, reliable software.
Efficiency: Streamlines development through reusable, certified components—cutting costs, reducing development time, and supporting fast iterative releases.
Summary Table:

Key Phase	**Description**
Requirement Categorization	Persistent vs Evolving needs identified, documented

Market Trends	Continuous analysis and updates
Customer Review	Milestone-based, focused, actionable feedback
Documentation	SRS, architecture, component specs maintained
Component Certification	Risk analysis, compatibility verification
Integration & Testing	Automated, interface-driven testing & integration
Validation & Feedback	Unified, integrated with component and user review
Maintenance & Delivery	Built-in strategies for smooth upgrades and support

4. A Comparative Analysis of available Component-Based model with the proposed model

How the Proactive Component Life Cycle Model (PCLCM) differs from conventional component-based software engineering (CBSE) models in theory and application is examined through a comparative analysis. The theoretical differences are listed below, along with a straightforward tabular graphic that highlights the most important ones.

Theoretical Comparison

The Twin Peaks, X, Y, Umbrella, V, W, Knot, and COTS structures are examples of traditional CBSE models that concentrate on particular topics, such as modularity, reuse, iterative development, or risk analysis[24]. But a lot of them either don't have strong feedback loops, don't incorporate constant validation, or aren't flexible enough to deal with changing needs[25]. The Proactive Component Life Cycle Model, on the other hand, formalizes iterative validation throughout each phase and centralizes customer-driven feedback. Particularly, PCLCM:combines validation for both changing and permanent requirements, ensuring that features are constantly examined despite their stability. continuously maximizes quality and reduces errors by integrating both static and dynamic automated testing with CI tools. gives milestone-based customer reviews top priority, closely coordinating software development with continuing business goals. encourages dependability and makes future upgrades simpler by requiring risk and quality certification for every component. supports continuous documentation and open communication to make compliance and maintainability easier.

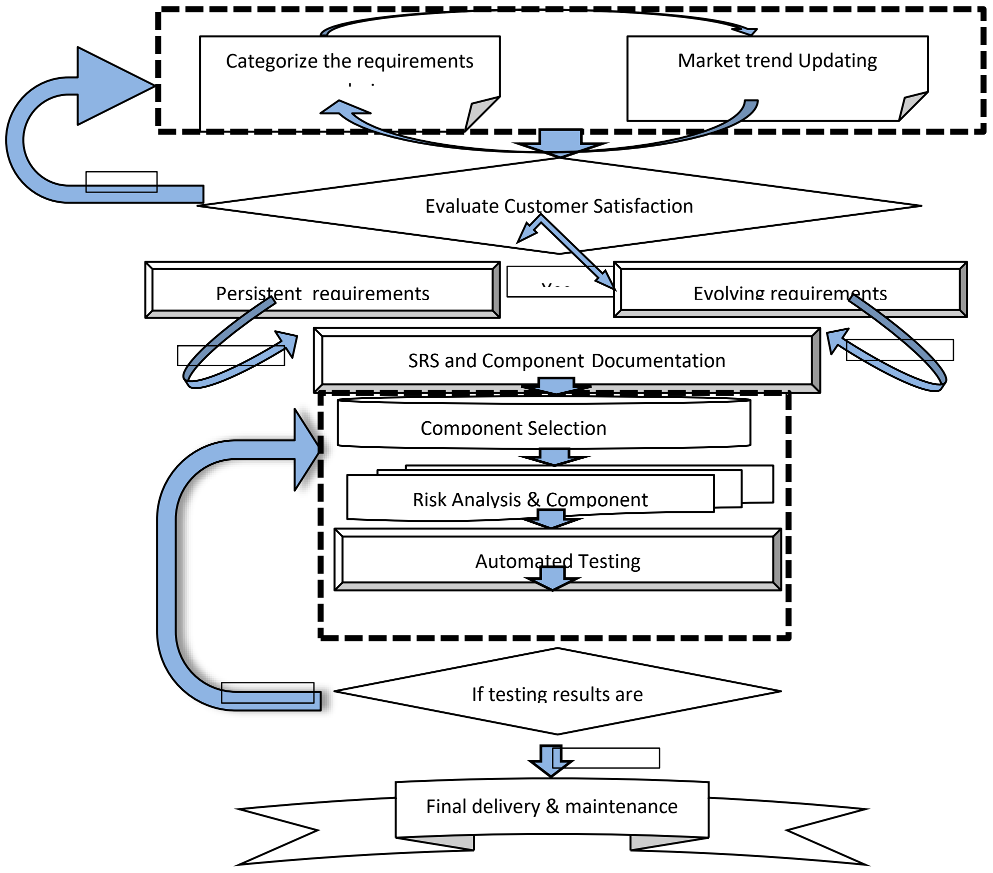

Categorize the requirements
Market trend Updating
Evaluate Customer Satisfaction
Persistent requirements
Evolving requirements
SRS and Component Documentation
Component Selection
Risk Analysis & Component
Automated Testing
If testing results are
Final delivery & maintenance

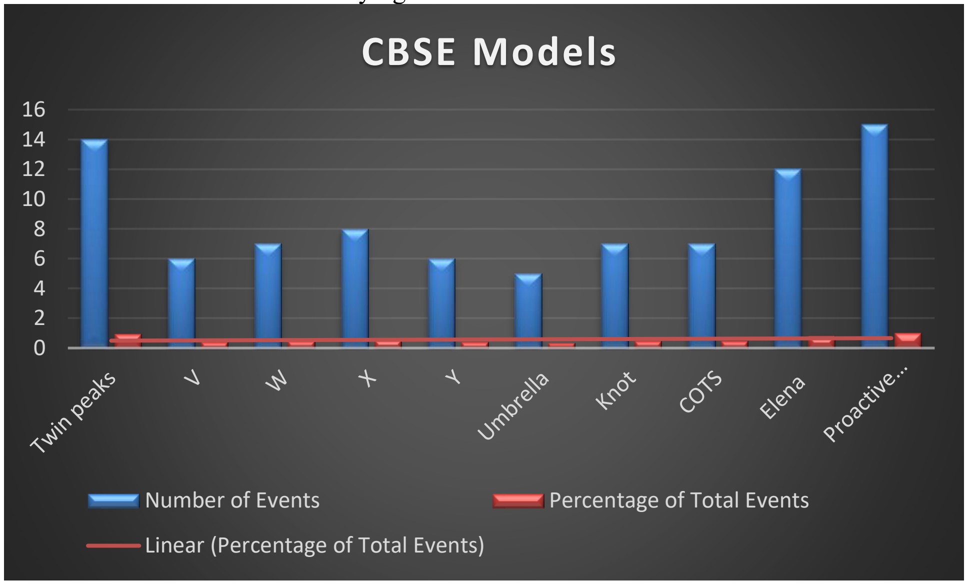
Figure 3- Proposed ProActive Component Life Cycle Model (PCLCM)

Table 1 – Comparative analysis of existing CBSE models

Model / Features	Twin Peak	X	Y	Umbrella	V	W	Knot	COTS	PCLCM (Proactive)
Reusability	SS	SS	SS	SS	SS	SS	SS	SS	SS
Iterative Feedback	NS	NS	SS	NS	NS	NS	SS	NS	SS
Customer Milestone Review	NS	NS	SS	NS	NS	NS	PS	NS	SS
Unified Validation	NS	NS	NS	NS	NS	NS	NS	NS	SS
Static + Dynamic Testing	PS	NS	NS	NS	PS	PS	PS	NS	SS
Risk Certification	NS	NS	NS	NS	NS	NS	SS	PS	SS
Persistent + Evolving Requirements Support	NS	NS	PS 1	NS	NS	NS	PS	NS	SS
Continuous Documentation	PS	NS	NS	NS	NS	NS	NS	NS	SS
CI/CD Integration	NS	NS	NS	NS	NS	NS	NS	NS	SS

Key: SS = Strong Support; PS= Partial Support; NS = No Support

I found various events while studying different CBSE models and created a chart.

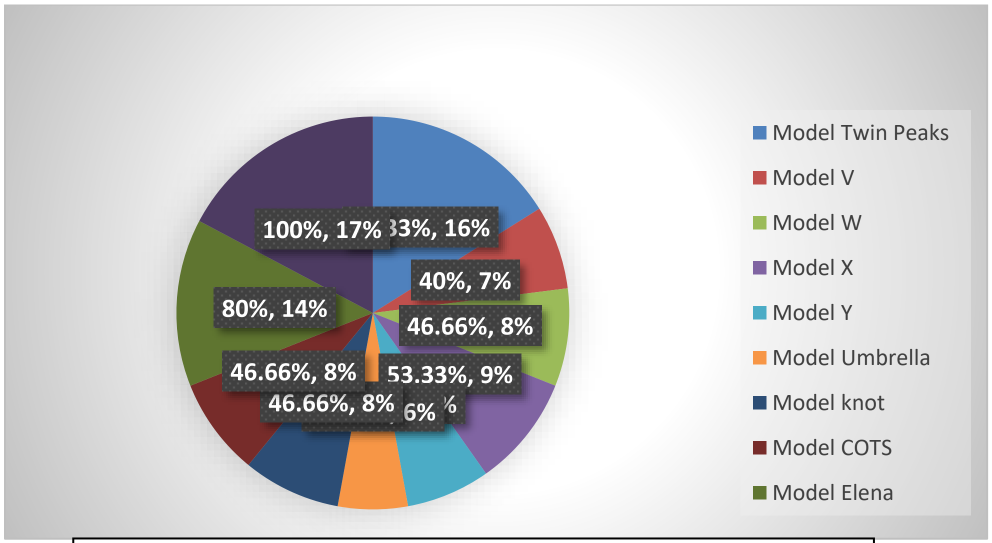


According to my conclusion these CBSE models have fifteen events.
Number of events found in Twin Peaks model=14
Number of events found in V model = 6
Number of events found in W model = 7
Number of events found in X model = 8
Number of events found in Y model = 6
Number of events found in Umbrella model = 5
Number of events found in knot model = 7
Number of events found in COTS model = 7
Number of events found in Elena Dynamic model = 12
Number of events found in PCLCM model=15

The following diagram drawn underneath shows the capability of various CBSE models.
By using this formula :

Percentage of events = (Number of events in a model) X 100 / (Total number of events)

Model Twin Peaks=	(14) X 100 / (15)=	93.33%
Model V =	(6) X 100 / (15) =	40%
Model W=	(7) X 100 / (15) =	46.66%
Model X =	(8) X 100 / (15) =	53.33%
Model Y=	(6) X 100 / (15) =	40%
Model Umbrella =	(5) X 100 / (15) =	33.33%
Model knot =	(7) X 100 / (15) =	46.66%
Model COTS =	(7) X 100 / (15) =	46.66%
Model Elena =	(12) X 100 / (15) =	80%
Model PCLCM =	(15) X 100 / (15) =	100%



Percentage of events in different CBSE Models using pie chart

5. CONCLUSION:

The Proactive Component Life Cycle Model (PCLCM) represents a significant advancement in component-based software engineering, bridging gaps found in traditional CBSE models through unified validation, milestone-based customer feedback, static and dynamic automated testing, and comprehensive risk analysis. By explicitly categorizing requirements as persistent or evolving and integrating continuous market trend analysis, PCLCM ensures agility and customer alignment throughout development. The model's holistic approach not only enhances software quality and reliability but also streamlines efficiency and maintainability. It encourages proactive adaptation to market and technology changes, supporting robust component integration and long-term sustainability of software systems. Overall, PCLCM stands out as a strategic and practical framework for developing resilient, customer-centric, and future-ready solutions in the fast-changing digital landscape.

6. REFERENCES

[1] Szyperski C., (1998). Component Software, Beyond Object-Oriented Programming, ACM Press, Addison-Wesley, NJ.
[2] G.T. Heineman and W.T. Councill, editors. Component-Based Software Engineering: Putting the Pieces Together. Addison-Wesley, 2001.
[3] Bertrand Meyer, "The Grand Challenge of Trusted Components," Proc. 25th Intl Conf on Software Engineering (ICSE 2003), IEEE Computer Society Press, 2003, pp. 660–667

[4] Malcolm Douglas McIlroy, “Mass Produced Software Components,” In Peter Naur and Brian Randell (Eds.), Proc. NATO Software Engineering Conference, Garmisch, Germany, 7–11 Oct 1968. NATO Science Committee, NATO, Brussels, Belgium, pp. 88–98, Jan 1969.
[5] Grady Booch, Software Components with Ada: structures, tools, and subsystems. 3rd Edition, Addison-Wesley, 1993.
[6] Myers, G., The Art of Software Testing, Wiley, 1979.
[7] M. Sitaraman and B. W. Weide , “Special Feature Component-Based Software Using RESOLVE”, ACM SIGSOFT Software Engineering Notes 19, No. 4, 21-67, October 1994.
[8] Ivica Crnkovic, “Component-based software engineering for embedded systems”, ICSE '05 Proceedings of the 2005, 27th international conference on Software engineering, ACM New York, NY, USA.
[9] Ivica Crnkovic; Stig Larsson; Michel Chaudron, “Component-based Development Process and Component Lifecycle.” Online Available: http://www.mrtc.mdh.se/publications/0953.pdf
[10] G.Pour, M. Griss, J. Favaro, “Making the Transition to Component-Based Enterprise Software Development: Overcoming the Obstacles – Patterns for Success,” Proceedings of Technology of Object-Oriented Languages and systems, 1999, pp.419 – 419.
[11] Lata Nautiyal, Umesh Tiwari, Sushil Dimri & Shashidhar G. Koolagudi, “Component based Software Development- New Era with new Innovation in Software Development,” International Journal of Computer Applications (IJCA), vol. 51, no. 19, pp. 5-9, August 2012
[12] M. Sitaraman and B. W. Weide , “Special Feature Component-Based Software Using RESOLVE”, ACM SIGSOFT Software Engineering Notes 19, No. 4, 21-67, October 1994.
[13] Twin Peaks Model Nuseibeh, B. (2001). Weaving together requirements and architecture. IEEE Software, 18(4), 60-67. https://doi.org/10.1109/52.934466
[14] Tomar, P., & Gill, N. S. (2010). Verification & Validation of components with new X Component-Based Model. 2010 2nd International Conference on Software Technology and Engineering, 2, V2-365-V2-371. https://doi.org/10.1109/ICSTE.2010.5608737
[15] Luiz Fernando Capretz, " Y: A new Component-Based Software Life Cycle Model ", Journals of Computer Science1 (1) : pp.76-82.
[16] Dixit, A., & Saxena, P. C. (2011). Umbrella: A New Component-Based Software Development Model. International Conference on Computer Engineering and Applications, IPCSIT, 2, 33–37.
[17] Falcão, R., Jedlitschka, A., Elberzhager, F., & Rombach, D. (2024). Experiences in Using the V-Model as a Framework for Applied Doctoral Research. In Teaching Empirical Research Methods in Software Engineering (arXiv:2407.04563). https://doi.org/10.48550/arXiv.2407.04563
[18] Lau, K.-K., Taweel, F. M., & Tran, C. M. (2011). The W Model for Component-Based Software Development. 2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications, 47-50. https://doi.org/10.1109/SEAA.2011.15
[19] Rajender Singh Chhillar, Parveen Kajla, “A New Knot Model for Component Based Software Development”, International Journal of Computer Science Issues Year: 2011 Vol: 8 Issue: 3 Pp.: 480-484
[20] Christine L. Braun, “A lifecycle process for the effective reuse of commercial off-the-shelf (COTS) software” in Proceedings of SSR '99 Proceedings of the 1999 symposium on Software reusability ACM New York, NY, USA, Pp. 29-36
[21] Anurag Dixit and P.C. Saxena, “Umbrella: A New Component-Based Software Development Model” International Conference on Computer Engineering and Applications2009IPCSIT vol.2 (2011) (2011) IACSIT Press, Singapore
[22] Lata Nautiyal and Dr. Neena Gupta “Elicit - A New Component based Software Development Model.” International Journal of Computer Applications (IJCA), vol. Vol. 63 – no.- 21, pp. 53-57
[23] Sandeep Chopra et al. (2018, January). Elena – A novel component-based life cycle model. European Journal of Advances in Engineering and Technology (EJAET). ISSN: 2394-658X
[24] Chopra.S et al. (2019, February). Boundary analysis for equivalent class partitioning by using binary search. International Journal of Computer Sciences and Engineering (IJCSE), 7(2). ISSN: 2347-2693.
[25] Chopra et al. (2018, June). A novel certification process for component-based life cycle model. International Journal of Computer Sciences and Engineering (IJCSE), 6(Special Issue-5). E-ISSN: 2347-2693