

Original Article

Enhancing Software Application Efficiency Through Design-Centric Methodologies: An Empirical Evaluation

Pooja Chandrashekar

Independent Researcher.

Received Date: 16 January 2022

Revised Date: 15 February 2022

Accepted Date: 11 March 2022

Abstract: Modern software applications require designs and development processes that would make them efficient, flexible, and users' satisfying, especially in tech environments that are continuously changing. User-Centered Design (UCD), Model-Driven Development (MDD), Prototyping, Design Thinking, and Rapid Application Development (RAD) are design-focused methodologies that offer you well-structured, user-centered approaches which in turn enhance functionality, maintainability, and performance besides facilitating the flow of new ideas. These methods minimize errors in design, increase the quality of the product, and shorten the delivery time by the use of early visualization, iterative development, and continuous customer feedback. This case study decides to use Design Thinking (DT) as a tool to examine the influence of design-centric methodologies on software efficiency by the help of a few quantitative metrics like Average Efficiency, Efficiency Gain, Innovation Success, and Return on Investment (ROI). The outcomes indicate that early customer understanding, prototyping, and round-robin brainstorming could make a project 3% to 15% more efficient. In interrelated projects, the total advantages can even reach 90%. Besides, the application of human-centered design principles facilitates collaboration among people, lowers risks, and makes things more scalable and reliable. By using DT, this case study demonstrates how the connection of design methods with measurable performance gains may lead enterprises to better software development outcomes and higher creative potential.

Keywords: Design-Centric, Design Thinking, Software Efficiency, Prototyping, Software Architecture, Performance Optimization.

I. INTRODUCTION

In today's software world, efficiency is very important for programs to work well on systems, mobile, and online platforms. Software that works well speeds up execution and uses less resources. It also makes the whole user experience better, which is increasingly more crucial in competitive industries [1][2]. The developers of software have a tough challenge to keep their systems fast when the systems are under heavy load, they have to make sure that more users can be handled, and that the systems work well with different kinds of hardware as the software programs become more complex. If people aim to improve a software, then they normally think of changes in the code such as better algorithms or memory management. These ways may change the system slightly but mostly they do not consider the impact of the system architecture levels on the overall performance of the system [3].

Design-driven approaches in software engineering present a very clear and organized path for creating apps. They pretty much from the start of development are very modular, simple to maintain, and scalable. Methods such as object-oriented design [4], If done correctly, component-based architectures and established design patterns have been shown to not only make codebases cleaner but also to improve the performance of the code visually. Essentially, these methods aim at reducing the overhead, increasing the reusability, and decreasing the required computing power by analyzing the interaction of the components, the flow of the data, and the sharing of the responsibilities [5][6]. Although design-focused strategies show clear advantages, they are not always put into practice in the real world. The reason for this is that a substantial number of people believe that these strategies will require more work initially. Another considerable number of people consider that there are no fast performance benefits and that there is not enough evidence that such strategies make applications run better.

Most of the previous research has concentrated on the theory behind design methods or on very specific performance enhancements in code or algorithms. It has become quite clear from this that there is a void in understanding the impact of high-level design methods on measurable efficiency results. [7]. This difference illustrates that we need to carefully evaluate whether design-focused methodologies may lead to demonstrable gains in execution speed, memory utilization, and responsiveness in real-world applications [8].

The existing research issue is the study of the effects of design-oriented strategies on the performance of software applications. The research does this by measuring the performance of the system through a trial and a performance review to see the influence of well-planned design choices on the overall system performance. It is a source of informed guidance to software developers who are simply thinking in the direction of making applications better while as a side effect, they enable



researchers to find new ways of effective applications development. Theoretically, by design linking to performance metrics, one can argue that the study is setting out to give a few practical and easily implementable examples of how designing practice can be used for more efficient, reliable, and maintainable software systems.

A. Structure of the Paper

The paper is structured to evaluate design-centric methodologies for software efficiency. Section II discusses modern software applications and architectural challenges. Section III details design-centric approaches and its comparison. Section IV presents a case study practice quantitatively. Section V examines prior research and gaps. Section VI summarize findings and outline directions for further research.

II. SOFTWARE APPLICATION IN MODERN ARCHITECTURE

The software applications are influenced by changes in architectural paradigms that focus on scalability, efficiency and flexibility. Such architectures are meant to be able to strike a balance between functionality, reliability and user experience, and allow a wide range of operational environments. They also accommodate issues concerning performance, maintainability as well as security so that the software systems are tough and receptive to adapt to changing demands in dynamic technological environments.

A. Overview of Software Architectures

Software architecture has undergone significant transformations, driven by evolving business requirements, developments in computer paradigms and the extensive use of cloud computing, as seen in Figure 1.

a) Monolithic Architecture

A monolithic architecture is a conventional method in which the user interface, business logic, and data access layers of an application are all constructed as a single, cohesive codebase. All functionalities are housed within a single deployment unit in this design, which has a closely linked structure [9]. Although it is organized as modules, a typical monolithic program is compiled, deployed, and run as a single unit.

b) Microservices Architecture

An application can be broken down into a number of loosely linked services, each of which is in charge of a distinct business function, using the microservices architectural technique [10]. Microservices function independently and communicate using well defined APIs, in contrast to monolithic programs, which have all of its components housed in a single codebase. Generally speaking, microservices adhere to domain-driven design (DDD), in which every service is matched with a particular business capability.

c) Serverless Architecture

Serverless computing is an architectural model that shifts management of the infrastructure away from the developers and thus they can focus on building code rather than setting up or maintaining servers. As per this model, cloud providers adjust the activities on the fly as per the requirement, they allocate resources automatically and run the code that is triggered by events [11]. Serverless computing is a resource and cost-saving mechanism that only performs the code when it is requested, unlike a traditional setup where applications are running on servers that are always on.

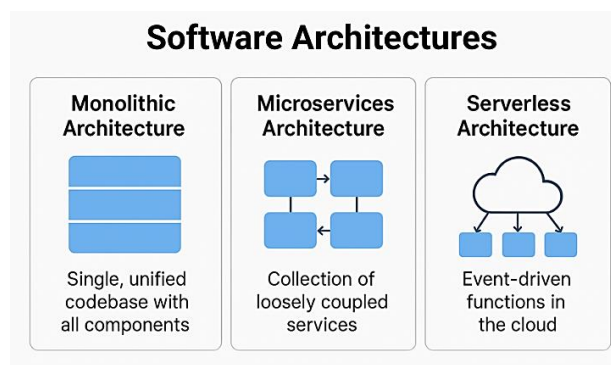


Figure 1 : Software Architecture Models

B. Key Characteristics of Modern Software Architectures

Modern software architectures are evaluated and designed based on several quality attributes that determine how effectively a software system performs under real-world conditions:

- **Functionality:** The capacity of the software product to do tasks that satisfy explicit and implicit demands when utilized in a certain way (i.e., how the software meets needs).

- Reliability: The software product's capacity to continue operating at its current level for a certain amount of time under specified conditions [12].
- Usability: The software product's capacity to be comprehended, learnt, used, and appealing to the user under certain circumstances (the effort required for utilization).
- Efficiency: The software product's ability to provide suitable performance based on the resources used in specific situations.
- Maintainability: The software's ability to change. Corrections, improvements, or adjustments to the program in response to shifts in the environment, needs, and functional specifications are examples of modifications. This is the effort needed to keep it updated.
- Portability: The software product's ability to adjust to different environments. The surroundings can include hardware, software, or organizational elements.

C. Challenges in Modern Applications

Modern software applications are encountering various efficiency problems. These issues can influence performance, scalability, and maintainability:

- Performance Bottlenecks: Slow performance of applications might be due to inefficient algorithms or database queries, or network latency [13]. The issues become worse with high user loads, resulting in decreased user experience and system throughput.
- Scalability Limitations: The applications must be capable of handling increased workloads efficiently. Typically, monolithic systems are difficult to scale horizontally, while distributed architectures require tight service coordination and resource management.
- Maintainability Problems: The tighter the code is coupled or the less its published documentation is, the harder it becomes to update and fix bugs [14]. High maintenance rates may decrease the development time and cause technical debt in the long run.
- Resource Usage: CPU, memory, or storage usage can be excessive and slow down performance and increase the cost of operation. Efficient utilization of resources guarantees uniform efficiency of the system.
- Security and Reliability Constraints: Security features and fault-tolerance mechanisms need to be implemented, and this may increase overhead in terms of performance. The issue of security, reliability, and efficiency is one of the problems to be considered in the context of contemporary software systems.

III. DESIGN-CENTRIC SOFTWARE APPROACHES

Design-oriented software development solutions are oriented towards developing solutions that are aligned with the needs and business objectives of the users. They also focus on the iterative development, early visualization and stakeholder review to create a functional, intuitive, and user-friendly software. These measures increase the quality of products, speed up their development, and make them competitive in the market through collaboration, promotion of innovation and adaptation to the changing demands.

A. Process Flow of Design-Centric Methodology

The process flow of the design-based software development approach is concerned with a user-oriented approach to the development based on an iterative development that aligns the software development with the business objectives and user requirements [15]. It integrates stakeholder modeling, visualization and feedback to render solutions functional, cohesive and user friendly. The objective of this approach is to enhance efficiency and effectiveness at the software development life cycle. It also promotes innovation and continual improvement based on the responsiveness to the changing needs. This process flow is demonstrated in Figure 2.



Figure 2 : Process Flow of Design Centric Methods

a) *Understand the Problem*

This step entails the study of business objectives and user requirements so as to have a clear vision of the organizational objectives and user expectations. It is also about the opportunities of design innovation in which creative ideas have the potential of enhancing functioning, effectiveness, and user experience.

b) *Conceptual Design & Modeling*

In this case, the high-level design concepts, user journeys and process flows are established to see how the users interrelate with the system. Components communication, dependencies and touchpoints are graphically modelled to enable the system to be cohesive and functional.

c) *Prototype & Visualization*

This step brings the design concepts into visual representation, in the form of flow charts, wireframes or mockups. Such images are useful in supporting assumptions based on stakeholder feedback and preliminary testing to ensure that solutions meet user and corporate needs.

d) *Workflow & Process Realization*

This phase transforms the conceptual designs into workflows that can be executed in order to specify the operations of the systems. The specifications of rules, services, and interactions are given to guarantee the effectiveness of working and performing smoothly in the real world.

B. Classification of Design Centric Approaches

This section gives a design-centric approach classification, including what they focus on, their products, and their benefits.

a) *User-Centered Design (UCD)*

UCD is a design-based approach that bases software development on the actual requirements, tastes, and habits of end users. According to Donald Norman, the User-Centered Design (UCD) concept is founded on the requirements and preferences of the user, with a focus on making things intelligible and useful [16]. Nonetheless, one popular method of making sure that users' wants and interests are being satisfied is to include them in the User-Centered Design process.

b) *Model-Driven Development (MDD)*

This approach focuses on the design of abstract models of the software system as the main artifact. These models describe system structure, behavior as well as interactions and code is frequently generated automatically out of them. It guarantees that there is consistency in designing [17], minimizes the errors in code, and enables the designers to concentrate on the larger picture of the system as opposed to the finer details of the implementation.

c) *Prototyping Model*

The Prototyping Model is a design-oriented software development framework which means developing a working prototype at the beginning of the process to visualize the requirements and design-concepts. Prototypes allow the stakeholders to get to know how it works, give feedback and also polish the design to a better best. It minimizes wrong interpretations, clears out the need, and enables the development team to identify possible design flaws in time.

d) *Design Thinking*

Design Thinking is a design-driven, human-centered methodology that encourages originality and creativity in solutions. It focuses on the compassion to users clearly stating problems and brainstorming of various design solutions, developing prototypes, testing them and determining their efficiency. This strategy makes the software solutions to be not only practical but also user-friendly and leads to innovation [18].

e) *Rapid Application Development (RAD)*

RAD is a design-based methodology that emphasizes rapid development through user involvement and iterative prototyping. Design stage is highlighted to develop reusable parts and prototypes so that the delivery of working software can be done in a fast manner. With the constant feedback, RAD makes the software design to be highly related to the user requirements and it is also of high quality.

The Table 1 provides a comparative analysis of design-based software development solutions, their aim, interaction with users, output, and main advantage. It is a brief source of information on how various strategies increase software efficiency and usability

Table 1 : Comparison of Design-Centric Software Development Methodologies

Aspect	User-Centered Design (UCD)	Model-Driven Development (MDD)	Prototyping Model	Design Thinking	Rapid Application Development (RAD)
Focus	Real needs, preferences, and behaviors of end-users	Abstract models of system structure, behavior, and interactions	Early working prototype to visualize requirements and design	Human-centered, creative problem solving and innovation	Fast development using iterative prototyping and reusable components
User Involvement	High; users actively involved to ensure needs are met	Low to moderate; focus is on model accuracy rather than direct user input	Moderate; users provide feedback on prototypes	High; users' problems and feedback drive ideation and testing	High; users provide continuous feedback for iterative improvement
Advantages	Ensures usability and satisfaction; aligns with user needs	Reduces coding errors; ensures system consistency; focuses on high-level design	Minimizes misunderstandings; identifies design flaws early; clarifies requirements	Encourages innovation and creative problem-solving; improves user experience	Quick delivery; reusable components; closely aligned with user requirements
Deliverables	Usable and understandable software	Consistent, model-based software, often with code generation	Working prototypes evolving into final product	Innovative and user-friendly solutions	Fast, functional software with iterative improvements

C. Effect of Design-Centric Approaches

Design-based approaches are significantly influential on the outcome of software development. Their influence may be summed up in several aspects, depicted in the Figure 3:

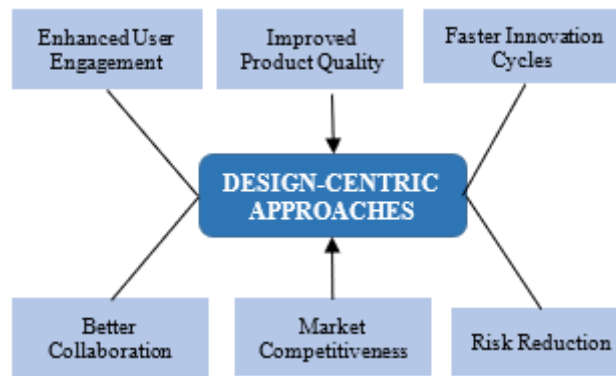


Figure 3 : Impact of Design Centric Approaches

- **Enhanced User Engagement:** Design-centric methodologies are very effective because they emphasize the actual user requirements, behavior, and preferences to keep the software solutions this way, so that they become intuitive, usable, and in tandem with the expectations of the users. This enhances acceptance and satisfaction.
- **Improved Product Quality:** The design mistakes and misinterpretations are minimized at the initial stages of the development process [19], and as a result, more stable and high-quality software is created through iterative prototyping and regular feedback of the stakeholders.
- **Faster Innovation Cycles:** Promoting experimentation and creative problem-solving help the teams to investigate new ideas and come up with new features and capability that allow products to be distinguished in the market.
- **Better Collaboration:** The design-based approach supports cross-disciplinary collaboration between developers, designers, and business analysts, which enhances communication and a common purpose and objective.

- Market Competitiveness: The design-centric approach guarantees that the software products created via design-based methods satisfy the expectations and are more likely to win the competition because the usability, aesthetics, and functionality are considered right at the start [20].
- Risk Reduction: Early visualization with prototypes and mock-ups enables the detection of possible design failures or areas that may cause dissatisfaction to the user prior to full scale implementation so that the development risks and costs are minimal.

IV. CASE STUDY: QUANTITATIVE EVALUATION OF DESIGN THINKING PRACTICES FOR PROJECT EFFICIENCY

The case study focuses on the application of design-oriented approaches, specifically Design Thinking (DT), to make projects more efficient as part of an innovation process. Human-centered solutions, such as DT, are also becoming more important in organizations to generate creativity, decrease risk, and enhance project outcomes as a whole. DT focuses on the iterative resolution of problems, quick prototyping and close interaction with the users, making sure that solutions are directly related to the real-world requirements. The current investigation is aimed at determining the effects of particular DT behaviors on project efficiency, as the percentage of meeting the goals on time, on budget, and of the desired quality. The case study empirically estimates the practical value of using DT in the innovation and software development initiatives by quantifying such impacts.

A. Strategy

The strategy is a laid down plan of action where the targeted results are the realization of certain long-term goals, and where there is a concentration of resource advancement as well as foreseeing difficulties so that the best results are attained as illustrated below:

a) Data

The data was collected regarding various projects on innovation by different organizations that actively used DT methodologies. To establish a comparative premise, two groups of projects were selected: those who used structured forms of DT practices and those who had little or no integration of DT.

b) Approach

Design Thinking is a design-cantered yet people-focused approach which focuses on empathy, creativity, and iterative problem-solving. It deals with the awareness of user needs and converting it into a novel and viable solution [21]. It has a role in improving the efficiency of the project and increasing collaboration, preventing risks, and making sure that design-driven solutions reflect the real-world challenges and organizational objectives.

c) Performance Measurement

These metrics are applied to determine the level of efficiency and project value delivery efficiency.

- Average Efficiency (%): Indicates the total project goal accomplishment in terms of time, cost and quality efficiency [22].
- Efficiency Gain: The indicator of the increased efficiency in comparison with baseline (non Design Thinking) projects, expressed in Equation (1):

$$Efficiency\ Gain = \frac{E_{DT} - E_{base}}{E_{base}} \times 10 \quad (1)$$

- Return on Investment (ROI %): Measures the economic performance or profitability of projects with the application of the methods.

B. Quantitative Findings

The table indicates the effect of six major Design Thinking (DT) practices on the project performance, in terms of average efficiency, efficiency improvement over baseline, rate of innovation success, and ROI. It offers hard data on the contribution of every practice towards improving the efficiency of software applications. The performance is presented in Table 2.

Table 2 : Quantitative Impact of Design Thinking Practices on Project Success

Design Thinking Methodology	Avg. Efficiency (%)	Efficiency Gain (%)	Innovation Success (%)	ROI (%)
User Needs Discovery	88	+15	85	120
Problem Comprehension	85	+12	82	110
Assumption Challenge	78	+5	75	95
Problem-Solution Navigation	75	+3	70	90
Ideation via Visualization	80	+8	78	105
Experiential Learning	82	+10	80	108

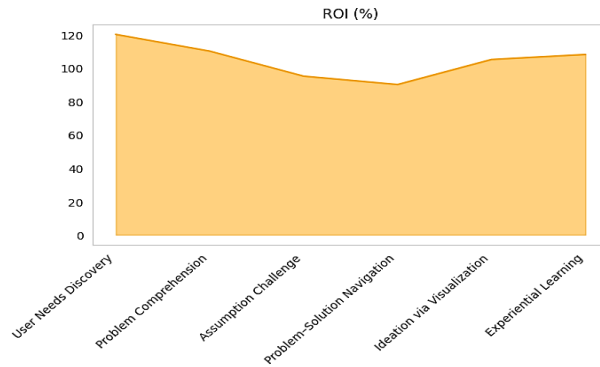


Figure 4 : ROI Impact of Design Thinking Practices on Project Efficiency

Figure 4 shows the influence of various techniques of Design Thinking on the efficiency of a project. The x-axis depicts six major steps, including User Needs Discovery, Problem Comprehension, Assumption Challenge, Problem, Solution Navigation, Ideation through Visualization and Experiential Learning. The y-axis indicates the Return on Investment (ROI) in per cent, with a start range of 0 and the maximum of 120 percent. The colored orange shows that there is a gradual drop between User Needs Discovery and Problem, Solution Navigation. It is then moderately increased by Ideation and Experiential Learning. This tendency implies that the initial steps of perception are followed by high ROI, and creative and experiential phases contribute to the efficiency enhancement and maintenance.

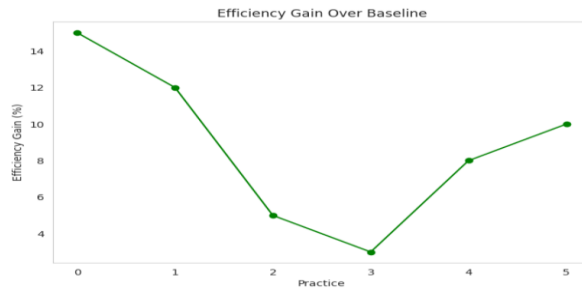


Figure 5 : Efficiency Gain Over Baseline Through Design Thinking Practices

Figure 5 demonstrates that the Design Thinking methods could potentially make projects more efficient. The x-axis is the various practices and the y-axis indicates the percentage of Efficiency Gain (%). The green line trend has a high efficiency gain of approximately 14%. It is slowly diminishing by the following practices to its lowest point at around 3%. Then it reappears to approximately 10%. This trend indicates that the initial phases of design thinking contribute hugely to efficiency. Nonetheless, there is a momentary slump as a result of mid-stage difficulties. Late practices display new development since ideas evolve and experience improves project outcomes.

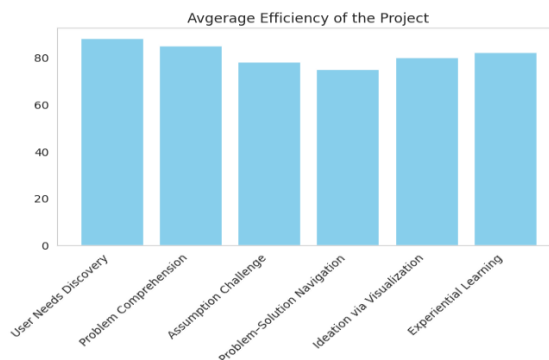


Figure 6 : Average Efficiency of the Project based on Design Thinking Practices

Figure 6 presents the performance of six major Design Thinking practices, such as User Needs Discovery, Problem Comprehension, Assumption Challenge, Problem-Solution Navigation, Ideation via Visualization, and Experiential Learning. It has been found that User Needs Discovery and Problem Comprehension had the most efficient results. This shows that it has a good knowledge and awareness of the user requirements. Assumption Challenge and Problem-Solution Navigation had a slightly lower score, and it indicates that critical assessment and mapping solutions should be improved. Ideation through Visualization and Experiential Learning presented balanced results, with the creative engagement and hands-on learning being involved in the overall project efficiency.

a) *Key Insights*

Design-centric and human-centered practice leading to application has a great impact in increasing efficiency of the software project, decreasing errors and improving the results of innovation.

- **User-Centric Focus:** The activities like identification of user need and problem solving greatly enhanced the efficiency of software project (1215% increase) and hence the need to empathic research.
- **Iterative Development Ideation:** Visualizations and Learning Experimentation The iterative development process cut down rework and errors and added 8-10 percent efficiency gains.
- **Combined Methodologies:** The integration of several practices of Design Thinking resulted in cumulative efficiency, and the overall efficiency of projects was as high as 90 percent.
- **Innovation and ROI:** Design-centric, human-centered practices increased the success rates of innovations and created more ROI that showed objectively the value in the development of software applications.

b) *Benefits*

The most significant benefits of the use of the design-centric approach to the improvement of the efficiency of software applications, the innovations, and the outcomes of the projects are given:

- **Optimized Software Efficiency:** The design-oriented approach enhances software project performance in terms of delivery in good time, minimize delays and improve quality of applications.
- **Empirical Support of User-Centric Approach:** The methodology achieves end-user requirements in solutions by applying end-user information, which the study demonstrated in a quantitative manner with increased efficiency.
- **Reduce Risk:** Early cycle prototyping and experimentation enables faulty design and functional defects to be identified and minimizes the errors, development cost, and increases the reliability.
- **Improved Software Design Innovation:** The innovative nature of the methodology supports the emergence of new features and functionality and increases the rate of innovation success in the study.

V. LITERATURE REVIEW

The literature presents design-based solutions in software, PSSs, branding, and engineering, where emphasis is laid on usability, efficiency and performance of the system. Research demonstrates the worth of empirical analysis and user-friendly design.

Ismail et al. (2021) conducted a study on the design and usability of a user-centered travelling application. Comparing the performance of two prototypes one web-based and the other mobile on various platforms was the goal of the study. The simplicity of the two prototypes was measured with the System Usability Scale (SUS) questionnaire. The SUS values for the two prototypes did not differ statistically significantly, suggesting that participants had no preference for one over the other. In order to ensure a well-organized and pleasurable journey, this study emphasizes the need of creating travel applications that meet users' wants and requirements [23]

Zhang et al. (2021) present a proposal of an approach for developing PSS that is sustainable and based on the design-centric complexity (DCC) theory. They also analyze the complexity of the system during the design of an early stage, setting the type and issues of complexity. A combination of the sub-field model of TRIZ which makes the system less complex can lead to the reduction of the problems to a problem of a system. This minimizes conflicts and potential design attribute problems, thereby increasing the likelihood of meeting the functional requirements (FRs) of PSSs. This provides stability and sustainability of the system in the long run. The viability of the framework is confirmed by a case study of a bicycle sharing service and management system.[24]

Nakhi (2020) studies underline the significance of branding in the process of conveying different identities, personalities, and values. The changing communications, technology, and culture are a challenge to the brands. Branding requires a significant means of tool, such as design, which is supposed to produce recognition, memorability and loyalty. There are other brands that are concentrating on people-oriented design culture, which supports imagination and collaborations. Design-based brands, such as Google, Apple, Puma, and Uber, perform better than others do due to their stronger relationships with consumers. The study proposes a road to Design Centricity as a branding strategic approach.[25]

Varadarajan (2020) highlights the importance of systems, design, and strategic thinking for engineers in the future. It emphasizes the need for a more insightful understanding of these concepts and their effects. He talks about his experience of applying systems thinking to an engineering design-based program in India and introduces a complexity-based metric to gauge these abilities' growth. It outlines the strategy used and a complexity-based metric to monitor the growth of systems thinking proficiency[26]

Taylor (2019) work on the focus on architecture and design as the main pillars is made in the work on software development. He gives a clear definition of architecture, outlines the important methods of designing and explains their usage

in the influential applications. The chapter talks about ways of analyzing architectures, with a special focus on the importance of connectors in complex systems. It further discusses trade-offs of cost-benefit, highlights the need to keep concepts clean, and examines the way forward in the field.[27]

Scott (2018) This study uses both historical models and the recently developed approach to do a multi-fidelity analysis of aircraft acquisition costs on a range of vehicles. The study examines the variations in insight produced by the various levels of fidelity as well as the overall accuracy of the various techniques. The comparison's findings are qualitatively analyzed in light of upcoming aircraft development to draw conclusions about cost that are relevant to current design trends and performance standards[28]

The Table III presents the most important research gaps in design-centric approaches, including available findings, methods, limitations, and opportunities, to offer a systematic perspective of enhancing the efficiency of software

Table 3 : Research Gaps in Design-Focused Methods for Optimizing Software Efficiency

Reference	Aspect / Theme	Research Findings	Methods Used	Limitations	Research Gap
Ismail et al., (2021)	Software Efficiency	Focus on performance optimization, memory usage, response time	Benchmarking, profiling, system monitoring	Mostly technical, ignoring user-centered design impacts	Need integration of technical efficiency and design-driven factors
Zhang et al., (2021)	User-Centered Design (UCD)	UCD improves usability and user satisfaction	Prototyping, SUS surveys, interviews	Focuses on usability alone, not system performance	Empirical evaluation of UCD's impact on software efficiency is limited
Nakhi, (2020)	Design Thinking / Design-Centric Approaches	Applied in innovative interface design and software development	Workshops, iterative design, case studies	Lack of standardized evaluation metrics; mostly qualitative	Quantitative evaluation of design-centric methodologies on efficiency is scarce
Varadarajan, (2020)	Multi-Platform / Cross-Environment Performance	Compares web vs. mobile apps or cloud vs. on-premise	Comparative benchmarking, platform testing	Focuses on platform performance rather than design influence	Need for studies examining design methodology's effect across platforms
Taylor, (2019)	Integration of Human & Technical Factors	Addresses either technical optimization or user experience separately	Surveys, usability tests, performance profiling	Fragmented approaches; no holistic framework	Gap in frameworks combining human factors, design methods, and technical efficiency
Scott, (2018)	Quantitative Metrics & Evaluation	Metrics like response time, memory usage, SUS, NPS, task success	Benchmarking, surveys, A/B testing	No standardized composite metric for design-centric efficiency	Opportunity to define metrics linking design-centric methods to measurable software performance gains

VI. CONCLUSION AND FUTURE WORK

The software development, achieving high efficiency, reliability, and innovation, requires methodologies that integrate user-focused design with technical rigor. The case study about Design Thinking (DT) shows a tremendous influence of design-oriented techniques on the work of software projects. Comparing UCD, MDD, Prototyping, RAD, and DT, it can be found out that iterative design, early visualization, and active user engagement are the best options to influence the outcomes. The study results in individual Design Thinking practices improving efficiency by 3-15 percent using quantitative measures (Average Efficiency, Efficiency Gain, Innovation Success, and ROI). The combination of these practices may increase the efficiency of the whole project up to 90. The User Needs Discovery and Problem Comprehension activities delivered the largest returns and the iterative prototyping and experiential learning activities supported the continued performance enhancement. The design-oriented practices also minimized errors, decreased risks, promoted collaboration among various departments, and used resources more effectively. This guaranteed quality and timely delivery software. The work in the future will focus on the

development of a unified model that will combine both design-based approaches with technical optimization strategies to perform a comprehensive evaluation of efficiency, usability and scalability of multi-platform and enterprise-wide systems. The quantitative evaluation metrics will be standardized, which will facilitate benchmarking and comparison of different projects. Extending this research will further strengthen the strategic role of Design Thinking as a key driver of innovation, project performance, and sustainable software development excellence.

VII. REFERENCES

- [1] L. Thomas et al., "Distance software: design and analysis of distance sampling surveys for estimating population size," *J. Appl. Ecol.*, vol. 47, no. 1, pp. 5-14, Feb. 2010, doi: 10.1111/j.1365-2664.2009.01737.x.
- [2] G. Casale et al., "Current and Future Challenges of Software Engineering for Services and Applications," *Procedia Comput. Sci.*, vol. 97, pp. 34-42, 2016, doi: 10.1016/j.procs.2016.08.278.
- [3] Y. Renard et al., "OpenViBE: An Open-Source Software Platform to Design, Test, and Use Brain-Computer Interfaces in Real and Virtual Environments," *Presence Teleoperators Virtual Environ.*, vol. 19, no. 1, pp. 35-53, Feb. 2010, doi: 10.1162/pres.19.1.35.
- [4] C.-H. Ko, T.-C. Chang, Y.-H. Chen, and L.-H. Hua, "A design-centric approach for augmented reality collaboration," *Int. J. Eng. Educ.*, vol. 28, no. 4, p. 811, 2012.
- [5] J. Villalobos and O. González, "A curricula model for supporting a design-centric computing engineering education," *Int. J. Eng. Educ.*, vol. 28, no. 4, p. 880, 2012.
- [6] A. R. Lyon and K. Koerner, "User-centered design for psychosocial intervention development and implementation.," *Clin. Psychol. Sci. Pract.*, vol. 23, no. 2, p. 180, 2016.
- [7] A. Freddi and M. Salmon, "Design principles and methodologies," Cham Springer, pp. 159-180, 2019.
- [8] K. K. Rao, G. Raju, and S. Nagaraj, "Optimizing the software testing efficiency by using a genetic algorithm," *ACM SIGSOFT Softw. Eng. Notes*, vol. 38, no. 3, pp. 1-5, May 2013, doi: 10.1145/2464526.2464539.
- [9] R. Kumar, "Architectural Crossroads: Navigating Monolithic, Microservices, Serverless, and Event-Driven Systems," *J. Softw. Eng. Simul.*, vol. 7, no. 7, pp. 30-41, Jul. 2021, doi: 10.35629/3795-07073041.
- [10] N. Dragoni, I. Lanese, S. T. Larsen, M. Mazzara, R. Mustafin, and L. Safina, "Microservices: How to make your application scale," in *International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, 2017, pp. 95-104.
- [11] B. R. Cherukuri, "Serverless revolution: Redefining application scalability and cost efficiency," *World J. Adv. Res. Rev.*, vol. 2, no. 30, pp. 039-053, Jun. 2019, doi: 10.30574/wjarr.2019.2.3.0093.
- [12] F. Losavio, L. Chirinos, N. Lévy, and A. Ramdane-Cherif, "Quality characteristics for software architecture," *J. object Technol.*, vol. 2, no. 2, pp. 133-150, 2003.
- [13] E. M. Maximilien and P. Campos, "Facts, trends and challenges in modern software development," *Int. J. Agil. Extrem. Softw. Dev.*, vol. 1, no. 1, p. 1, 2012, doi: 10.1504/IJAESD.2012.048305.
- [14] D. B. Rawat and S. R. Reddy, "Software defined networking architecture, security and energy efficiency: A survey," *IEEE Commun. Surv. & Tutorials*, vol. 19, no. 1, pp. 325-346, 2016.
- [15] R. Wieringa, *Design science methodology for information systems and software engineering*. Springer, 2014.
- [16] D. Norman, *The design of everyday things*. Basic books, 2013.
- [17] Á. Domingo, J. Echeverría, Ó. Pastor, and C. Cetina, "Evaluating the Benefits of Model-Driven Development," in *International Conference on Advanced Information Systems Engineering*, 2020, pp. 353-367. doi: 10.1007/978-3-030-49435-3_22.
- [18] S. K. D. Dwivedi, S. Upadhyay, and A. K. Tripathi, "A working framework for the user-centered design approach and a survey of the available methods," *Int. J. Sci. Res. Publ.*, vol. 2, no. 4, pp. 12-19, 2012.
- [19] L. Waidelich, A. Richter, B. Kolmel, and R. Bulander, "Design Thinking Process Model Review," in *2018 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*, IEEE, Jun. 2018, pp. 1-9. doi: 10.1109/ICE.2018.8436281.
- [20] J.-J. Lee, "The True Benefits of Designing Design Methods," *Artifact*, vol. 3, no. 2, p. 5, Aug. 2014, doi: 10.14434/artifact.v3i2.3951.
- [21] J. C. Pereira and R. de F. S. M. Russo, "Design Thinking Integrated in Agile Software Development: A Systematic Literature Review," *Procedia Comput. Sci.*, vol. 138, pp. 775-782, 2018, doi: 10.1016/j.procs.2018.10.101.
- [22] A. Tang and H. van Vliet, "Design Strategy and Software Design Effectiveness," *IEEE Softw.*, vol. 29, no. 1, pp. 51-55, Jan. 2012, doi: 10.1109/MS.2011.130.
- [23] N. A. Ismail et al., "User-centred Design and Evaluation of Web and Mobile based Travelling Applications," *Int. J. Adv. Comput. Sci. Appl.*, vol. 12, p. 2021, 2021, doi: 10.14569/IJACSA.2021.0120854.
- [24] P. Zhang, S. Jing, Z. Nie, B. Zhao, and R. Tan, "Design and Development of Sustainable Product Service Systems Based on Design-Centric Complexity," *Sustainability*, vol. 13, no. 2, 2021, doi: 10.3390/su13020532.
- [25] R. Nakhi, "Design Centricity as a powerful Branding strategy," *Fac. to Fact.*, 2020.
- [26] S. Varadarajan, "Measuring the value of systems thinking for design-centric engineering education," *Proc. Des. Soc. Des. Conf.*, vol. 1, pp. 1835-1842, May 2020, doi: 10.1017/dsd.2020.72.
- [27] R. N. Taylor, "Software architecture and design," in *Handbook of Software Engineering*, Springer, 2019, pp. 93-122. R. Scott, "A design-centric evaluation of multi-fidelity cost modeling approaches," 2018