Received: 26 June 2022 | Revised: 18 August 2022 | Accepted: 13 December 2022

DOI: 10.1049/sfw2.12085

IET Software

ORIGINAL RESEARCH

A model for developing dependable systems using a component-based software development approach (MDDS-CBSD)

Hasan Kahtan[1] | **Mansoor Abdulhak**[2,3] | **Ahmad Salah Al-Ahmad**[4] | **Yehia Ibrahim Alzoubi**[4]

[1]Cardiff School of Technologies, Cardiff Metropolitan University, Cardiff, UK

[2]KNOWBIS Solutions & Consultancy, Selangor, Malaysia

[3]University Prince Mugrin, Madinah, Saudi Arabia

[4]Management Information Systems Department, American University of the Middle East, Kuwait, Kuwait

Correspondence
Hasan Kahtan, Cardiff School of Technologies, Cardiff Metropolitan University, Llandaff Campus, Western Avenue, Cardiff CF5 2YB, UK.
Email: hkahtan@cardiffmet.ac.uk

Abstract
Component-based software development (CBSD) is an emerging technology that integrates existing software components to swiftly develop and deploy big and complex software systems with little engineering effort, money, and time. CBSD, on the other hand, has difficulties with security trust, particularly dependability. When a system provides the desired outcomes while causing no harm to the environment, it is said to be dependable. Dependability encompasses several attributes, including availability, confidentiality, integrity, reliability, safety, and maintainability. Developing dependable component software is achieved by embedding dependability attributes in CBSD. Thus, the CBSD model must address the dependability attributes. Hence, the objectives of this work are: (1) to propose a model for developing a dependable system using component-based software development approach (hereafter the model is referred to as MDDS-CBSD), which aims to mitigate software component vulnerabilities, and (2) to assess the proposed model. The best-practice method was used to frame the CBSD architecture phases and processes, as well as embed the six dependability attributes. The MDDS-CBSD architecture was evaluated using expert opinion. The MDDS-CBSD was also used to develop an information and communications technology (ICT) portal using an empirical study method. Vulnerability Assessment Tools were used to assess the developed ICT portal's dependability. The MDDS-CBSD may be used to create web application systems and to protect them from attacks. Model developers may use CBSD to describe and assess dependability attributes at any point during the model development process. The reliability of this model can also let companies utilise CBSD with confidence.

KEYWORDS
component-based software development, dependability attributes, software architecture, software development management, software engineering, software security development

1 | INTRODUCTION

CBSD is an emerging technology that builds systems by combining existing software components. CBSD focusses on composing software systems rather than programming software. Assuming that certain parts of a large software system recur frequently, common parts must be written once and reused many times rather than written repeatedly [1, 2]. At the same time, CBSD has a number of advantages, from increasing programmer productivity to reducing software development costs [3]. Software reuse can help meet demands for faster delivery, lower software production and maintenance costs, and better quality. Thus, CBSD's main goal is to reduce overall software development costs. That is, software development and maintenance must be less costly. CBSD is also required for faster software delivery. For this reason, the software industry has adopted CBSD to rapidly build and deploy large, complex software systems with minimal engineering effort. The software must fit within a competitive market window. It also aims to produce high-quality software that meets process requirements with minimal failures [4–10].

Using CBSD in software development, however, has its drawbacks. Moradian and Håkansson [11] claim that interdependencies between software components cause issues during integration. Thus, early in the CBSD cycle, dependability attributes of software components must be considered. Furthermore, pervasive computing raises serious questions about current development models' reliability. Because CBSD is a software engineering approach, its reliability is unknown [12].

A component may also fail to meet an application's requirements due to differences in requirements and cycles [13–15]. First, changes at the application level (e.g. modifying a few components or updating new versions) can cause system failure [16–19]. Second, reusing defective components may erode software system trust. So critical systems, such as military systems, require an exceptional software development process [20]. This process' main goals are to validate the system's implementation's consistency with its requirements [21–23].

2 | CBSD APPLICATIONS: RELATED WORK

The CBSD approach emerged in the late 1990s, when reusable components were incorporated into development processes. The component base is the basic element of the current software system, and CBSD is a technique that uses existing software codes [24]. With this technique, software applications need not be developed from scratch [25]. This technique also facilitates the assembly of software applications using reusable software codes, thereby improving time and budget constraints on software development [6]. CBSD is widely used by middleware platforms and tools and has become the mainstream for current software development. CBSD design for distributed networks (including the Internet) has promoted e-commerce and may expand business markets considerably. Furthermore, CBSD utilises software components that are easier to produce and more pervasive than ever before [24, 26, 27].

Unlike traditional software development approaches, CBSD offers a range of benefits and manages complex systems [24]. Software systems consist of a series of components into which software functions and non-functional properties are implemented separately. CBSD promotes the employment of effective specialists who can develop reusable components within the scope of their expertise instead of application specialists who perform the same type of work on different projects [25]. The approach also reduces the time and effort needed to develop software [26]. Moreover, CBSD facilitates the development of components that are independent of specific applications and improves the reusability of components. Software system developers can thus maximise existing structures and components, thereby improving the efficiency of software development. CBSD also generates a repository of components that supports software system development by providing reusable and tested components [6, 24, 25, 27, 28].

CBSD likewise increases the productivity of programmers. Constantly rewriting codes is an inefficient process because programmers can write and document only limited lines of code per day. With CBSD, programmers can utilise the interactive development environment (IDE) to assemble components in the desired programme. Therefore, many lines of code can be written each day and productivity is enhanced. Owing to the significant number of economic benefits gained, CBSD is an ideal approach to building software systems [6, 24, 26, 27]. Table 1 presents the application of CBSD in different domains.

3 | MOTIVATION

Despite widespread industry adoption and academic research publications, CBSD lacks formal foundations for non-functional requirement specification, composition, and verification. As a result, current CBSD practices do not support the development of reliable systems [21–23]. To build a reliable and secure system, dependability must be built into the CBSD process. A system is dependable when it produces the intended results with no adverse effects on the intended environment. Availability, confidentiality, integrity, reliability, safety, and maintainability are all attributes of dependability [31, 32].

Adding dependability attributes to the software component development process is difficult enough, but evaluating their dependability is even more difficult due to the complexity of the operational environment and the need to specify dependability attributes early in the software component development process. The research community is still working on establishing a reliable scale to measure software component dependability. Traditionally, dependability attributes such as reliability, safety, and integrity are treated as afterthoughts, implemented after the software component is developed. Thus, evaluating component-based software for dependability attributes is critical in determining system dependability [8, 33–38].

This paper proposes a model for developing a dependable system that mitigates software component vulnerabilities. This study refers to a model for developing a dependable system using a component-based software development approach, as an (MDDS-CBSD). The six dependability attributes are embedded in the CBSD architecture phases and processes for the MDDS-CBSD development. The CBSD gap analysis in Ref. [32], the awareness survey in Ref. [38], and the vulnerability assessment on selected web applications in Ref. [36] all inspired the design of MDDS-CBSD. The MDDS-CBSD is also used to develop web application systems. Finally, this paper evaluates the developed MDDS-CBSD on dependability.

4 | MDDS-CBSD METHODOLOGY

The MDDS-CBSD development methodology has three phases: Phase 1: Problem identification and gap analysis, Phase 2: MDDS-CBSD development, and Phase 3: MDDS-CBSD

TABLE 1 CBSD applied to different domains

Domains	Description
Cloud computing, autonomous driving [28]	The authors present an adaptive software component-based middleware. This research presents a component-level offloading technique for autonomous driving software based on software component allocation optimisation. Autonomous driving software components can be optimally deployed to specific computing units using software component-based allocation optimisation.
Internet of Things (IoT) [29]	This study investigates how component-based software can be used to define IoT systems. This study will aim to identify common IoT system traits and analyse how well these properties relate to component-based software characteristics.
Industrial Control Systems (ICS) [3]	Based on a revolutionary component model, this paper proposes an Open Software Architecture for Industry. The goal is to provide a component-based solution for Industrial Control Systems as well as an open framework for multiple application components and multi-vendor collaboration.
Heritage content [5]	Using the capabilities of component-based software, the paper creates a methodology for developing heritage content made of various components.
E-commerce [4]	The goal of this project is to create a standard framework for e-commerce application development that is component-based and loosely connected, but highly integrated.
Computer Numerical Control (CNC) systems [6]	This study uses component-based software and the dependency inversion principle to create a revolutionary open CNC application that allows CNC capabilities to be customised.
E-educational system [30]	This article proposes to investigate component-based software development as a future alternative for an e-educational system.

evaluation. Figure 1 shows the flowchart of the development methodology's activities and deliverables.

Phase one is a preliminary study. A component-based software development approach, software dependability attributes, and evaluation methods are discussed in the preliminary study. There was a gap analysis of the current literature on CBSD models relating to dependability features and a gap analysis of the current issues on CBSD models relating to dependability features.

The goal is to understand the existing CBSD process models to identify the features offered by each model. Each model is evaluated for strengths and weaknesses. Also highlighted is the gap analysis of existing CBSD models. This research paved the way for the MDDS-CBSD process. To derive the features for the MDDS-CBSD process, strengths are retained, and weaknesses are removed.

To overcome the dependability issues, a thorough analysis of existing research has been conducted for the software dependability domain. Several attributes, such as dependability, trustworthiness, and survivability, were identified as interchangeable terms for software security. Analysts found that dependability attributes cure security threats, abnormal behaviour, and untrustworthy issues in software systems [39, 40]. In this way, the dependability attributes for component-based software development are identified. The six dependability attributes are availability, reliability, confidentiality, integrity, safety, and maintainability. These dependability attributes will protect the CBSD product from security threats. Moreover, embedding dependability attributes will relieve end users of threats and vulnerabilities concerns.

In the evaluation method domain, Vulnerability Assessment Tools (VATs) are identified and used to assess dependability. VATs is a systematic evaluation of networks used to detect dependability breaches and determine appropriate dependability measures. The goal of the VATs approach is to provide efficient, thorough, and automated detection of known vulnerabilities in a specific operating system configuration. As an outcome of phase 1, gap analysis has been identified, and pilot studies on an awareness survey and VATs on 210 WASs have been conducted.

Phase two represents the MDDS-CBSD development that mitigates software component vulnerabilities. As shown in Figure 1, the MDDS-CBSD design process is motivated by the results of an awareness survey, VATs, and a gap analysis. The MDDS-CBSD design process considers five sub-activities namely, comparison, classification, categorisation, CBSD elements, and allocation. Phase two concludes with MDDS-CBSD development based on: (1) MDDS-CBSD architecture and (2) MDDS-CBSD embedding guideline.

Phase three examines the MDDS-CBSD. Interviews and surveys with industry experts are used to evaluate the MDDS-CBSD architecture. As shown in Figure 1, the architecture evaluations consider three activities: survey design, submission, and analysis. Our previous work [34] describes the survey methodology, details, and criteria used to select expert participants. A similar survey of industry experts was conducted to assess their awareness of dependability features embedded in the CBSD process. The survey to evaluate the MDDS-CBSD architecture uses a similar method. Twenty-five industry experts took part in the survey. Participants are experts from an in-house software development company.

Based on industrial practicality, an empirical study was conducted to evaluate (Verify and Validate V&V) the MDDS-CBSD using a real-world test bed system. This empirical study

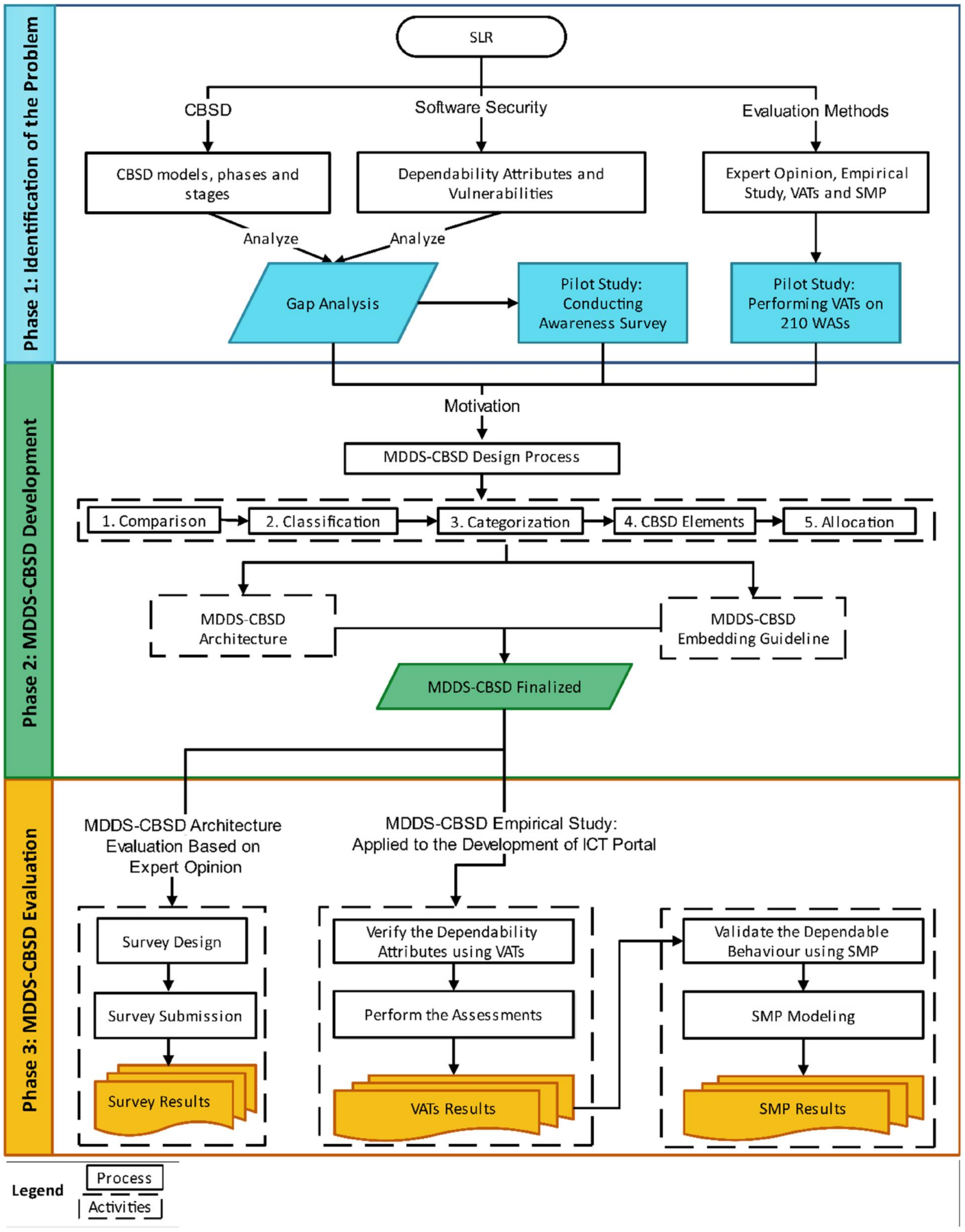
Phase 1: Identification of the Problem
SLR
CBSD
Software Security
Evaluation Methods
CBSD models, phases and stages
Dependability Attributes and Vulnerabilities
Expert Opinion, Empirical Study, VATs and SMP
Analyze
Analyze
Gap Analysis
Pilot Study: Conducting Awareness Survey
Pilot Study: Performing VATs on 210 WASs
Phase 2: MDDS-CBSD Development
Motivation
MDDS-CBSD Design Process
1. Comparison
2. Classification
3. Categorization
4. CBSD Elements
5. Allocation
MDDS-CBSD Architecture
MDDS-CBSD Embedding Guideline
MDDS-CBSD Finalized
Phase 3: MDDS-CBSD Evaluation
MDDS-CBSD Architecture Evaluation Based on Expert Opinion
MDDS-CBSD Empirical Study: Applied to the Development of ICT Portal
Survey Design
Survey Submission
Survey Results
Verify the Dependability Attributes using VATs
Perform the Assessments
VATs Results
Validate the Dependable Behaviour using SMP
SMP Modeling
SMP Results
Legend
Process
Activities

FIGURE 1 MDDS-CBSD methodology. Abbreviations: MDDS-CBSD, Model For Developing Dependable System Using Component-Based Software Development; SLR, Systematic Literature Review Method; SMP, Semi-Markov Process; VATs, Vulnerability Assessment Tools; WAS, Web Application System

applies the MDDS-CBSD to ICT portal development. VAT configuration, VAT execution, and VAT analysis are considered for model evaluation. We did a similar vulnerability assessment in Ref. [35]. Vulnerability assessments of developed web application systems are also performed using similar methodologies. The assessments are designed to evaluate the developed web application systems' dependability and their ability to mitigate vulnerabilities. SMP was also used to test the developed ICT portal's reliability.

5 | MDDS-CBSD DESIGNING PROCESS

MDDS-CBSD development consists of three main activities as shown in Figure 1. The activities are: (1) MDDS-CBSD designing process; (2) MDDS-CBSD architecture phases and process; and (3) MDDS-CBSD embedding guideline. The first step taken in deriving the model was to determine the elements and processes involved. Five sub-activities were considered as follows:

1. Comparison: A comparative study on existing CBSD process models was conducted by identifying the elements and processes of each model and investigating their application processes.
2. Classification: The processes in existing models were classified using the compartmentalisation method used for the CBSD phases. Three fundamental CBSD phases were used for this procedure, namely, system requirement and qualification, component development, and system development.
3. Categorisation: These processes were further categorised according to their descriptions. Despite having different names, some of these processes describe the same activities. Therefore, these differences and similarities must be resolved before processes can be identified. Processes with similar sets of activities were noted, and different process names were substituted with names that reflect the set of activities.
4. CBSD Elements: CBSD elements have been defined based on the analysis of the classification and categorisation stages:
    a) Stages that comprise the CBSD processes and reusability features preservation;
    b) Compartmentalisation method of the architecture phases and processes;
    c) Iterative and incremental integration process methods.

Therefore, the MDDS-CBSD architecture has been finalised.

5 Allocation: Using the finalised MDDS-CBSD architecture, the dependability attributes were embedded into the proposed model process in the following stages:
  a) Dependability requirement: emphasis on dependability attributes in the requirement analysis and component selection.
  b) Dependability design, architecture, implementation, and testing: emphasis on dependability attributes in the mentioned three stages.
  c) Dependability component implementation and testing: emphasis on dependability attributes in the mentioned two stages.
  d) Dependability system testing: emphasis on dependability attributes at the system testing stage.

5.1 | MDDS-CBSD architecture

The first step taken in developing the MDDS-CBSD architecture was determining the phases and processes that needed to be included in the model. Figure 2 presents the MDDS-CBSD architecture.

Each process in the existing models was grouped into architecture phase compartmentalisation, iterative and incremental integration process, and preserving the reusability as shown by the legend box in Figure 2.

5.1.1 | Architecture phase compartmentalisation

To successfully apply CBSD, the architecture phases must be compartmentalised in the model. Compartmentalisation allows multiple activities to run concurrently without one activity blocking the other. Thus, architecture phase compartmentalisation saves time and resources. The architecture phase is divided into three parts: the system requirement and qualification phase, component development phase; and the system development phase.

1. *System Requirements and Qualifications Phase:* A set of software components is identified, built, catalogued, and distributed for use in existing and future software systems. To analyse the system domain, system requirements and qualifications are used to identify common areas and methods of describing the system. If system reusability is considered, this phase should be performed early in the software specification. This phase allows software engineers to reuse software components on new and old systems.
2. *Component Development Phase:* This phase is divided into three sub-phases. Following the system requirements and qualification phase, the component development team will decide which components can be reused (e.g. those already in the organisation repository or purchased) and which must be developed from scratch. Each sub-phase has its own life cycle. So, three teams will work on the sub-phases in parallel.
   2.1 *Development for Reuse:* The component interface is defined first in the component development process. This is a fixed mechanism among components. The component design and implementation can begin once the interface is defined and the methods' goals are defined.
   2.2 *Development without Modification:* A component can be reused or migrated into a specialised subclass of an existing component created by a programmer.
   2.3 *Development Post-Modification Development:* Component-based development avoids creating new modules from scratch. Adapting some existing components may require minor or major changes. But

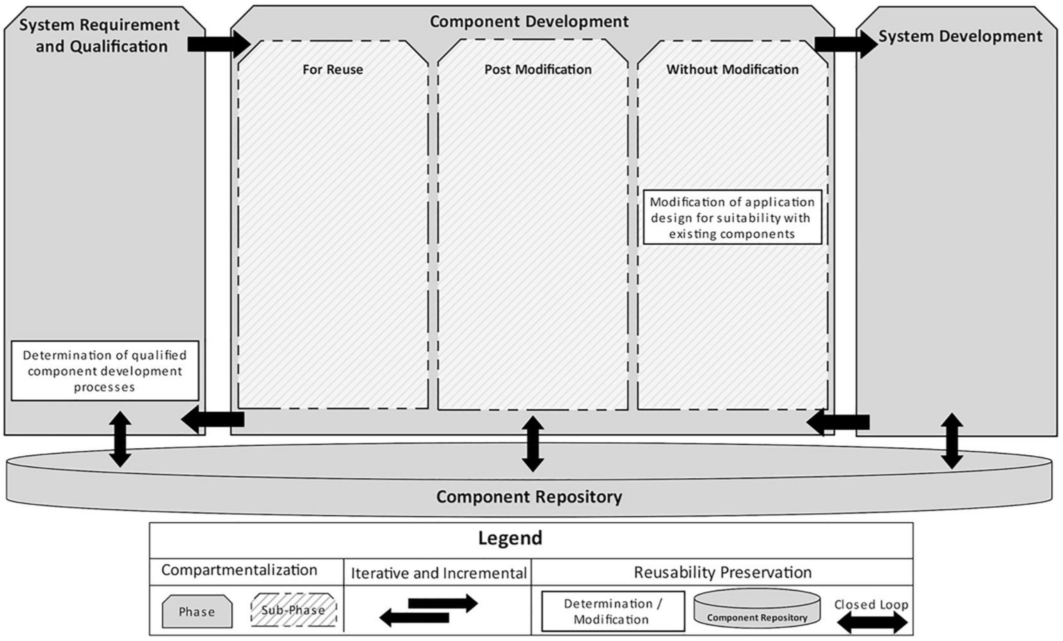
System Requirement and Qualification
Component Development
System Development
For Reuse
Post Modification
Without Modification
Modification of application design for suitability with existing components
Determination of qualified component development processes
Component Repository
Legend
Compartmentalization
Iterative and Incremental
Reusability Preservation
Phase
Sub-Phase
Determination / Modification
Component Repository
Closed Loop

FIGURE 2 Model for developing a dependable system using component-based software development architecture.

reusing an existing component necessitates some adaptations. For example, the component interface or some methods may need to be changed. Adaptation is the process of appending a thin layer of code to a component to make the required changes.

3. *System Development Phase:* It integrates the component development team's components into an architecture style and connects them to appropriate infrastructure for effective coordination and management.

5.1.2 | Iterative and incremental integration process

Uncertainty in requirements and implementation approaches often plagues software component development. Associating before resolving uncertainties complicates the development process. Parties will receive unclear specifications at the start of development. During the project, all parties must collaborate. Practices and processes necessitate long-distance collaboration.

The use of iterative and incremental development (IID) as a process model is required in software development. IID is an iterative system that adds new features incrementally. Because IID reacts quickly to changes, it is suitable for distributed development.

5.1.3 | Reusability preservation

The lack of specifications that allow programmers to anticipate component reuse is one of the issues that discourage it. This can be overcome by requiring developers to consider reusability early in the development process. MDDS-CBSD maintains reusability by:

1. *Selection of Suitable Component Development Processes:* The component development phase follows the system requirements and qualification phase. The team will execute the task if the component is available in-house or acquired from a third party. If the chosen component needs to be modified, another team will be assigned (post-modification development). Otherwise, the other team (development for reuse) will do it.
2. *Customisation of Application Design Based on Components:* In some cases, modifying the application design to fit the components is required. The goal is to reduce the cost of developing new components. When tailoring or modifying the components is costly or time-consuming, the application design is customised.
3. *Reusable Library (Repository)*: To apply CBSD successfully, the component repository deposition process must be shown. Reusable components should be chosen to improve component-based software productivity. The repository manages reusable parts. Using a repository for reusable components allows for classification, searching, modification, testing, implementation, version control, change control, and current and consistent documentation.
4. *Closed Loop*: In a closed-loop model, previous development components are explicitly fed back into the model to populate the repository. CBSD emphasises reusing previous development lifecycle components.
5. *Traceability*: To assist developers, each phase's stage will be demonstrated sequentially. Each step includes detailed explanations and examples to help developers apply this model. To avoid confusion, minor steps will be omitted.

5.2 | MDDS-CBSD embedding guideline

The dependability attributes are important to consider overcoming the lack of poor software development that causes dependability issues, as stated in Ref. [36]. So, we created a best practice guideline for embedding dependability attributes into the CBSD process [37]. This method ensures the employees understand their responsibilities in implementing dependability rules. The guideline was created with the help of Malaysian software developers and security consultants. This guideline contains best practices for incorporating dependability into the CBSD process. Dependability attributes are embedded into the requirements, design, implementation, and testing phases of the CBSD process. Figure 3 depicts the CBSD process' dependability attributes.

6 | MDDS-CBSD

Hence, the MDDS-CBSD is finalised. Figure 4 depicts the MDDS-CBSD. The figure depicts three phases: *System Requirement and Qualification, Component Development,* and *System Development.*

The *System Requirement and Qualification* phase is used to identify common areas and methods of describing the system. The *Dependability Attributes* in the requirement and component selection are highlighted as important stages in this phase.

Development for Reuse, Development Post Modification, and Development without Modification are three sub-phases of Component Development. Each of these three sub-phases has multiple stages. These three sub-phases include design, architecture, implementation, and testing based on Dependability Attributes.

Plotting out the *System Development* phase's stages, it integrates the component development team's components into an architecture style and connects them to appropriate infrastructure for effective coordination and management. The *Dependability Attributes Testing System* is integrated.

The arrows in Figure 4 connect the three main phases and the model repository, which presents iterative and incremental development (IID). Also shown is the component repository deposition process, which preserves reusability.

7 | EMPIRICAL STUDY

A practical study on the MDDS-CBSD is conducted. This empirical study applies the MDDS-CBSD to an ICT portal. Using the MDDS-CBSD for ICT portal development ensures proper integration of dependability attributes and generalisation of results [38–41].

The ICT portal development was done in collaboration with a Malaysian company. The company name was kept secret due to the competitive nature of the software development industry. So, we call it the Software Development Company (SDC). The ICT portal was created by a six-person software development team at SDC [42].

Therefore, an evaluation of the MDDS-CBSD is carried out in this paper to verify that the MDDS-CBSD is capable of mitigating the vulnerability in the developed model.

Thus, the MDDS-CBSD is evaluated in this work to ensure that it is capable of mitigating the vulnerability in the developed ICT portal.

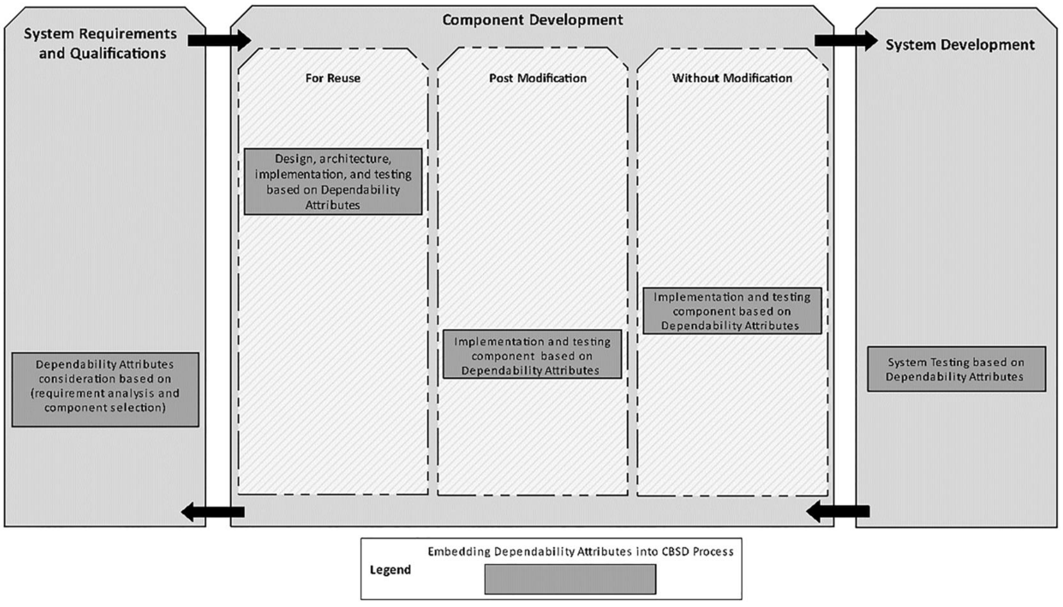
System Requirements and Qualifications
Component Development
System Development
For Reuse
Post Modification
Without Modification
Design, architecture, implementation, and testing based on Dependability Attributes
Implementation and testing component based on Dependability Attributes
Implementation and testing component based on Dependability Attributes
Dependability Attributes consideration based on (requirement analysis and component selection)
System Testing based on Dependability Attributes
Legend
Embedding Dependability Attributes into CBSD Process

FIGURE 3 Embedding dependability attributes into the component-based software development process.

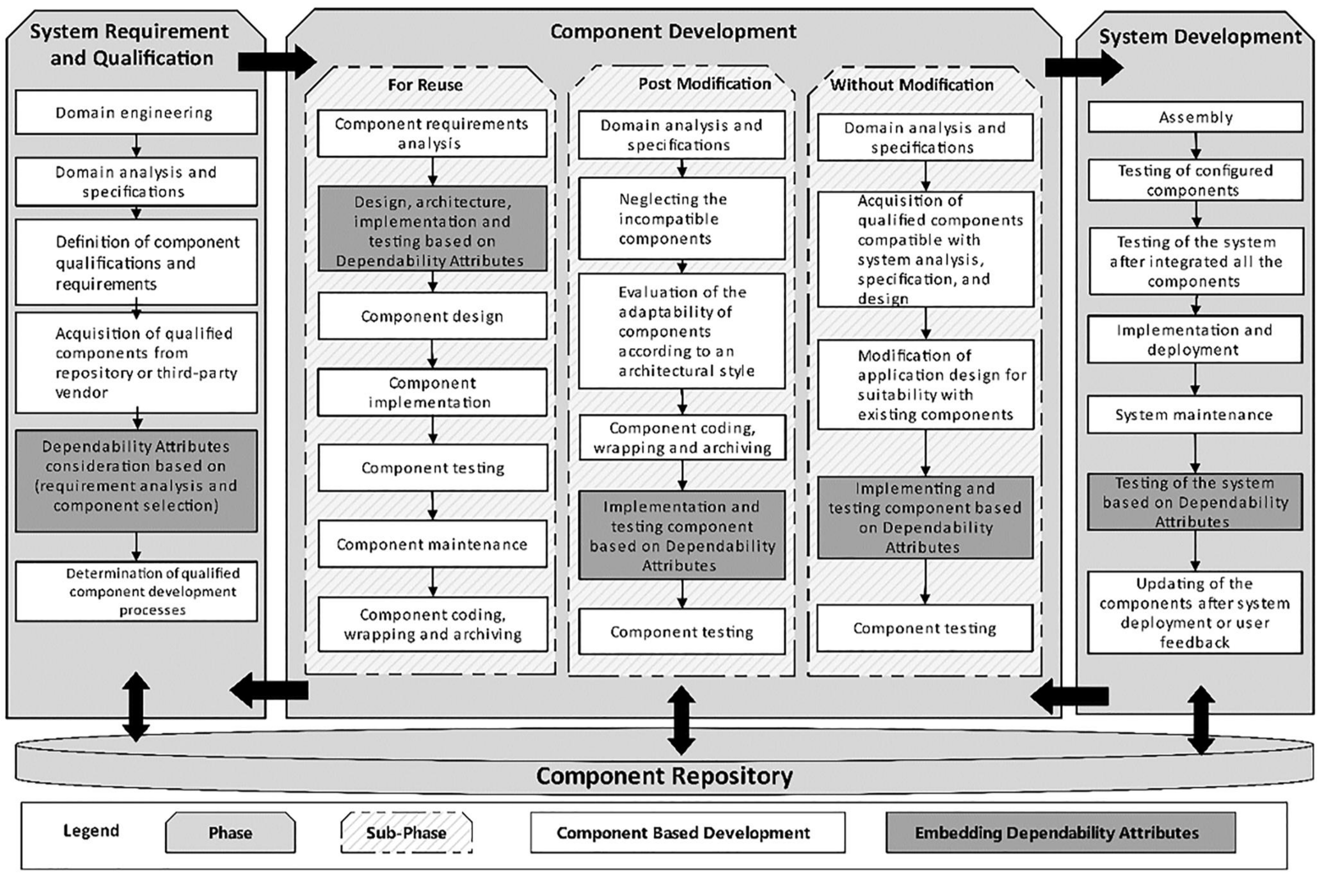
System Requirement and Qualification
Domain engineering
Domain analysis and specifications
Definition of component qualifications and requirements
Acquisition of qualified components from repository or third-party vendor
Dependability Attributes consideration based on (requirement analysis and component selection)
Determination of qualified component development processes
Component Development
For Reuse
Component requirements analysis
Design, architecture, implementation and testing based on Dependability Attributes
Component design
Component implementation
Component testing
Component maintenance
Component coding, wrapping and archiving
Post Modification
Domain analysis and specifications
Neglecting the incompatible components
Evaluation of the adaptability of components according to an architectural style
Component coding, wrapping and archiving
Implementation and testing component based on Dependability Attributes
Component testing
Without Modification
Domain analysis and specifications
Acquisition of qualified components compatible with system analysis, specification, and design
Modification of application design for suitability with existing components
Implementing and testing component based on Dependability Attributes
Component testing
System Development
Assembly
Testing of configured components
Testing of the system after integrated all the components
Implementation and deployment
System maintenance
Testing of the system based on Dependability Attributes
Updating of the components after system deployment or user feedback
Component Repository
Legend
Phase
Sub-Phase
Component Based Development
Embedding Dependability Attributes

FIGURE 4 Model for developing a dependable system using component-based software development.

The MDDS-CBSD was evaluated based on two elements as the following:

I. MDDS-CBSD architecture is evaluated by conducting a survey with 26 experts.
II. The dependability attributes embedded in the MDDS-CBSD process is evaluated by verifying the dependability attributes of the developed ICT portal.

7.1 | Expert evaluation

The results were based on the supplemental documents and expert explanations for MDDS-CBSD. Figures 5–14 shows the MDDS-CBSD architecture survey results. Figure 5 shows that 44% of experts strongly agree and 50% agree that the MDDS-CBSD processes are essential to CBSD. This result validates the model architecture by 94%.

Figure 6 shows that 42.3% of experts strongly agree that MDDS-CBSD solves the sequential structure issue, followed by 53.8% who agree and 3.8% who disagree. Figure 7 shows the MDDS-CBSD is simple to understand and apply, with 73.1% of experts who strongly agree. Moreover, 42.3% of experts strongly agreed that the MDDS-CBSD considers the use of reusable components. Also, 53.8% agreed. Only 3.8% of experts chose fair. Figure 8 reflects this.

Figure 9 shows how the MDDS-CBSD improves component development by breaking it down into three sub-phases to help developers save time and resources. 73.1% of experts strongly agree that the MDDS-CBSD has improved component development, with 27% agreeing. Figure 10 shows the advantages of architecture phase compartmentalisation. 25 experts agreed (strongly agree and agree) on the benefits of architecture phase compart-tnqh_9; mentalisation.

Figure 11 demonstrates the process model of iterative and incremental development (IID). 25 experts strongly agree and agree that IID is suitable for the development process. Figure 12 shows that 57.7% of experts strongly agree that the MDDS-CBSD provides traceability, followed by 38.5% who agree and 3.8% fair.

61.5% of experts strongly agree that component-based application design helps reduce development costs. Also, 23.1% of experts think this is true. Figure 13 shows the results. Figure 14 shows that 96.1% of experts strongly agree (strongly agree and agree) that including the closed-loop feature will promote reuse of previous development life-cycle components.

Based on the survey results, the MDDS-CBSD architecture supports rapid development, increased productivity, product quality, and reusability of CBSD, and thus, promise that the application of CBSD will be successful.

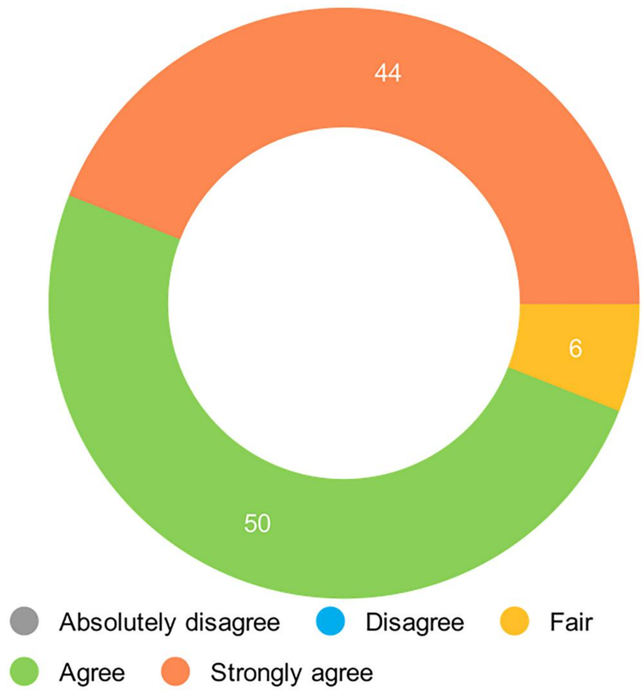

FIGURE 5 Model for developing a dependable system using component-based software development processes is essential for the component-based software development model.

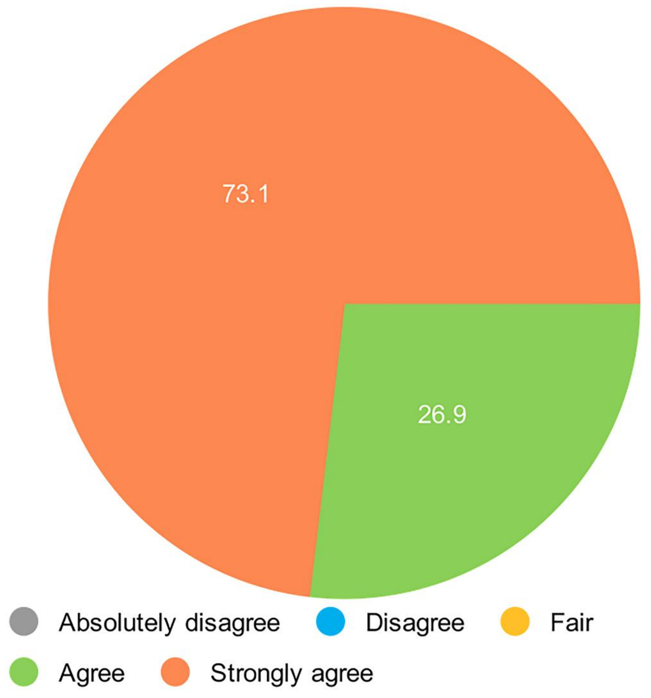

FIGURE 7 Model for developing a dependable system using component-based software development is easy and simple to be applied.

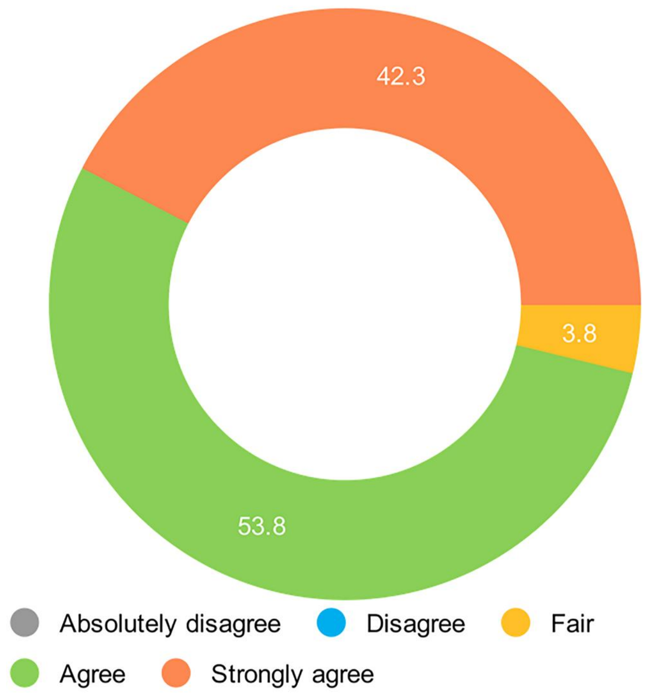

FIGURE 6 Model for developing a dependable system using component-based software development addressed the sequential structure problem.

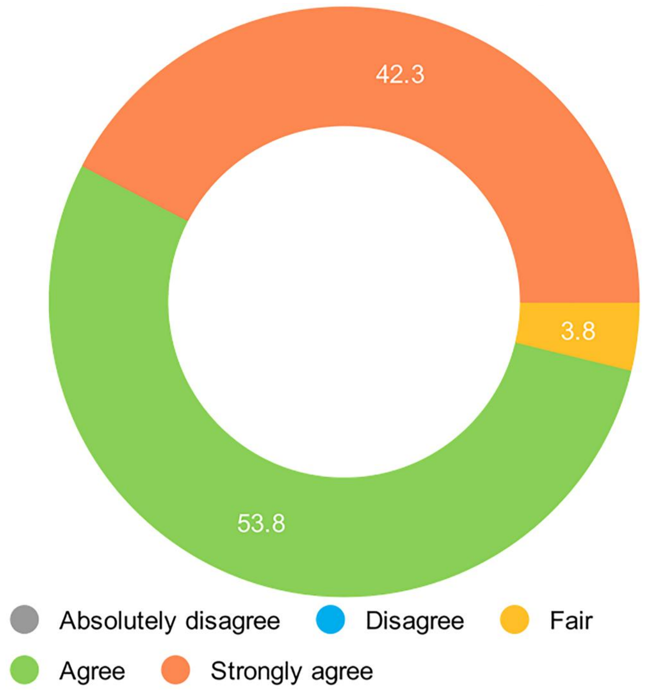

FIGURE 8 Model for developing a dependable system using component-based software development address the reusable components.

7.2 | Dependability attributes verification

Our previous work [42] discussed an empirical study of the proposed MDDS-CBSD. This study will apply the proposed MDDS-CBSD to develop an ICT portal. The developed ICT portal was evaluated using Vulnerability Assessment Tools (VATs) to verify its dependability.

The ICT portal was developed in two versions to assess the MDDS-CBSD's dependability. The first version was built using the traditional CBSD model (no dependability attributes), hence the name 'traditional deployment'. The second ICT portal was built with an MDDS-CBSD with embedded dependability, henceforth referred to as 'MDDS-CBSD deployment'. The ICT portal was evaluated on two key dimensions:

1. The MDDS-CBSD deployment should be less prone to failure.
2. MDDS-CBSD deployment should be more dependable than traditional deployment.

Experiments were run to assess the ICT portal on these two dimensions. The MDDS-CBSD results from both cases were compared against the traditional deployment. The results of comparing traditional and MDDS-CBSD deployment are briefly explained. The comparison was made to see how the two deployment methods affected the system's vulnerabilities. The results of JMeter, OpenVAS, and RATS scanning of the ICT portal with various deployments are shown in the following sections.

7.2.1 | Apache JMeter results

JMeter was used to test the ICT portal's availability and reliability. The results came after executing all the test plans. Figure 15 shows that before 100 threats, Bar1 and Bar2 throughput are close because the test bed was only loaded with a few threats. From 150 to 200 threats, Bar 1's average value is roughly 10% higher than Bar 2's. However, increasing the number of threats to over 200 per second causes a rapid decrease in Bar2, resulting in system failure.

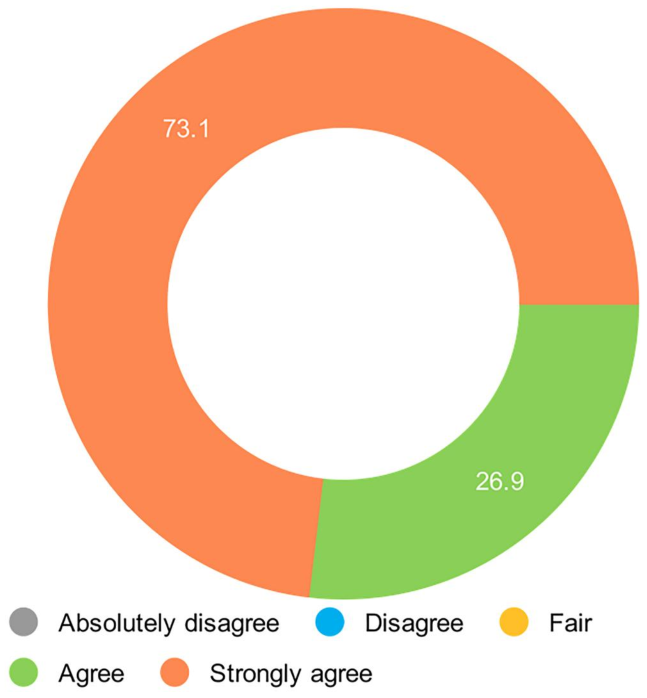

FIGURE 9 Model for developing a dependable system using component-based software development enhances the component development phase.

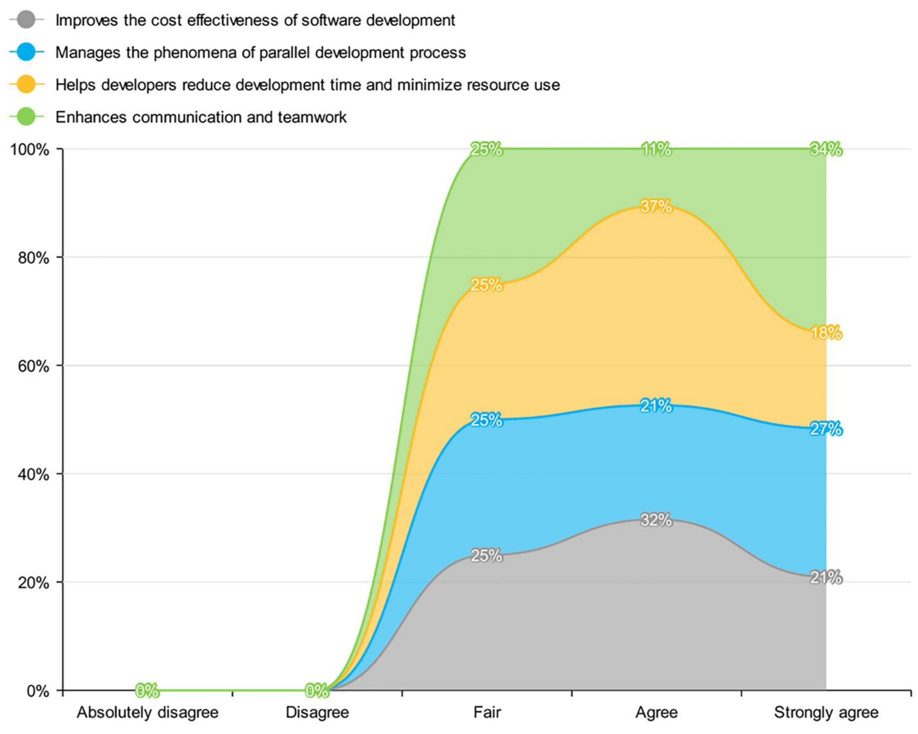

FIGURE 10 Architecture phase compartmentalisation advantages.

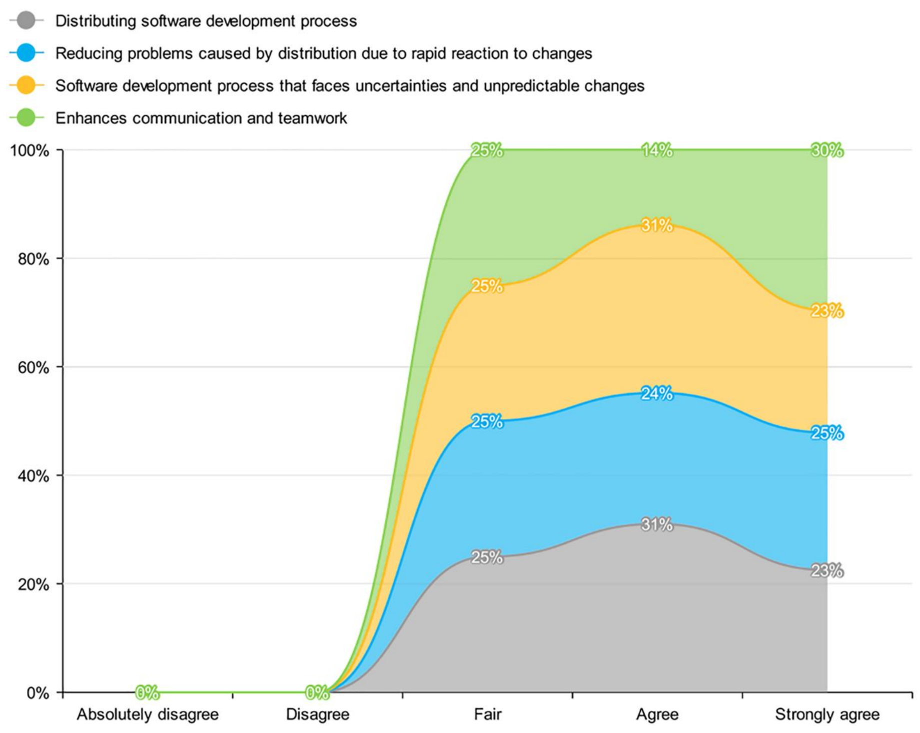

FIGURE 11 Iterative and incremental development advantages.

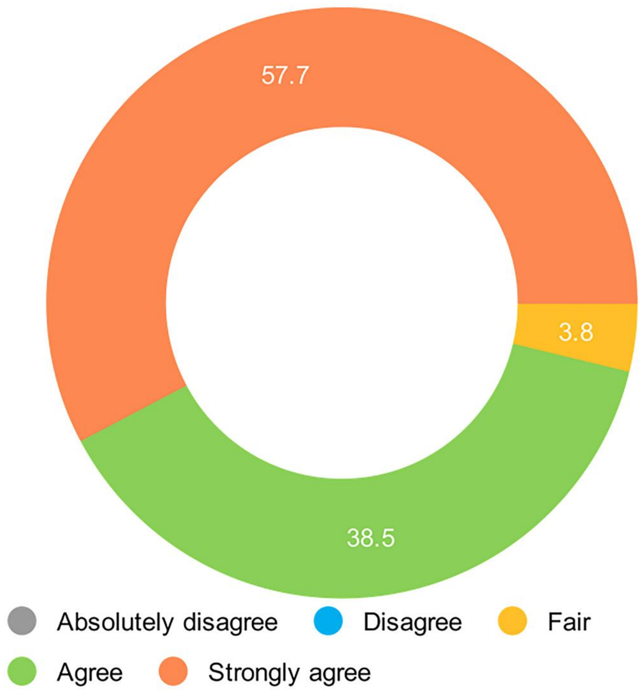

FIGURE 12 Model for developing a dependable system using component-based software development provides traceability.

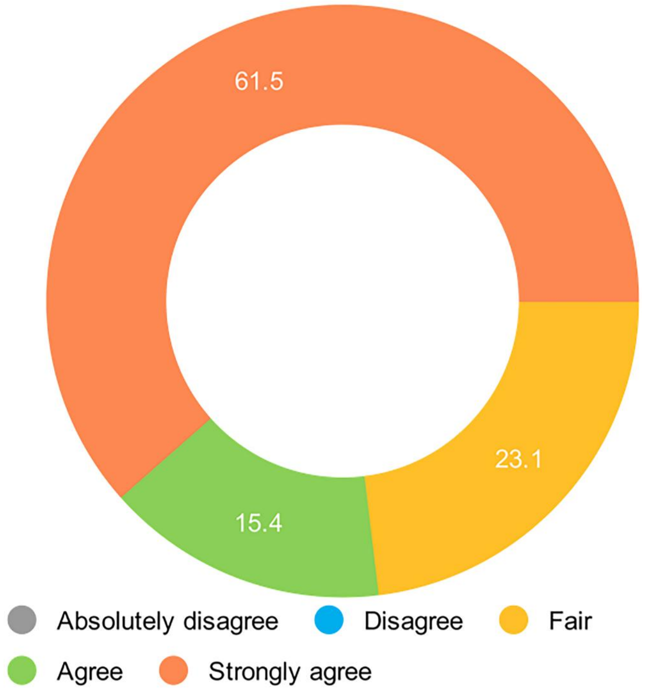

FIGURE 13 Cost reduction based on customisation of application design.

The services are kept active despite the decrease in Bar1. As a result, the service system with MDDS-CBSD deployment outperforms the traditional deployment, and the fluctuations in bar1 are less than those in Bar2. Bar1 now works better in service systems. For both MDDS-CBSD and traditional deployments, availability and reliability are summarised in Figure 15. The observed availability and reliability change linearly with the number of requests per second.

However, when the number of requests exceeds the threshold value of 175, the availability and reliability begin to decline. The decline in availability and reliability in the MDDS-CBSD has improved to 3.34%. The results of the assessments show that MDDS-CBSD deployment performs better than traditional deployment when the system's load is increased gradually. The graphs indicate that throughput for Bar1 is stable for 100 and 200 users but unstable for 250 users, due to the ICT portal designed to support 200 users concurrently.

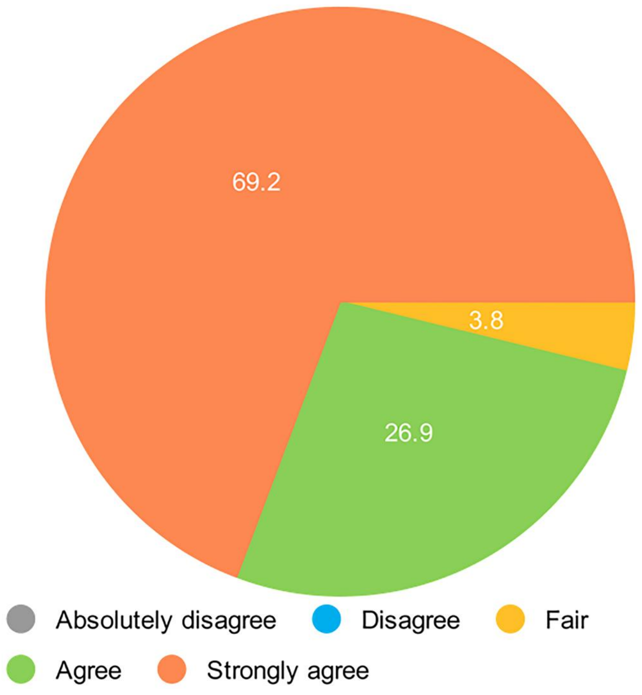

FIGURE 14 Including closed-loop will promote the reusability.

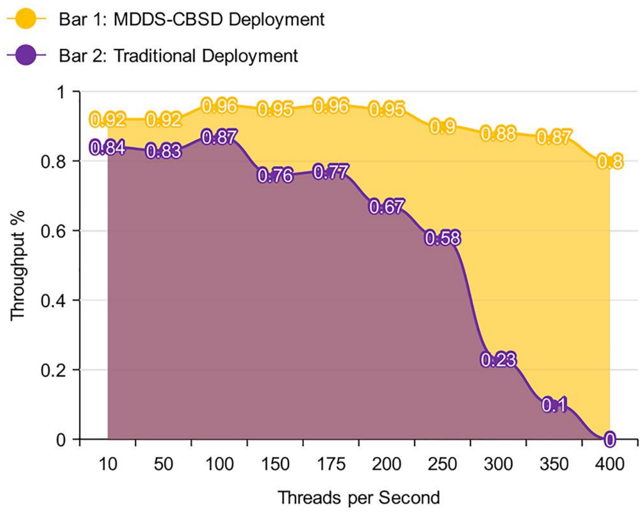

FIGURE 15 Availability and reliability comparison.

7.2.2 | OpenVAS results

The scan was performed using OpenVAS on both traditional and MDDS-CBSD. The same configuration was used for both scans. The first scan was traditional, and the second was MDDS-CBSD. The scans produced two reports. These reports detail the flaws found in both traditional and MDDS-CBSD. Figures 16–18 show each scan's filtered results. Vulnerabilities were addressed based on their type and dependability attributes. Figure 16 compares confidentiality vulnerabilities. According to Figure 16, OpenVAS detected 24 vulnerabilities for traditional deployment and 5 for MDDS-CBSD deployment. The medium risk factor is 12 for traditional deployment and 8 for MDDS-CBSD, and for the low risk factor, 18 for traditional deployment and 12 for MDDS-CBSD deployment.

Figure 17 compares integrity vulnerabilities. OpenVAS detected 29 vulnerabilities for traditional deployment and 8 for MDDS-CBSD deployment. The medium risk factor is 14 for

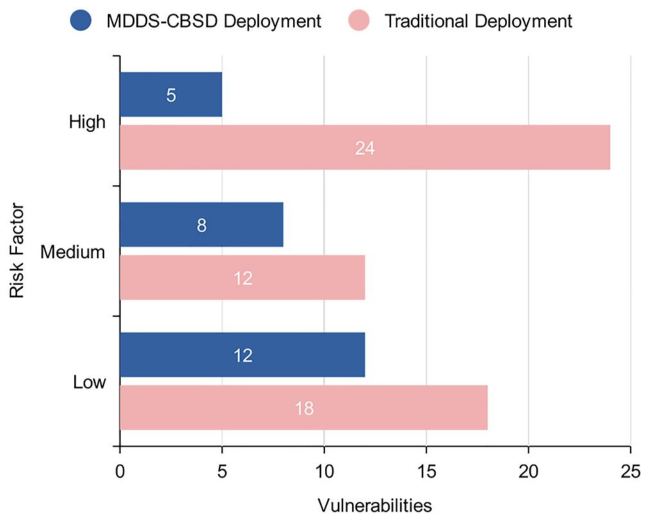

FIGURE 16 Confidentiality vulnerabilities comparison.

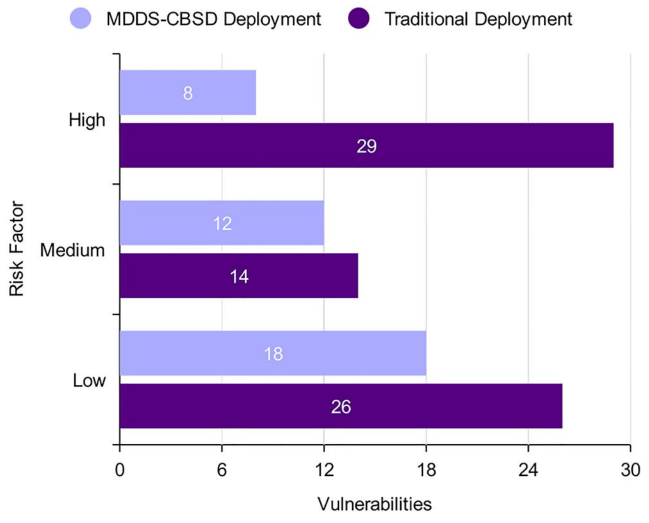

FIGURE 17 Integrity vulnerabilities comparison.

traditional deployment and 12 for MDDS-CBSD, and for the low risk factor, 26 for traditional deployment and 18 for MDDS-CBSD deployment.

Figure 18 compares safety flaws. OpenVAS detected 34 vulnerabilities for traditional deployment and 7 for MDDS-CBSD deployment. The medium risk factor is 26 for traditional deployment and 18 for MDDS-CBSD, and for the low risk factor, 22 for traditional deployment and 14 for MDDS-CBSD deployment.

7.2.3 | RATS results

The ICT portal's maintainability was assessed using RATS. Figure 19 compares the maintainability of traditional and MDDS-CBSD deployments. The results show that 75% of traditional deployment source code issues were detected, while only 25% of MDDS-CBSD deployment source code issues were detected. The MDDS-CBSD clearly improves system maintainability. Figure 19 compares maintainability vulnerabilities. RATS detected 36 vulnerabilities for traditional deployment and 8 for MDDS-CBSD deployment. The medium risk factor is 14 for traditional deployment and 12 for MDDS-CBSD, and for the low risk factor, 26 for traditional deployment and 18 for MDDS-CBSD deployment.

7.2.4 | Summary of the verification process

The MDDS-CBSD results were compared to traditional deployment. The ICT portal's dependability attributes show that MDDS-CBSD deployment prevents more failures than traditional deployment. Moreover, MDDS-CBSD deployment mitigates vulnerabilities better than traditional deployment. The availability of the MDDS-CBSD has also been improved to 3.34%. As the system load increases, MDDS-CBSD deployment outperforms traditional deployment. Also, system vulnerabilities will be quickly tolerated, and risks will be managed.

8 | DISCUSSION

This work compares MDDS-CBSD to other models to show the differences in phases, stages, and features. The MDDS-CBSD has the following key features that most other models lack:

1. **Embedding Dependability Attributes:** MDDS-CBSD shows how the six dependability attributes are embedded into the component-based software development process. This model helps software developers, designers, and engineers build dependable systems. The model also enables managers and developers to track dependability attributes, requirements, design, implementation, and testing throughout the CBSD process.
2. **Guidelines for Composing Dependability Attributes in CBSD Phases:** MDDS-CBSD outlines a best practice guideline. The guideline was created with the help of Malaysian software developers and security consultants. This guideline contains the best practices for incorporating dependability into the CBSD process. The guideline includes processes for eliciting and defining dependability attribute requirements using risk analysis and assessment.
3. **Empirical Study:** An empirical study on the MDDS-CBSD has been conducted. This empirical study applies the MDDS-CBSD to an ICT portal development to ensure a proper integration of dependability attributes and generalisation of results.
4. **Evaluations:** MDDS-CBSD was evaluated to see if it could mitigate the vulnerability in the developed model. The MDDS-CBSD was assessed on two levels: (a) Expert

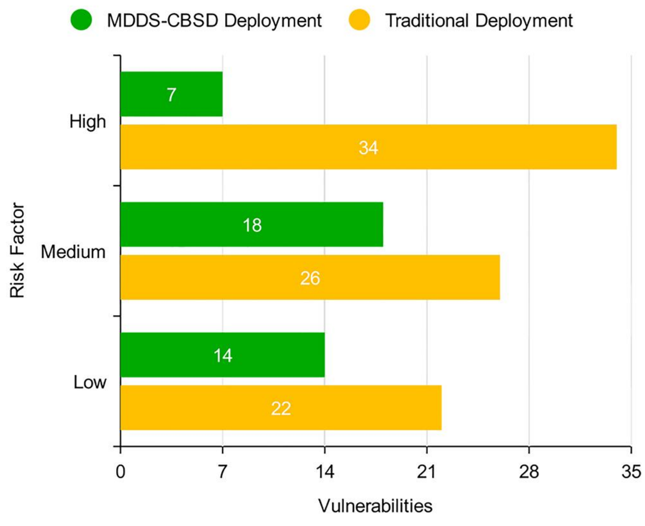

FIGURE 18 Safety vulnerabilities comparison.

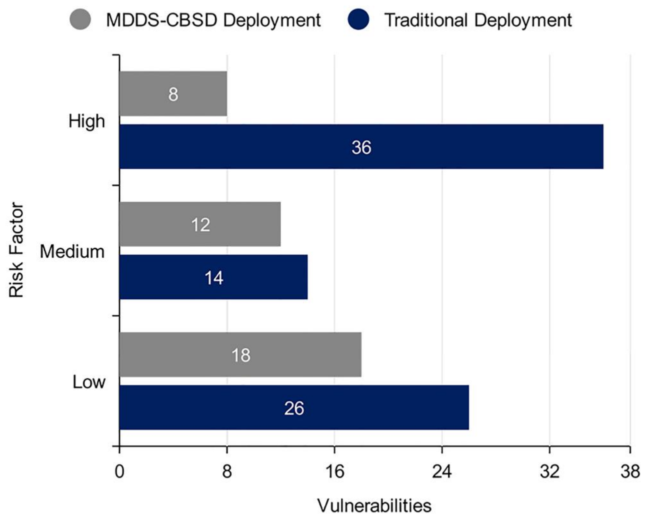

FIGURE 19 Maintainability vulnerabilities comparison.

TABLE 2 Mapping between existing CBSD models and key features

	Key features							
CBSD models	**Embedding dependability attributes**	**Guidelines for composing dependability attributes in CBSD phases**	**Empirical study**	**Expert evaluation**	**Dependability/ Security evaluation**	**Compartmentalisation**	**Reusability**	**Iterative and incremental development (IID)**
[41]							√	
[42]							√	
[43]							√	√
[44]							√	√
[45]							√	√
[46]							√	√
[47]							√	√
[48]							√	√
[49]							√	√
[50]							√	√
[51]							√	√
[50]							√	√
[52]						√	√	√
[53]			√	√		√	√	√
[54]						√	√	√
[55]						√	√	√
[56]						√	√	√
[1]						√	√	√
[57]						√	√	√
[58]						√	√	√
[59]						√	√	√
[60]						√	√	√
[61]						√	√	√
[12]						√	√	√
[62]			√			√	√	√
[63]			√	√		√		
[64]						√	√	√
[65]							√	√
[66]			√			√	√	√
[67]			√			√	√	√
MDDS-CBSD	√	√	√	√	√	√	√	√

evaluation of MDDS-CBSD architecture; (b) The dependability of the developed ICT portal was evaluated against the dependability of the MDDS-CBSD process.

5. **Compartmentalisation**: This problem is addressed in MDDS-CBSD by compartmentalising the design phases. Compartmentalisation allows multiple activities to run concurrently without one activity blocking the other. Thus, architecture phase compartmentalisation saves time and resources.
6. **Reusability**: MDDS-CBSD overcomes the issues that discourage component reuse by forcing developers to consider reusability early in the development process.

7. **Iterative and Incremental Development (IID):** MDDS-CBSD addressed the IID method as a software development process. IID is an iterative system that adds new features incrementally. Because IID reacts quickly to changes, it is suitable for distributed development.

Table 2 summarises the existing CBSD model-key feature mapping. A check mark (√) indicates that a model supports the key features. There is no check mark for process models that do not support the key features.

9 | CONCLUSION

This study proposed a model for developing dependable systems using a component-based software development approach (MDDS-CBSD). The MDDS-CBSD methodology and design process were detailed. The MDDS-CBSD was evaluated in two stages. First, a survey and expert interviews evaluated the MDDS-CBSD frame. Second, the developed system's dependability was tested using VATs. According to the MDDS-CBSD architecture survey, experts agree on the following: (a) All MDDS-CBSD processes are required for CBSD; (b) architecture phase compartmentalisation is beneficial; (c) IID is suitable for distributed development; and (d) MDDS-CBSD specifies a place for considering reusable components. Thus, the MDDS-CBSD architecture succeeds. Aside from that, the MDDS-CBSD can mitigate vulnerabilities better than the traditional development model. The availability of the MDDS-CBSD has improved to 3.34%. As the system load increases, MDDS-CBSD deployment outperforms traditional deployment. Our future work will validate the MDDS-dependable CBSD's behaviour using the Semi-Markov Process (SMP), and in addition, apply MDDS-CBSD to more than one software development company.

AUTHOR CONTRIBUTIONS

Hasan Kahtan: Conceptualisation; Data curation; Formal analysis; Funding acquisition; Investigation; Methodology; Project administration; Resources; Software; Supervision; Validation; Visualisation; Writing – original draft; Writing – review & editing. **Mansoor Abdulhak:** Conceptualisation; Methodology; Software; Validation; Visualisation; Writing – original draft. **Ahmad Salah Al-Ahmad:** Methodology; Resources; Visualisation; Writing – review & editing. **Yehia Ibrahim Alzoubi:** Methodology; Visualisation; Writing – review & editing.

ACKNOWLEDGEMENT

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

CONFLICT OF INTEREST

The authors have no conflicts of interest to declare.

DATA AVAILABILITY STATEMENT

Data available in the article.

ORCID

Hasan Kahtan https://orcid.org/0000-0001-6521-7081

REFERENCES

1. Gill, N., Tomar, P.: Modified development process of component-based software engineering. Software Eng. Notes 35(2), 1–6 (2010). https://doi.org/10.1145/1734103.1734120
2. Bentrad, S., Kahtan Khalaf, H., Meslati, D.: Towards a hybrid approach to build aspect-oriented programs. IAENG Int. J. Comput. Sci. 47(4) (2020). https://www.iaeng.org/IJCS/issues_v47/issue_4/IJCS_47_4_08.pdf
3. Wang, Y., et al.: OSAI: a component-based open software architecture for modern industrial control systems. Arabian J. Sci. Eng. 47(3), 3805–3819 (2022). https://doi.org/10.1007/s13369-021-06123-3
4. Dilhara, W.: Web Component Based Ecommerce Application Development Framework for Hosted Software Solutions (Pvt) Ltd. In: Master of Computer Science. School of Computing, University of Colombo, Sri Lanka (2021)
5. Iliev, O., Yoshinov, R.: Component-based software architecture applied for design of heritage content. In: Eleventh International Conference Digital Presentation and Preservation of Cultural and Scientific Heritage DiPP2021. Burgas, Bulgaria (2021)
6. Liu, L., Yao, Y., Li, J.: Development of a novel component-based open CNC software system. Int. J. Adv. Manuf. Technol. 108(11), 3547–3562 (2020). https://doi.org/10.1007/s00170-020-05590-6
7. Salehudin, N.B., et al.: A proposed course recommender model based on collaborative filtering for course registration. Int. J. Adv. Comput. Sci. Appl. 10(11), 162–168 (2019). https://doi.org/10.14569/ijacsa.2019.0101122
8. AlAhmad, A.S., et al.: Mobile cloud computing models security issues: a systematic review. J. Netw. Comput. Appl. 190(103152), 1–17 (2021). https://doi.org/10.1016/j.jnca.2021.103152
9. Al-bashiri, H., et al.: Memory-based collaborative filtering: impacting of common items on the quality of recommendation. Int. J. Adv. Comput. Sci. Appl. 10(12), 132–137 (2019). https://doi.org/10.14569/ijacsa.2019.0101218
10. Mohanty, S., Acharya, A.A., Mohapatra, D.P.: A model based prioritization technique for component based software retesting using uml state chart diagram. In: 2011 3rd International Conference on Electronics Computer Technology, pp. 364–368. IEEE, Kanyakumari, India (2011). https://doi.org/10.1109/ICECTECH.2011.5941719
11. Moradian, E., Håkansson, A.: Controlling security of software development with multi-agent system. In: Knowledge-Based and Intelligent Information and Engineering Systems, pp. 98–107. (2010)
12. Sommerville, I.: Software Engineering, Ninth Edition. Pearson-Addison Wesley, Boston (2011)
13. Karen, G.: Software Security Assurance: A State-Of-The-Art Report (SOAR). DTIC Document (2007)
14. Al-Ahmad, A.S., et al.: Systematic literature review on penetration testing for mobile cloud computing applications. IEEE Access 7, 173524–173540 (2019). https://doi.org/10.1109/ACCESS.2019.2956770
15. Al-Ahmad, A.S., Kahtan, H.: Test case selection for penetration testing in mobile cloud computing applications: a proposed technique. J. Theor. Appl. Inf. Technol. 96(13) (2018). http://www.jatit.org/volumes/Vol96No13/23Vol96No13.pdf
16. Karen, G., et al.: Security in the Software Life Cycle: Making Software Development Processes–And the Software Produced by Them–More Secure, Draft 1.1. Department of Homeland Security, p. 46 (2006)
17. Al-Ahmad, A.S., Kahtan, H., Fuzz test case generation for penetration testing in mobile cloud computing applications, Vasant P. Zelinka I. Weber G.W. Intelligent Computing & Optimization. ICO 2018. Advances in Intelligent Systems and Computing, vol. 866. Springer, Cham (2018) https://doi.org/10.1007/978-3-030-00979-3_27
18. Al-Ahmad, A.S., Kahtan, H.: Cloud computing review: features and issues. In: 2018 International Conference on Smart Computing and Electronic Enterprise (ICSCEE), pp. 1–5. IEEE, Kuala Lumpur (2018). https://doi.org/10.1109/ICSCEE.2018.8538387

19. Hasan, K., et al.: Motion analysis-based application for enhancing physical education. Adv. Sci. Lett. 24(10), 7668–7674 (2017). https://doi.org/10.1166/asl.2018.12997
20. Redwine, S., Jr.: Software Assurance: A Curriculum Guide to the Common Body of Knowledge to Produce, Acquire, and Sustain Secure Software, vol. V11. Department of Homeland Security, WA (2007)
21. Khan, R.A., et al.: Systematic literature review on security risks and its practices in secure software development. IEEE Access 10 (2022). https://doi.org/10.1109/ACCESS.2022.3140181
22. Khan, R.A., et al.: Security assurance model of software development for global software development vendors. IEEE Access 10(2022). https://doi.org/10.1109/ACCESS.2022.3178301
23. Khan, R.A., et al.: Systematic mapping study on security approaches in secure software engineering. IEEE Access 9, 19139–19160 (2021). https://doi.org/10.1109/access.2021.3052311
24. Khan, S.U., et al.: Critical success factors of component-based software outsourcing development from vendors' perspective: a systematic literature review. IEEE Access 10 (2021). https://doi.org/10.1109/ACCESS.2021.3138775
25. Rana, T.: Ex-man component model for component-based software construction. Arabian J. Sci. Eng. 45(4), 2915–2928 (2020). https://doi.org/10.1007/s13369-019-04213-x
26. Wareham, T., Sweers, M.: Exploring viable algorithmic options for automatically creating and reconfiguring component-based software systems: a computational complexity approach (full version). no. 2205.05001 [cs.SE], p. 38 (2022). https://doi.org/10.48550/ARXIV.2205.05001
27. Agarwal, J., Dubey, S.K., Tiwari, R.: Usability estimation of component-based software system using adaptive neuro fuzzy approach. Int. J. Adv. Intell. Paradigms 22(1-2), 99–113 (2022). https://doi.org/10.1504/ijaip.2022.123018
28. Park, J., et al.: Optimization of software component allocation for autonomous driving in cloud-vehicle edge environment. SSRN 4095613, 24 (2022). https://doi.org/10.2139/ssrn.4095613
29. Saari, M., Nurminen, M., Rantanen, P.: Survey of component-based software engineering within IoT development. In: 2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO), pp. 824–828. IEEE (2022)
30. Irina-Miruna, R.: Advantages and challenges of using component–based software development in the vision of building a modern educational system, [Online]. In: The 19th International Conference on Informatics in Economy (IE 2019) (2019). https://doi.org/10.12948/ie2019.04.10. https://www.conferenceie.ase.ro/
31. Kahtan, H., et al.: Embedding dependability attributes into component-based software development. Comput. Fraud Secur. 11, 8–16 (2014). https://doi.org/10.1016/s1361-3723(14)70548-2
32. Kahtan, H., Abu Bakar, N., Nordin, R.: Reviewing the challenges of security features in component based software development models. In: 2012 IEEE Symposium on E-Learning, E-Management and E-Services, pp.1–6 (2012). https://doi.org/10.1109/IS3e.2012.6414955
33. Jha, S.K., Mishra, R.: Predicting and accessing security features into component-based software development: a critical survey. In: Software Engineering, pp. 287–294. Springer (2019)
34. Alzoubi, Y.I., Alahmad, A., Kahtan, H.: Blockchain technology as a Fog computing security and privacy solution: an overview. Comput. Commun. 182, 129–152 (2021). https://doi.org/10.1016/j.comcom.2021.11.005
35. Kahtan, H., Abu Bakar, N., Nordin, R.: Embedding dependability attributes into component-based software development using the best practice method: a guideline. J. Appl. Secur. Res. 9(3), 348–371 (2014). https://doi.org/10.1080/19361610.2014.913230
36. Kahtan, H., et al.: Evaluation dependability attributes of web application using vulnerability assessments tools. Infor. Tech. J. 13(14), 2240–2249 (2014). https://doi.org/10.3923/itj.2014.2240.2249
37. Kahtan, H., Bakar, N.A., Nordin, R.: Dependability attributes for increased security in component-based software development. J. Comput. Sci. 10(8), 1298–1306 (2014). https://doi.org/10.3844/jcssp.2014.1298.1306
38. Kahtan, H., Bakar, N.A., Nordin, R.: Awareness of embedding security features into component-based software development model: a survey. J. Comput. Sci. 10(8), 1411–1417 (2014). https://doi.org/10.3844/jcssp.2014.1411.1417
39. Souza, L., Camboim, K., Alencar, F.: A systematic literature review about integrating dependability attributes, performability and sustainability in the implantation of cooling subsystems in data center. J. Supercomput. 78(14), 1–37 (2022). https://doi.org/10.1007/s11227-022-04515-2
40. kamal Kaur, R., Pandey, B., Singh, L.K.: Dependability analysis of safety critical systems: issues and challenges. Ann. Nucl. Energy 120, 127–154 (2018). https://doi.org/10.1016/j.anucene.2018.05.027
41. Brown, A.W., Wallnan, K.C.: Engineering of component-based systems. In: Proceedings of ICECCS '96: 2nd IEEE International Conference on Engineering of Complex Computer Systems (Held Jointly with 6th CSESAW and 4th IEEE RTAW), pp. 414. IEEE Computer Society, Montreal (1996). https://doi.org/10.1109/ICECCS.1996.558485
42. Aoyama, M.: Process and economic model of component-based software development: a study from Software CALS Next Generation Software Engineering program. In: Proceedings Fifth International Symposium on Assessment of Software Tools and Technologies, pp. 100–103. IEEE, Pittsburgh (1997). https://doi.org/10.1109/AST.1997.599919
43. Tran, V.: Component-based integrated systems development: a model for the emerging procurement-centric approach to software development. In: Proceedings. The Twenty-Second Annual International Computer Software and Applications Conference (Compsac '98) (Cat. No.98CB36241), pp. 128–135. IEEE, Vienna (1998). https://doi.org/10.1109/CMPSAC.1998.716648
44. Lee, S., et al.: COMO: a UML-based component development methodology. In: Proceedings Sixth Asia Pacific Software Engineering Conference (ASPEC'99) (Cat. No.PR00509). IEEE, Takamatsu, Japan (1999). https://doi.org/10.1109/APSEC.1999.809584.54
45. Yau, S., Dong, N.: Integration in component-based software development using design patterns. In: Proceedings 24th Annual International Computer Software and Applications Conference, pp. 369–374. COMPSAC2000 (2000). https://doi.org/10.1109/CMPSAC.2000.884750
46. Cheesman, J., Daniels, J., Szyperski, C.: UML Components: A Simple Process for Specifying Component-Based Software (No. 1). Addison-Wesley Reading, MA, USA (2001)
47. Allen, P.: Ebiz components. Objective View (6), 12–20 (2003)
48. Crnkovic, I.: Component-based software engineering—new challenges in software development. Software Focus 2(4), 127–133 (2003). https://doi.org/10.1002/swf.45
49. Hutchinson, J., et al.: A service model for component-based development. In: Proceedings. 30th Euromicro Conference, pp. 162–169. IEEE, Rennes (2004). https://doi.org/10.1109/EURMIC.2004.1333368
50. Capretz, L.: Y: a new component-based software life cycle model. J. Comput. Sci. 1(1), 76–82 (2005). https://doi.org/10.3844/jcssp.2005.76.82. https://ir.lib.uwo.ca/electricalpub/136/
51. Mei, H.: ABC: supporting software architectures in the whole lifecycle. In: Proceedings of the Second International Conference on Software Engineering and Formal Methods, 2004 SEFM 2004. IEEE, Beijing (2004). https://doi.org/10.1109/SEFM.2004.1347538
52. Crnkovic, I., Chaudron, M., Larsson, S.: Component-based development process and component lifecycle. J. Comput. Inf. Technol. 13(4), 44 (2006). https://doi.org/10.2498/cit.2005.04.10
53. Aris, H., Salim, S.: The development of a simplified process model for CBSD. Int. Arab J. Inf. Technol. 4(2), 89–96 (2007)
54. Qureshi, M., Hussain, S.: A reusable software component-based development process model. Adv. Eng. Software 39(2), 88–94 (2008). https://doi.org/10.1016/j.advengsoft.2007.01.021
55. Kouroshfar, E., Yaghoubi Shahir, H., Ramsin, R.: Process patterns for component-based software development. Model Data Eng. 54–68 (2009). https://doi.org/10.1007/978-3-642-02414-6_4
56. Sharp, J., Ryan, S.: A theoretical framework of component-based software development phases. ACM SIGMIS - Data Base 41(1), 56–75 (2010). https://doi.org/10.1145/1719051.1719055

57. Bose, D.: Component based development. Arxiv no. 011.2163 [cs.SE] (2010). https://doi.org/10.48550/arXiv.1011.2163
58. Chhillar, R.S., Kajla, P.: A new-knot model for component based software development. Int. J. Comput. Sci. 8 (2011)
59. Lau, K.K., Taweel, F.M., Tran, C.M.: The W model for component-based software development. In: 2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications, pp. 47–50. IEEE, Oulu, Finland (2011). https://doi.org/10.1109/SEAA.2011.17
60. Pandeya, S.S., Tripathi, A.K.: Testing component-based software: what it has to do with design and component selection. J. Software Eng. Appl. 4(1), 37–47 (2011). https://doi.org/10.4236/jsea.2011.41005
61. Shang, M., Wang, H., Jiang, L.: The development process of component-based application software. In: 2011 International Conference of Information Technology, Computer Engineering and Management Sciences, pp. 11–14. IEEE, Nanjing, China (2011). https://doi.org/10.1109/ICM.2011.227
62. Barnawi, A., et al.: Novel component based development model for sip-based mobile application. Arxiv no. 1202.2516 [cs.SE] (2012). https://doi.org/10.5121/ijsea.2012.3107
63. IrshadKhan, A., et al.: Validation of component based software development model using formal B-method. Int. J. Comput. Appl. 67(9), 24–35 (2013). https://doi.org/10.5120/11423-6768
64. Biondi, A., Buttazzo, G., Bertogna, M.: A design flow for supporting component-based software development in multiprocessor real-time systems. R. Time Syst. 54(4), 800–829 (2018). https://doi.org/10.1007/s11241-018-9301-3
65. Jha, S.K., Mishra, R.: A review on reusability of component based software development. Reliability: Theory & Applications 14(4), 32–36 (2019)
66. Derakhshanmanesh, M., et al.: Model-integrating development of software systems: a flexible component-based approach. Software Syst. Model 18(4), 2557–2586 (2019). https://doi.org/10.1007/s10270-018-0682-5
67. Umran Alrubaee, A., et al.: A process model for component-based model-driven software development. Information 11(6), 302 (2020). https://doi.org/10.3390/info11060302

How to cite this article: Kahtan, H., et al.: A model for developing dependable systems using a component-based software development approach (MDDS-CBSD). IET Soft. 1–17 (2022). https://doi.org/10.1049/sfw2.12085