SN Computer Science (2024) 5:467
https://doi.org/10.1007/s42979-024-02797-2

REVIEW ARTICLE

Development of a Software Design Error Taxonomy: A Systematic Literature Review

Tushar Agrawal[1] · **Gursimran Singh Walia**[2] · **Vaibhav K. Anu**[3]

Received: 19 April 2023 / Accepted: 14 March 2024

Abstract

The Software design process involves a high degree of human interaction and design artifacts creation by software engineers. Software designers use their experience to make assumptions and decisions. Therefore, the software design is prone to errors that lead to a bad quality software. Software engineering literature focuses on software errors and failures. This research relates the concept of human and organizational errors to the errors committed by software engineers during the software design phase to improve software quality. The authors performed a systematic literature review (SLR) of the software engineering to identify software design errors and later classify them. The outcome is a software design errors taxonomy that provides errors and a way to classify those errors based on their similarities and origin. This research proposes a guide to comprehend common software design errors. The guide will allow software engineers to focus and create robust software design. The guide also provides a structured approach to detect software design errors upfront during the software design inspection process. Software development organizations can use this awareness to develop countermeasures to help with error prevention, thereby improving the overall software quality.

Keywords Systematic literature review · Software design errors · Human errors · Design errors taxonomy · Software quality improvement

Introduction

With technology advancements, the software systems are becoming integral to all industry segments today. The software systems have become increasingly large and complex in design with the growing business necessities. Software design is a complex process and requires attention from software designers. Software designers must adhere to software design practices to create the best possible software design for solving business problems.

"Human fallibility, like gravity, weather and terrain, is just another foreseeable hazard…The issue is not why an error occurred, but how it failed to be corrected. We cannot change the human condition, but we can change the conditions under which people work."—James Reason, Managing the Risks of Organizational Accidents [1]

The software design process involves understanding all the information gathered from the users, applying experience and knowledge and using tools and techniques to come up with the blueprint/design of the system. During the software design phase, software designers interact with business stakeholders and transform the software requirements into design specifications to create a blueprint for the software system. Software designers make decisions and assumptions based on the requirements specifications throughout the process. They work with the stakeholders to draft an efficient software solution. Software engineers also inspect the design to validate the completeness and viability of the solution under design. Therefore, software design involves a bit of manual activity by software designer(s) and their

✉ Tushar Agrawal
tushar.agrawal@ndsu.edu

Gursimran Singh Walia
gwalia@augusta.edu

Vaibhav K. Anu
anuv@montclair.edu

[1] Department of Computer Science, North Dakota State University, Fargo, ND, USA

[2] School of Computer and Cyber Sciences, Augusta University, Augusta, GA, USA

[3] Department of Computer Science, Montclair State University, Montclair, NJ, USA

Published online: 20 April 2024

involvement with business stakeholder(s). Organization policies and software development processes also impact the software design.

As mentioned earlier, software design is a creative activity that involves problem-solving and execution through blueprinting. Focusing on a higher quality earlier in the software development process results in fewer faults and improved software quality in the end [2]. Software design is an essential subset of a software designer's work at the beginning of the software development cycle. Therefore, it is necessary to take due diligence while creating a software design and prevent errors.

In 2022, the Consortium for Information & Software Quality (CISQ) published a study highlighting that the cost of poor-quality software in the US in 2022 is approximately $2.41 Trillion annually [3]. This cost is significantly high and an indication of the poorly followed processes. The current software design reviews and inspection techniques are inadequate to prevent poor quality software. Software engineers spend a bit of time troubleshooting software failures and developing a fix. An erroneous design is typically the cause of many such software failures and faults. Therefore, there is an immense interest in reducing this cost because it is difficult to detect, diagnose and repair large software systems. To improve software quality, many researchers have created and published taxonomies [4, 5]. However, these studies categorize errors in requirements and development phases of the software lifecycle. This systematic literature review (SLR) focuses on the errors introduced during the software design phase of the Software Development Life Cycle (SDLC). More specifically, these errors result in software failures. This research also classifies the software design errors based on their similarities and origin beyond human errors like organizational influences.

The high-level goal of this research is to propose a framework (elaborated in Sect. "Review Results") using a classification-based approach (Sect. "RQ2 How can the Design Error Information Gathered in Response to the Question Above be Organized?", Table 3) for describing chains of breakdowns that lead software engineers (designers) to commit errors while creating software design. This research provides visibility into the underlying software design errors (root causes). The research also provides a methodology to analyze a software design objectively. The research further aids in the identification of the root causes of the software failures. It provides a structured approach to detect software design errors upfront and improve software quality.

The paper organization is as follows: In Sect. "Background", the paper defines the key terms like errors, faults and failures used throughout the paper and provides the history of efforts to improve software quality. Section "Research Method" elaborates the research method used to perform this systematic literature review. Section "Review Results" provides the detailed results from the literature review and creates a taxonomy of software design errors using a classification-based approach to address the research questions. Section "Discussion" defines the insights from the results, the applicability of the framework and methodology and threats to validity. Section "Conclusion and future work" concludes the review report and provides direction on future work.

Background

This section establishes some basic definitions used throughout the paper. It further reviews the history of the software quality improvement efforts. The section also explains why software engineers commit errors during the software design from this research's perspective.

Error, Fault and Failure

Software engineering literature often has competing definitions for the same terms. Based on IEEE Standard 24,765 (ISO/IEC/IEEE 24765:2010) [6] and Leszak [2], this research uses the terms:

- *Error* as a root cause of software design faults and failures. It is a failure of human cognition or failure of the process in problem-solving, planning, or execution during the software design process.
- *Fault* as the manifestation of an error recorded in a software artifact. Faults, when not identified and fixed, can lead to software failure.
- *Failure* is the incorrect execution of a software system for example, unexpected operation or wrong output resulting in a crash etc.

One error may cause several faults. Similarly, one fault may be a result of multiple errors. Lack of domain knowledge [2] is a software design error. This error can occur when a software designer does not understand the business domain while designing the software. In an example order management system (defined later in Sect. "RQ2 How can the Design Error Information Gathered in Response to the Question Above be Organized?"), the software designer lacks the domain knowledge of order management and its related processes like order lifecycle. This lack of understanding can result in software design faults like missing key attributes in UML class diagrams for the order object or data flow diagram that indicates the order lifecycle. The attributes like the status of each order line and payment status of order determine the overall status of the order. These software design faults can lead to failures in the

delivered software if not addressed during the design process. The system could determine incorrect order status. The users will face difficulty in understanding the accurate status of their order. Figure 1 shows the relationship between software design errors, faults and failures in the delivered software.

Software Quality Improvement Efforts—History

Orthogonal Defect Classification (ODC) [7] is the most used mechanism to classify software fault and changes. It provides a structured approach to identifying software problems. It adds a structure to the traditional Root Cause Analysis (RCA) approach [2, 8, 9]. This process allowed software engineers to analyze a failure, find the root cause(s), and potentially address them. These techniques are more reactive rather than preventative. They start late in the software development when the users have already found many faults. The fault investigation process starts after software failures have occurred. Faults are relatively expensive to fix. Leszak [2] took this further and created multiple ways to classify the software faults earlier in the software lifecycle to improve software quality and reduce the costs to incorporate a fix or change.

Despite all these efforts, as published in CISQ reports, the cost of poor-quality software has not declined enough over the years [3]. The traditional approaches are not adequate to improve software quality. They do not stress enough to address the underlying error or the root cause. They rely on fixing the faults that surface after the software is in the testing or is delivered. It is vital to focus on "the state of mind" of a software engineer in the context of software engineering processes and collect insights into human behavior for error prevention [10–12]. Reason [13] defined a plan as a series of actions required to achieve a goal. Reason classified human errors into cognitive failures during planning and execution. A Plan can be adequate or inadequate. Human actions can be intentional or unintentional.

- Execution Errors: These errors happen when the plan is adequate, and the human action is unintentional and does not stick to the plan. A slip of action or a memory lapse are two kinds of execution errors.
  - Slip: These occur due to a lack of proper attention in routine or procedural activities required to carry out a planned action.
  - Lapse: These occur due to memory failure or incorrect application of acquired expertise to carry out a planned action. Instances like forgetting to do something or losing the sequence of actions in the plan can result in such errors.
- Planning Errors: These errors occur when the plan itself is inadequate. Even though human actions align with the plan, errors are bound to happen.
  - Mistake: People with a lack of knowledge and experience tend to make mistakes. Mistakes also happen because of biases and limitations caused by wrongful interpretation of a problem.

Human Errors can cause a software designer to make wrong assumptions and provide incorrect software design. Human error research plays a significant role in Software Engineering processes. Research on human errors help in identifying, understanding, correcting, and avoiding the software errors [4, 5]. It is essential to focus on the software design phase of the software development processes. A focused approach in preventing errors would allow software engineers to improve the software quality. A good quality product will meet customer expectations and increase customer satisfaction levels.

Humans tend to exhibit the same behavior under a given circumstance or context. For example, drivers buckle themselves before triggering the ignition in a car. These "default" behaviors may lead to errors during the exception. For example, a driver may have forgotten to buckle themselves and started to drive when they are running late and rushing through to get to work on time. They may have buckled after the car beeped/alarmed. Software designers follow a pattern for software design based on their experience or what has been found successful. These past experiences may fail and often lead to cognitive breakdowns. Reason [13] proposes that errors can become recurring and predictable. This recurrence

Fig. 1 Relationship between software design errors, faults, and failures

and predictable nature allow software engineers to create error mitigation plan(s) as relevant based on the nature of failure(s). Based on the analysis, software engineers should proactively use mitigation techniques and prevent the failure from happening again.

Practitioners like Wiegmann and Shappell derived "The Human Factor Analysis and Classification System" (popularly known as HFACS) [14–16] based on Reason's work. The aviation industry used HFACS model to assist with the investigation of errors and prevention them from happening again. Several industries have benefited from the HFACS model created by these specialists and improved quality control practices. HFACS pointed out errors caused by organizational influences and processes beyond human errors. Leszak [2] mentions software design problems caused by the software project and reviews in addition to human errors. This research uses the contributions of these well-known models for software design error classification with a premise to apply similar approaches to benefit the software design process.

Research Method

In this systematic literature review, the authors analyzed software design problems reported in the software engineering literature with an objective to capture the underlying root causes of the software design faults. The authors drafted a set of research questions (Table 1) to guide the review. The following subsections describe a review protocol used to select the primary studies for data extraction and synthesize a review report.

Research Questions

The primary objective of this study is to identify different types of software design errors (i.e., the root causes of design faults) with a motivation to identify avenues for improvement in the existing approaches and improve software quality. The secondary objective of this study is to classify the identified errors using an error classification-based methodology to create a taxonomy (Sect. "Review Results"). The following research questions reflect the primary and the secondary objective.

Search Strings and Source Selection

The following search strings represent keywords from the research question RQ1 (Table 1) and their synonyms. The authors incorporated different types of design (e.g., interface design, database design and user-interface design etc.) into the search string to get a comprehensive set of studies.

S1: ((Software OR system OR application) AND (design OR architecture OR component-design OR interface-design OR database-design OR user-interface-design) AND (Problems OR issues OR concerns OR obstacles))
S2: ((Root causes or causes) AND (flaws OR defects OR faults) AND (Software OR system OR application) AND (design OR architecture OR component-design OR interface-design OR database-design OR user-interface-design))
S3: ((Obstacles OR issues OR concerns OR problems) AND (transforming OR transition OR translate) AND (requirements specifications OR analysis model) AND (design specifications OR design model OR architecture design))

The authors used three databases (IEEE Xplore digital library, ACM digital library, and Science Direct) to search for papers and minimize redundant results. These databases were accessible and provided an advanced search capability to create search strings with keywords. They include journals and conference papers focused on software engineering empirical studies and software quality. Multiple search statements, created using the combinations from the search string, resulted in 12,404 unique papers from the selected databases. The search focused on the publications after the year 2000 with the intent to find a comprehensive list of studies.

Quality Criteria

The following two quality criteria assessed the quality of relevant papers following the guidelines of Kitchenham [17].

Table 1 Research questions

#	Question
RQ1	What software design errors have been reported in the software engineering literature?
RQ2	How can the software design error information gathered in response to the question above be organized into a taxonomy?

*RQ2 is a meta-question that uses the information gathered from RQ1 and well-established error frameworks from Reason [13] and HFACS [14–16] to create a taxonomy

QC1: Contribution to software design quality: This criterion evaluated a paper for its contribution to the software design errors or root cause of software faults and improving the quality of the delivered software.

QC2: Number of citations: This criterion checked if a paper had more than 0 citations or not using Google Scholar based search.

Inclusion–Exclusion Criteria

An inclusion–exclusion criterion (Table 2) ensured that the results are relevant for further review in this research.

Review Process

Figure 2 shows the procedure applied in this systematic literature review to identify the primary studies relevant to the research focus and creating a review report. A three-step review process eliminated papers unrelated to the research focus area. Application of Inclusion/Exclusion (Table 2) criteria in parallel kept the results relevant. Only 13 papers were relevant from the three selected databases. Section "Lack of research focus on the software design errors" describes the reason for such a short number of available studies in this research area. The three-step process further evaluated (snowballing) the references of the 13 papers. Snowballing added four papers (indicated using ^ in the references) to the relevan paper list. The quality criteria (Sect. "Quality Criteria") performed a quality assessment on the 17 relevant papers. Although all papers passed QC2, two papers did not pass the QC1. The final list of 15 "primary studies" (indicated using ** in the references) underwent data extraction and analysis.

Data Extraction

A data extraction form template Appendix B (Tables 11, 12) ensured a consistent and accurate extraction of relevant content from the selected list of papers. The process began with extracting individual software design problems from the 15 primary studies (indicated using ** in the references), their root causes (errors) and error descriptions from the identified list of papers. The authors analyzed the data extracted from the primary studies and identified the software design faults and software design errors. The authors reviewed data extract forms for the quality of

Table 2 Inclusion and exclusion criteria

Inclusion criteria	Exclusion criteria
• Papers that focus on using errors for improving software quality • Empirical studies (qualitative or quantitative) of using error information in the software development lifecycle • Papers that provide errors, mistakes, or problems in the software development lifecycle • Papers that provide error, fault, or fault classifications • Empirical studies (qualitative or quantitative) that provide causal analysis or root causes of software faults	• Papers based only on expert opinion • Papers not related to any of the research questions • Preliminary conference versions of included journal papers

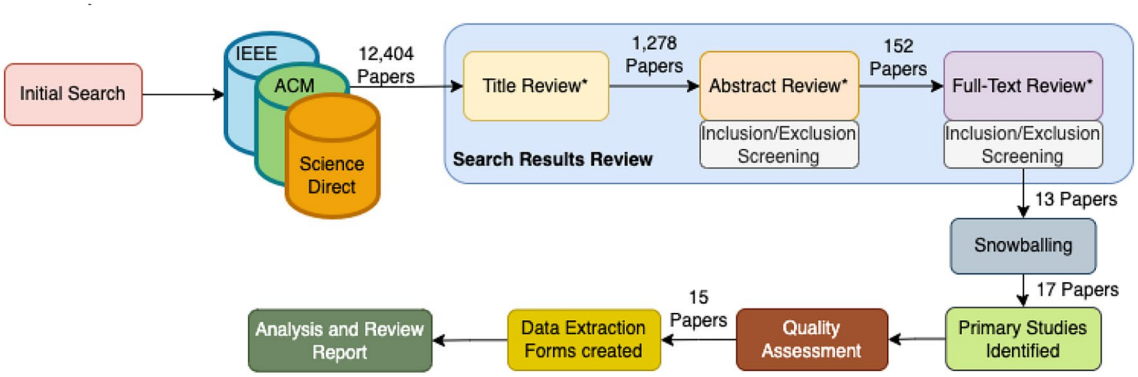

Fig. 2 Systematic literature review process applied in this study. *During title, abstract and full-text review steps, an inter-rater reliability (IRR) test between two authors was performed on a sample set of papers to remove any bias from the review process. The IRR was 90% during the process

information and later synthesized into findings for reporting the review.

Review Results

This section is organized based on the search findings for the two research questions defined in Table 1.

RQ1 What Types of Software Design Errors or Problems can be Identified from the Literature?

The authors reviewed the primary studies to extract software design faults and errors reported by other researchers to obtain an initial list of software design errors. The authors analyzed the list of errors to eliminate the irrelevant ones and errors that looked similar. Table 4 shows the final consolidated list of errors. These errors are committed by software designers during the software design and lead to a software failure(s).

RQ2 How Can the Design Error Information Gathered in Response to the Question Above be Organized?

Software designers commit errors during the software design process. Several other factors play a significant role in breakdowns beyond cognitive reasons. The authors derived a classification (Table 3) for the errors based on well-established error frameworks from Reason [13] and HFACS [14–16]. The similarities and the origin of error is the basis for this classification.

Table 3 Software design error classification system: high-level categories

#	Category	Description
1	Cognitive load errors	Errors of this type occur due to the physiological state of the designer. The amount of stress or fatigue a designer has during the software design process can lead to such errors
2	Knowledge-based errors	Errors of this type are related to a lack of knowledge awareness during the planning. Software design is a human-centric process, and it is a human tendency to make such planning errors. The designer does not have enough knowledge, leading to the development of plans that can cause faults in software design
3	Organizational influence errors	Errors of this type happen due to a lack of support and pressure from the upper-level management. These can also be attributed to interruptions or delays caused by organizational changes, approaching deadlines, time crunch resulting in a poorly conceived design
4	Process errors	Errors of this type occur due to non-compliance in routine tasks or deviation from software engineering processes. Based on the urgency and necessity, software designers tend to deviate from the standard processes to achieve the goals

Table 4 Software design errors classified using high-level categories

#	Error category	Software design error name
1	Cognitive load errors	Lack of preparation [2]
2		Overlooking a design element [2, 28]
3		Clerical error during translation of requirements specifications [20–22]
4		Loss of information from stakeholders [4]
5	Knowledge—based errors	Lack of knowledge about business domain & related processes [2, 5, 10, 23, 24, 29]
6		Lack of tools knowledge [2, 5, 18, 24]
7		Lack of knowledge about software design standards and technical know-how [2, 18, 21, 25, 26]
8		Bounded rationality when choosing third party products [2]
9		Wrong assumptions about stakeholder opinions/problem space [4, 23]
10	Organizational influence errors	Unavailability of resources required for software design [2]
11		Unavailability of training (on software design) [10, 27]
12		Upper-level management's functional fixedness (a fixed mindset) [20]
13		Time pressure from approaching deadlines [2, 29]
14		Miscommunication/Misunderstanding within teams [5, 10, 21, 23, 24]
15	Process errors	Lack of process conformance during design documentation [2, 5, 24, 28, 29]
16		Introduced with a repair or maintenance activity [2, 10]

All the errors found during the review process were grouped by the error category as shown in Table 4.

Order Management System (OMS) Overview: The authors leveraged an example Order Management System (OMS) used by a retail business to explain the errors identified in this research. An order management system: (1) captures orders via multiple channels like online website, call center, store point-of-sale applications from domestic and international markets. During the order capture, the system (2) provides a real-time view of available inventory of items, (3) keeps customer information for personalization, (4) captures payment information using a secure checkout experience, (5) allows customers to track order and view order history. OMS orchestrates the order through its fulfillment lifecycle after the order capture. At a high level, (6) the system sources inventory, (7) schedules the order and releases it to the warehouse, (8) tracks shipments through the Pick/Pack/Ship process. The OMS system facilitates (9) payment settlement, (10) Returns and Exchange processes and (11) feeding data to various downstream systems to report against the order data. The system (12) archives the transaction data to keep up with the growing data. From the high-level list of activities, it is clear that the OMS is a highly integrated system within a business where the order is the main activity.

Subsections "Cognitive Load Errors" through "Process Errors" describe a taxonomy of software design errors based on the error classification mentioned in Table 3. These subsections provide detailed description of each error identified in Table 4 to make the software design error information more consumable and understandable with examples.

Cognitive Load Errors

Errors of this type occur due to cognitive loads like stress, fatigue, inattention etc. These errors are the *failure of execution* in Reasons [13] human error classification. First, a **slip** is a *failure of execution* where the agent fails to carry out a well-planned step because of inattention. Second, a **lapse** is a *failure of execution* where the agent fails to carry out a well-planned step because of a memory failure. Software design is prone to errors caused due to the physiological state of the designer because of its human-centric nature. The amount of stress or fatigue the designer has during the software design process can result in errors. Table 5 shows software design errors caused by cognitive loads along with their descriptions, an example of the design error using the order management system and an example of failure in the delivered software caused by the design error.

Knowledge-Based Errors

Errors of this type are related to a lack of knowledge available for planning. These are the *failure of planning* in Reasons [13] human error classification. A mistake is a failure of planning error that occurs when the plan is inadequate to achieve the goal. Software design is a human-centric process, and it is a human tendency to make such planning errors. The designer does not have enough knowledge, leading to the development of plans that can cause faults in software design. Table 6 shows software design errors caused due to failures of planning along with their descriptions, an example of the design error using the order management system and an example of failure in the delivered software caused by the design error.

Organizational Influence Errors

Several organizational factors impact productivity and create limitations for software engineers. Organizations go through a restructuring that often cause delays and interruptions. Workers are interrupted, thereby reducing productivity and the generated outcome. Budget constraints limit the availability of tools/resources for software designers to do their job. Other factors that contribute errors are organizational policies/culture, constant pressure to deliver from upper management etc. Flawed communication, poor coordination, poor planning, and teamwork can also result in poorly conceived design. Table 7 shows software design errors caused by organizational influences along with their descriptions, an example of the design error using the order management system and an example of failure in the delivered software caused by the design error.

Process Errors

Software designers and organizations often record errors but fail to apply the lessons learned in a subsequent initiative. Errors creep into software design due to failure to resolve known issues by taking corrective actions. Software implementations often tailor the software engineering processes suited for the project. Errors can creep in due to non-compliance or deviation from the software engineering processes. It is human tendency to behave or take actions depending on circumstances. Therefore, errors can occur due to decisions taken by software engineers during exceptional situations. Table 8 shows software design errors caused by process-related problems along with their descriptions, an example of the design error using the order management system and an example of failure in the delivered software caused by the design error.

Table 5 Software design errors caused by cognitive loads

Software design error name	Software design error description	Example of software design error	Example of software fault	Example of software failure
Loss of information from stakeholders	These errors occur when the designer misplaces/forgets information received from the stakeholders. Losing such information can cause designers to create the design without sufficient details, leading to faults in software design	Retailers typically buy products from multiple vendors and sell them to end consumers. The stakeholders discussed that each product has different part numbers for selling to the customers and purchasing from vendors. The solution designer did not take appropriate notes to indicate that every product two-part numbers	The software designer lost the fact that every item should have a mapping between selling item number and vendor item number in the software design. Therefore, the designer did not create an association between the Item object in the class diagram with corresponding vendor-related information like vendor item number, vendor name etc. The software implementation does not have a way to tie items to its source vendor based on the design	There is no way to tie the returns for defective items to the vendor who supplied the item. The operations team at the warehouse does not know where to send the defective returned items in this situation. The system does not understand the mapping between vendor information and the item to provide vendor information. This inventory is scrap and creates a financial loss to the retailer
Lack of preparation	Software designers appear for design discussions with the stakeholders without enough preparation due to fatigue. In such incidents, the software designer forgot to ask relevant questions about the system under design due to inadequate preparation. Therefore, the designer committed errors and led to a faulty design	The legacy ordering process (spreadsheets) had many years of order data in the spreadsheets. The designer did not prepare questions on the legacy process due to fatigue. Therefore, the designer did not ask if the users would need to look up their older (historical) orders in the new order management system (OMS). Consequently, the designer did not design a process to load the historical data from the spreadsheets in to the OMS	The software designer did not create flowcharts and process maps to convert historical order data into a new format and to load historical order data into the new system based on their notes from the meeting. The development team did not implement historical order lookup module due to a lack of relevant design artifacts	The application started with an empty database when the system was launched and had no orders from the previous system. The business teams could not decommission the legacy process because users had to refer to the older data stored in the spreadsheets for any inquiry about or updates to the older orders
Clerical error during translation of requirements specifications	Software designers refer to requirements to create or update software design artifacts. They actively translate requirements into software design based on their understanding, assumptions and discussions. If the designers are not careful when translating the requirements to design, transcription errors can occur and lead to incorrect translation of correct requirements	The requirements specified different shipping carriers for domestic vs international orders. The software design supported shipping of the orders in the order management system. The designer was not careful while translating the requirements, and there was no difference in design for domestic vs international shipping providers	International shipping processes are usually different compared to domestic. Integration interface for international shipping have added requirements to track customs. Because of the translation errors, the designer did not differentiate between the carriers. The application did not support additional integration requirements of the international shipping provider process	Shipping an item from one country to another requires additional checks and balances by the international carriers. The international orders did not move further in the shipping process due to a lack of appropriate integration

Table 5 (continued)

Software design error name	Software design error description	Example of software design error	Example of software fault	Example of software failure
Overlooking a design element	Software designers often tend to forget or miss low-level details of a design element (such as the relationship between objects or attributes in an object in a class diagram) due to distraction or stress. This type of software design error occurs due to an oversight or inattention by a software designer while creating or updating the design	Shoppers are sent coupon codes for discounts via email or paper mail. Due to too many relationships between shopping carts and other objects, the designer forgot to specify the relationship between the shopping cart and coupon code in the design of the OMS system and therefore coupon was not in the shopping cart workflow	The design for the shopping cart does not provide a process flowchart for coupon validation and processing. The developers did not create a provision to enter a coupon code in the shopping cart or the checkout pages	Shoppers do not get to use coupon codes to get discounts on their purchases. They had to call the call center to receive discounts if needed
Loss of information from stakeholders	These errors occur when the designer misplaces/forgets information received from the stakeholders. Losing such information can cause designers to create the design without sufficient details, leading to faults in software design	Retailers typically buy products from multiple vendors and sell them to end consumers. The stakeholders discussed that each product has different part numbers for selling to the customers and purchasing from vendors. The solution designer did not take appropriate notes to indicate that every product two-part numbers	The software designer lost the fact that every item should have a mapping between selling item number and vendor item number in the software design. Therefore, the designer did not create an association between the Item object in the class diagram with corresponding vendor-related information like vendor item number, vendor name etc. The software implementation does not have a way to tie items to its source vendor based on the design	There is no way to tie the returns for defective items to the vendor who supplied the item. The operations team at the warehouse does not know where to send the defective returned items in this situation. The system does not understand the mapping between vendor information and the item to provide vendor information. This inventory is scrap and creates a financial loss to the retailer

Table 6 Software design errors caused by knowledge-based planning issues

Software design error name	Software design error description	Example of software design error	Example of software fault	Example of software failure
Lack of tools knowledge	Software designers follow procedures and use software design tools (Visio, draw.io etc.) they have used before to create or update a software design based on their skills and experience. This type of software design error occurs when software designers are new to a software design tool or do not understand the design tool, its usage and limitations	The software designer didn't know how to use a UML tool available to them to create class diagrams. The designer used the tool to show objects and relationships required to complete the design	Software designers did not establish a proper relationship between different object classes in the Order Management System. The designer established a 1-to-1 relationship instead of 1-to-n relationship between the order and payment method. Therefore, the software implementation did not account for supporting multiple payment methods for a single order	Customers sometimes have gift cards to use for purchases. They expect the system to use the gift card prior to paying any balance using their credit card. The user must either pay using a credit card or have enough balance in the gift card to pay for the order since only one payment method was allowed
Bounded rationality when choosing third-party products	Detailed analysis and proofs-of-concept are required to validate the compatibility and compliance between the software under design and the bundled third-party products and libraries. This type of software design error occurs because of including another product during the software solution design without in-depth analysis or when bundling a third-party product/library without reading the accompanying documentation	Mailing-address validation is required while capturing an order to ensure that user enters a valid address to ship the items. The designer selected a third-party address validation service provider and included it in their design without thorough analysis	Software designers created designs for the user profile and checkout pages to ensure that any mailing addresses entered by customers are always validated. Based on the design and proposed pseudocode, the developers integrated the OMS application with the third-party mailing address validation service	The address validation service provided data validation for only the US-based addresses. It did not provide any validation for international addresses. The users can enter any generic address in the address fields when a country other than the US is selected
Lack of knowledge about business domain & related processes	Software designers may not understand the business domain and the associated processes. Therefore, the designer does not know how to design the software workflow to support business processes like order capture or order return/exchange	Software designers lack the domain knowledge of order management in the retail industry. The designer did not understand the exchange process of an item as a single transactional process. An exchange includes the return of an item by the customer, followed by a replacement with another item (same or different)	The software designer created the process maps for the exchange process as a two-step process from a transaction perspective. At first, the user must return an item to get credited and purchase the replacement item as a new sale later. The implementation team adhered to the documented design instead of making exchange a single transaction	Typically, customers expect an exchange to be one single transaction. A single transaction allows the customer to use the credit from the returned item towards the purchase price of the replacement item. The customer only pays the difference in the price. The customer must pay for the exchanged item at the time of purchase because of the two-step process. The system processes the return as a separate transaction. The two-step process design delays the credit to the customer based on their bank's ability to apply for the credit after the system has issued the refund

Table 6 (continued)

Software design error name	Software design error description	Example of software design error	Example of software fault	Example of software failure
Lack of knowledge about software design standards and technical know-how	This software design error occurs when the software designers do not understand the software design standards or best practices (e.g., modularity, immutability, availability, and data management) while creating the design	The software designer did not understand the performance aspects of an enterprise software application. An order management system collects large volumes of data depending on the time of the year, promotions etc. Querying/ fetching large amounts of data from the database can result in performance issues, resulting in slowness in response for the system. The designer designed OMS to pull order history without enough performance analysis	The software designer did not plan for optimal performance for different operations on objects in the OMS. The Order history data pull did not account for pagination when data has grown over time. Therefore, the developers did not implement order data in multiple page views when displaying order history	The system provided a slower response on the order history page with a growing number of orders due to lack of pagination. The database could not respond fast enough to provide better performance because orders kept filling up the database
Wrong assumptions about stakeholder opinions/problem space	Software designers interact with business stakeholders to complete the process of software design. During the interaction, the designers can misinterpret what stakeholders expect. This type of error occurs when a software designer makes certain assumptions about the stakeholders' point of view/ opinion in software design without clarifying them with the stakeholders	In the existing business process, when the items are not in stock, the order is considered as backordered until the new inventory arrives in the warehouse to ship. Direct shipping delays the order shipments and has been an area that business stakeholders want to improve with the new OMS by allowing vendors to ship the item directly (drop ship) to the customer. However, the software designer assumed that the process would remain the same	Software designers did not create the drop-shipment flowcharts and process maps to automatically create drop ship orders with vendors to handle backordered items. Drop shipment capability would have allowed vendors to ship the items directly to the customer. The developers did not create any such provision in the system	The system does not route the order to vendors and creates backorders. The purchasing team reviews the backordered list and manually create orders with vendors. This manual process delays the fulfillment process. The purchasing team has additional work when the order volumes increase during the peak times

Table 7 Software design errors introduced due to organizational influences

Software design error name	Software design error description	Example of software design error	Example of software fault	Example of software failure
Upper-level management's functional fixedness (a fixed mindset)	The upper management make functional decisions related to products and offerings that directly impact the underlying software design. Business strategy around and sales and marketing drive these management-level decisions. Software designers are typically not consulted for these organizational strategy decisions. Therefore, software designers are forced into creating their design around a fixed/rigid business strategy, often leading to faults in software design	The upper management decided to go to market with two software products for eCommerce and Customer Service. While eCommerce would be the self-service channel, the Customer Service Representative (CSR) users will use the software to take orders over the phone or email. CSRs will also help customers with modification, returns, refunds etc. The go-to-market strategy pushed the software designer into creating two parallel modules for order management processes instead of re-using the same module due to software lifecycle, licensing, and other issues	Software designers had to introduce redundancy in OMS design to create two independent software products with their software lifecycle independent of one another. Two development teams worked on the design without much overlap and did not re-use modules	Whenever a user calls to report an issue, CSR cannot see the issue because the screens for the end-user and CSR are different with different information. There is no way for CSR to see what the user sees Whenever there is an issue in order capture, the developers must fix the problem in both modules (the end-user module and the CSR module). The designer could have designed the two modules using components and similar screens. The team would then fix the problem in the component where the issue is and avoid the double work
Time pressure from approaching deadlines	Software designers are under constant pressure from upper management to deliver due to approaching deadlines or even reduction in budgets. Software designers might fail to look at a broader picture or miss certain design elements due to a lack of focus to meet approaching deadlines. This results in a poor-quality software design with faults	Software designers did not design a process to force users to update the password after every [x] number of days to keep the user account secured to meet the approaching deadlines	The software designer did not document the background process that would check for the last change in the password and notify the user if the password has expired. The development team didn't program the system to force the user to change the password to keep it updated	Shopper are allowed to use a password that never expires. Having a perpetual password made the system less secure and vulnerable
Unavailability of training (on software design)	Different organizations have different work cultures. Some organizations do not have a learning culture or do not cultivate skill development due to cost-cutting measures. Software designers lack the necessary skills to create a good quality software design. This lack of skills is primarily because of the unavailability of appropriate training processes	The software designer did not create system context and integration process maps appropriately. The designer did not develop their skillset by attending relevant training courses	The software designer did not create the system workflow diagrams properly. The incomplete design artifact did not specify the system of record or the system that handles the item record and its relevant attributes in OMS	The system does not have established integration with the system of record (product master). The source product master and the OMS should have the item data for selling. Disconnected and disparate systems created two silos of item data leading to data inconsistency

Table 7 (continued)

Software design error name	Software design error description	Example of software design error	Example of software fault	Example of software failure
Unavailability of resources required for software design	The upper management make decisions due to changes in organizational direction and policies. These decisions impact physical resources like hardware and human resources due to cost cutting and layoffs. A limited number of people must assume more responsibility. Therefore, when such changes in the available resources occur, there is an impact in the software design quality	The management team did not provide a raise and bonus to the employees due to cost-cutting within the organization. Software engineers from the team left for a better job opportunity and perks. The remaining team members assumed the increased workload under pressure	The designer worked with limited resources available to them The designers did not create sequence diagrams and missed details in interface specifications in the order management system's design. They did not specify the negative test cases in use case design documents. The developers implemented the software based on the design and did not unit test all possible validations	The user interface did not reserve the inventory for the item during the order checkout. Lack of promised inventory increased the number of order canceled by the warehouse because the number of orders exceeded the available inventory
Miscommunication/ misunderstanding within teams	Software designers must communicate with stakeholders from multiple business functions impacted by the software (e.g., sales, marketing, procurement, finance etc.) and the support teams to complete the software design. A good communication with different teams is necessary when the software must interface with other applications to accomplish a process. Errors can occur when a software designer fails to communicate or miscommunicate specifications or processes related to the design within a team or with other teams	One of the payment methods selected for online orders was PayPal. The PayPal team did not commit to any changes required for the interface with OMS. The designer did not communicate their plan of action to create the interface with PayPal within the development team to perform proofs-of-concept	The designer designed the OMS to support the PayPal payment method as an option to pay for orders. The designer discussed the detailed integration design with the PayPal team. However, the designer failed to communicate the details with the development team	The designer intended OMS users to have PayPal as another payment option. The OMS and PayPal should integrate seamlessly to provide a unified experience to the user. The developers created an unpleasant user experience using a pop-up screen to integrate with PayPal. The checkout process added multiple clicks and was not mobile friendly for PayPal

Table 8 Software design errors caused by process related problems

Software design error name	Software design error description	Example of software design error	Example of software fault	Example of software failure
Introduced with a repair or maintenance activity	Software designers are often required to re-design or create new designs around an existing design to address new use cases. The designer should perform a change impact analysis to evaluate the impact on the overall software solution and processes. Software design errors can occur when a software designer introduces an error in the system while addressing another design problem without a detailed analysis	The designer initially designed the application with the concept of a unique "User-ID" for login. This capability allowed users to create multiple accounts with different logins. A user report identified that customers created multiple user accounts with the same email address using different User-IDs. The designer attempted to fix this issue by making the user enter their "full email address" as the login	The designer initially designed the User Information class with a unique User-ID attribute in its class diagram. The designer made a modification to use email as the User Id because email is supposed to be unique per person. However, the designer did remove the email attribute from the user object and introduced redundancy	Due to two email fields (one for user id and another for email), the users had to provide the email two times during registration. Some users provided two different email addresses. The users were not allowed to update their email because the user id was a non-editable field
Lack of process conformance during design documentation	Software designers create design artifacts that include design diagrams, documents etc., as a part of the software design creation or update process. These artifacts are then handed over to the developers to develop the software. These artifacts are shared with the support teams to maintain the software post-deployment. These software errors occur when a software designer does not create detailed enough documentation during the software design process	Different data objects require different kinds of validations for the data fields. While some rules are simple, the others can be complex and require special attention from the developers. The designer did not document the data validation rules on attributes in multiple objects for certain common data types	The software designer did not include validations that were standard in the design. For example, the designer did not specify the rules for email address validation and phone number format validation in the design artifact. These data elements are a part of the address and personal information and should be validated as the user enters them while creating an order in the order management system	The order screen allowed users to specify invalid email addresses (without @ symbol) and phone numbers that are invalid formatted for the selected country. The developers treated the email as a string field and phone number as a numeric field while development because the designer did not document the validation criteria The order confirmation email process failed to send an email to the customer because of an incorrect email format

Discussion

The high-level goal of this research is to propose a framework (elaborated in Sect. "Review Results") using an error-classification approach (Sect. "RQ2 How can the Design Error Information Gathered in Response to the Question Above be Organized?", Table 3) for describing human errors that lead software engineers (designers) to commit errors while creating software design. This section provides insights into the results and additional work done by the authors, along with the key objective of this research.

Lack of Research Focus on the Software Design Errors

Out of a large set of research papers returned from the search execution, only 0.12% (15 out of 12,404) included information regarding software design errors consistent with our definition of an error defined in Sect. "Error, Fault and Failure". The terms error, fault and failure have been used synonymously and often with competing definitions in publications. As an example, "Poor choice of platform/architecture [19]" or "Poor modularity of code [18]" are software design faults manifested by an underlying error "Lack of knowledge about software design standards and technical know-how [2]". Therefore, the prior research on software design errors is somewhat inconsistent and

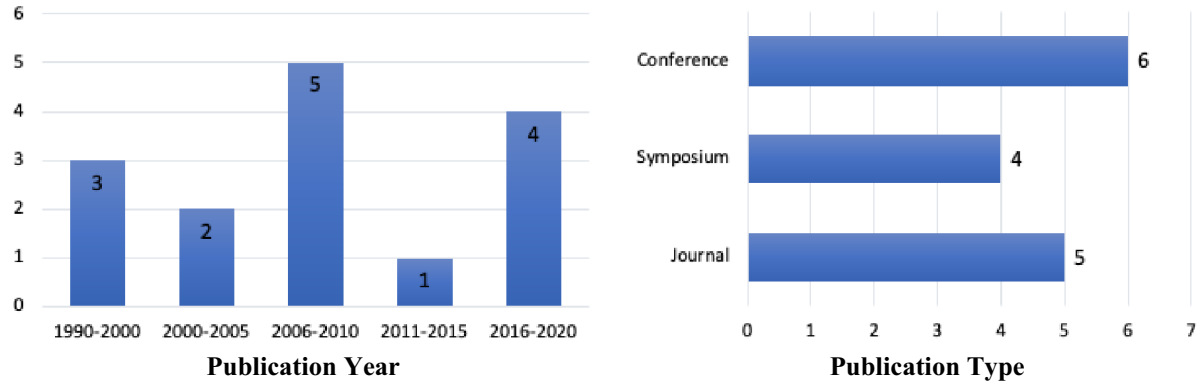

Fig. 3 Distribution of primary studies grouped by publication years and type

Table 9 Software design error distribution by source database

Source database	Number of papers	Number of errors*
ACM	2955	12
IEEE	4466	7
Science Direct	4983	10

*Some of these errors were similar and grouped together into a single error, as shown in Table 4

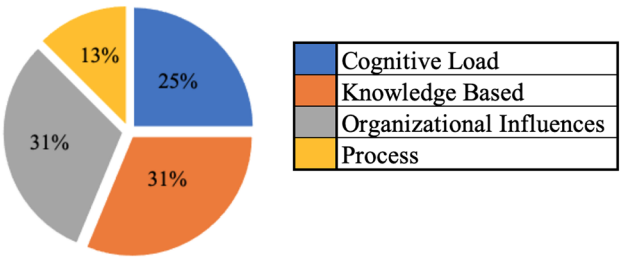

Fig. 4 Software design errors distribution by error categories

further strengthens the motivation behind this review. Figure 3 shows the distribution of primary studies grouped by publication years and type.

The authors thoroughly examined issues/problems reported in the primary studies (including the root causes of the faults). Most of these reported problems were software design faults (manifestation of an underlying error and not the error itself). The researchers have emphasized on reducing the number of programming faults in the delivered software instead of the underlying root cause. A comprehensive list of software design faults (reported in the primary studies) appears in Appendix A. Table 9 shows the total number of papers and software design errors found in the source databases and from the primary studies.

Knowledge Gap: A Major Cause of Software Design Faults

The software design errors found in primary studies indicate that most errors happen due to knowledge gaps in the software practitioners. Organizational influences and human cognition are also significant contributors to software design problems. The papers referred to more sophisticated software projects and applications that could have skewed this analysis in some fashion. The authors lean towards knowledge gap over organizational influences as the primary cause. The authors believe that adequate amount of education can reduce knowledge-based errors. On the contrary, the organization errors which may require significant changes.

Figure 4 provides a view of percentage of errors as classified using the high-level error categories in Table 4. The data indicates that bridging the knowledge gaps through learning or training can improve software quality by 31%. For example, the error Lack of knowledge about the business domain & related processes [2, 23, 29] would not manifest software design faults if the software designers are trained in or made aware of the business domain and associated processes. The other factors highlighted in this review do not impact the individual learning. However, more concrete evidence is required to ensure that resolving one type of error is not dependent on the other.

The data also indicates that reducing the impact of organizational policies and decisions that impact software design will improve overall software quality. Human cognition contributes a sizable number of errors found in this review. This research proposes that software development organizations could benefit from a coordinated effort between software engineering and organizational psychology. Organizations can create proactive approaches to prevent software design errors.

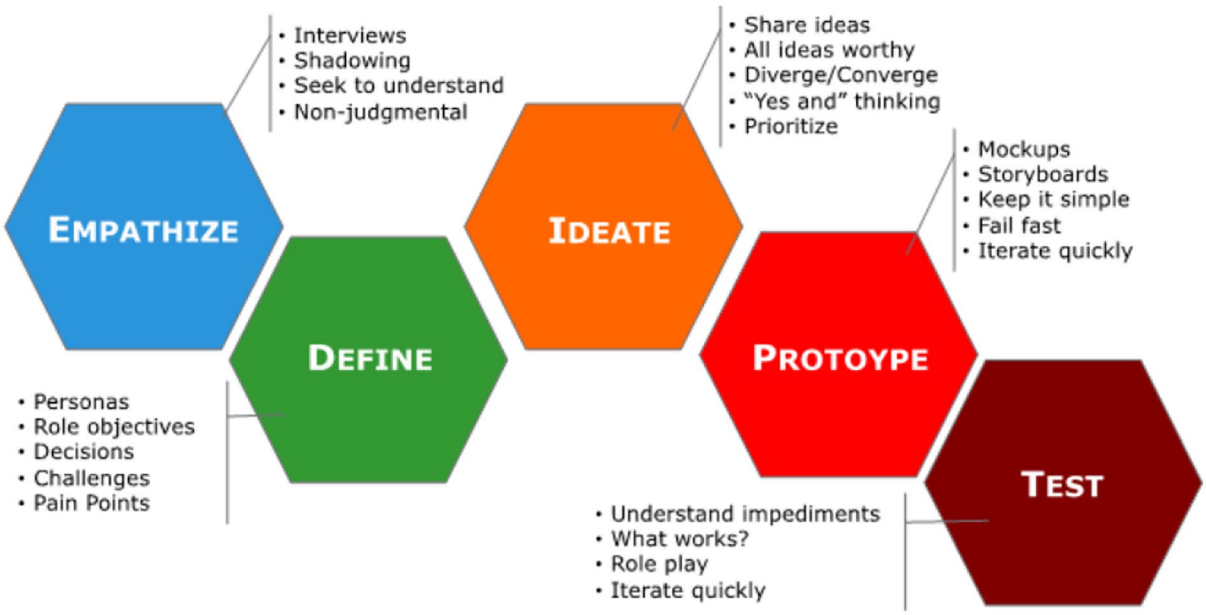

Fig. 5 Software design thinking phases [31]

Software Design Thinking and Software Design Errors

Software quality is defined by how useful it is for its users. There is a strong focus on designing and improving software based on the feedback from its user base because this helps software designers to create a more realistic solution for customers.

"Design Thinking is a methodology applied by project teams for innovation activities focused on satisfying user needs" – Lucena et. al, IBM [30]

Software design thinking provides a purpose to software design process over execution. Software design thinking (Fig. 5) is an iterative process where users give constant feedback to the software professionals to create a winning solution. The process creates a stronger bond between the software professionals and the users of their software. A

Table 10 Mapping software design errors to software design actions

Software design thinking phases is → Software design errors ↓	E	D	I	P	T
Clerical error during translation of requirements specifications		✓		✓	✓
Upper-level management's functional fixedness (a fixed mindset)	✓		✓	✓	
Miscommunication / Misunderstanding within teams	✓		✓		✓
Lack of knowledge about software design standards and technical know-how		✓	✓	✓	
Unavailability of training (on software design)		✓	✓	✓	
Overlooking a design element		✓	✓	✓	
Lack of Preparation	✓		✓		
Lack of tools knowledge		✓	✓	✓	✓
Bounded rationality when choosing third party products				✓	✓
Unavailability of resources required for software design		✓		✓	✓
Introduced with a repair or maintenance activity		✓		✓	
Lack of knowledge about business domain & related processes	✓	✓			✓
Time pressure from approaching deadlines	✓	✓		✓	✓
Lack of process conformance during design documentation		✓	✓		✓
Loss of information from stakeholders	✓	✓		✓	✓
Wrong assumptions about stakeholder opinions	✓	✓		✓	✓

software professional can understand the process and problems of the users better by direct involvement with them.

Each of the software design thinking phase can be a potential home for different kinds of errors. Therefore, this research proposes a secondary organization (Table 10) of the errors based on the software design thinking phase during which the software design error may occur. This secondary organization of errors provides researchers and software engineers with another basis of classification to add and define errors not found during this review.

The Usefulness of Software Design Error Taxonomy

This research proposes a classification of the most frequently observed software design errors committed by the software designers. It provides a framework to define a consistent and well-defined way to represent software design errors. This research leveraged the Reasons Human Error framework [13] and HFACS [14–16] to propose an error classification. The proposed classification is based on the origin of the error to create a software design error taxonomy (SDET). SDET can work parallel to employing HFACS or the Reasons Human error framework. Furthermore, the SDET framework from in this research has the following importance:

- Software Designer's Checklist: Such a taxonomy can serve as a checklist for software engineering practitioners to be more careful. Software engineers should also be mindful of the identified errors and take appropriate actions to prevent them from happening to create an error-robust design for good quality software.
- Software Design Inspection Guide: Software design inspections or audits involve examining the system design and evaluating it against the requirements specifications for completeness and viability. An inspector's job is to find holes/errors in the software blueprint up front to prevent any failures from happening post-implementation. The research can aid design reviewers to be more diligent with their work and focus on the highlighted error classes to search for errors in the software design. They can catch anomalies and send for revision before the development team starts the software implementation.
- Proactive Error Prevention: For each error identified in the taxonomy (shown in Table 4), organizations can perform retrospective error analysis using this framework and spearhead developing countermeasures to prevent these errors from happening.
- Educational Value: Software Engineering students can familiarize themselves with the observed root causes of software failures as a starting point. This research will reinforce the concepts of a good software design thinking process and avoid pitfalls. Further, it will help students to avoid the same or similar errors as they can correlate with the taxonomy and prevent breakdowns in their career as software engineers.

Threats to Validity

The authors used three databases (IEEE Xplore digital library, ACM digital library, and Science Direct) to search for papers, minimize redundant results and gather evidence. A set of comprehensive search strings represented the keywords from the research question RQ1 (Table 1) to ensure completeness of the search process. The search strings also included the synonyms of the keywords and different types of design (e.g., interface design, database design and, user-interface design etc.). Multiple unique search statements, crafted using the combinations from the search string, were used to execute the search. An inter-rater reliability (IRR) test between two authors was performed on a sample set of papers to reduce researcher bias from the review process. The IRR was 90% during the process. This process also ensured that the authors extracted consistent information from each primary study. The authors note that this review does not account for the errors reported in the literature after the search process completed.

Conclusion and Future Work

A careful application of human error research to software design resulted in a theoretically sound framework of software design errors along with error classification. The primary contribution of this work is a taxonomy of software design errors to organize common software design errors using an error-classification approach (Sect. "RQ2 How can the Design Error Information Gathered in Response to the Question Above be Organized?", Table 3) for describing chains of breakdowns that lead software design practitioners to commit errors. The review further indicates that emphasis on software design errors is somewhat fragmented in the software engineering literature based on the primary studies.

This review proposes that the human and organizational psychology and software engineering disciplines should work in tandem to address software quality issues. Using the framework developed in this review will help software practitioners identify common software errors. This framework and methodology can support many aspects of software design error identification to inspire software practitioners to hone their software design skills, create robust design and prevent errors. This framework will help software organizations develop countermeasures, prevent the reoccurrence of the error and improve software quality.

The research opens future work to identify synergies between human and organizational psychology and software engineering. This collaboration may bring efficiencies and improved quality at a reduced cost for software development companies. Furthermore, specific derivative correlations of the proposed framework to the type of software implementations (from scratch or a COTS implementation) or software development methodology (waterfall vs agile) can be derived. As a continuation of this review, it will be helpful for the authors to perform an empirical study to validate the usefulness of this framework to detect and prevent errors.

Appendix A

A Software design faults identified during the literature review.

#	Software design fault	#	Software design fault
1	Extraneous Information [32]	2	Fat/ambiguous interface [32]
3	Cyclic Dependency [32]	4	Scattered concern [32]
5	Ambiguous Interface [32]	6	Procedural violation [32]
7	Delayed Project [27]	8	Incorrect fact [21]
9	No message dictionary for the embedded network [18]	10	Feature envy [34]
11	Flowcharts are used when state charts would be more appropriate [18]	12	Poor modularity of code [18]
13	No methodical approach to user interface design [18]	14	Delegator [34]
15	Design is skipped or created after code is written [18]	16	Architectural tech. debt [28]
17	Incorrect Program codes defining software behavior [25]	18	God class [25]
19	Inadequate security context for informed decisions [26]	20	Lazy class [34]
21	Incorrect Interfaces, issues in data transfer/exchange [25]	22	Data class [25]
23	Brain Class [25]	24	Brain method [25]
25	Poor choice of platform/architecture [19]	26	Feature envy [25]
27	Starting a phase before completing a previous phase [19]	28	Intensive coupling [25]
29	Mistaken assumptions about value or structure of data [33]	30	Dispersed coupling [25]
31	Misunderstanding of external environments like H/w, OS [33]	32	Shotgun surgery [25]
33	Mistake in control logic or computation of expression [33]	34	Classification harmony [25]
35	Embedding sensitive forms on insecure web pages [26]	36	Collaboration harmony [25]
37	Inadequate security contexts and policies [26]	38	Identity harmony [25]
39	Inconsistent information from one part to another in a software artifact [22]	40	Incorrect UML diagrams [18]
41	Necessary information about the system has been omitted from the software artifact [22]	42	No real-time schedule analysis [18]
43	System behavior mismatched with certain system operation specifications [25]	44	No defined software architecture [18]
45	Inadequate policies for user ids and passwords e.g., Hard-to-guess credentials [26]	46	Big class and long methods [34]
47	Confidentiality—E-mailing security-sensitive information insecurely [26]	48	Not securing security-relevant context [26]
49	Failure to undertake effective project reviews and take decisive action [19]		

Appendix B: Data Extraction Form Template (Tables 11, 12)

Table 11 Common data items for extracting information

Data item	Description
Study identifier	Unique identifier for the paper (same as the reference number)
Bibliographic data	Author, year, title, source
Type of article	Journal/conference/technical report
Focus of area	The field in which the research was conducted e.g., software engineering or industrial engineering or cognitive psychology or aviation or medicine
Study aims	The aims or goals of the primary study
Search focus	Search focus/research area (s) of the paper
Study design	Type of study—industrial experiment, controlled experiment, survey, lessons learned, etc
Control group	Yes, no; if ''Yes": number of groups and size per group
Data collection	How the data was collected, e.g., interviews, questionnaires, measurement forms, observations, discussion, and documents
Data analysis	How the data was analyzed; qualitative, quantitative, or mixed
Concepts	The key concepts or major ideas in the primary studies
Higher-order interpretations	The second- (and higher-) order interpretations arise from the key concepts of the primary studies. This can include limitations, guidelines or any additional information arising from the application of major ideas/ concepts
Study findings	Major findings and conclusions from the primary studies

Table 12 Data items relative to each search focus

Search focus	Data item	Description
Software design errors	Problems	Problems reported in the design stage (or any other stages of software lifecycle but relevant to the design stage)
	Errors	Reported errors (if provided in the paper) in software design
	Faults	Faults (if any information provided) in software design
	Mechanism	The process used to analyze or abstract software design errors (select one of the following): Ad-hoc just something the investigators thought up but could have been supported by empirical work showing its effectiveness; or Evidence-based—a notation of a systematic problem in the software engineering process that leads to a specific remediation method; or Theory-based—draws support from research on human errors
Error-fault- taxonomies	Focus	The focus of the taxonomy (i.e., error, fault, or failure)
	Error focus	Yes or No; if ''Yes", What was the process used to classify errors into a taxonomy? During what software lifecycle phase, classification was done?
	Benefits and limitations	Benefits and/or Limitations of the taxonomy
	Evidence	The empirical evidence regarding the benefits of error/fault taxonomy for software quality
Software design inspections/reviews	Focus	The focus of the inspection method (i.e., error, fault of failure)
	Error focus	Yes or No; if ''Yes", how did it focus reviewers' attention to detect errors during the inspection
	Software design phase	Yes or No (Does the study inspect software design documents?)
	Evidence	The empirical evidence regarding the benefits/limitations of the error-based inspection method
Human errors	Human errors and any other relevant classifications	Description of errors made by human beings and classes of their fallibilities during planning, decision making and problem-solving
	Evidence	The empirical evidence regarding errors made by humans in different situations that are related to the software design stage

Declarations

Conflict of Interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Reason J. Managing the risks of organizational accidents. Aldershot: Ashgate; 1997.
2. Marek Leszak, Dewayne E. Perry, Dieter Stoll (2000) A case study in root cause defect analysis. In Proceedings of the 22nd international conference on Software engineering (ICSE '00). p 428–437. https://doi.org/10.1145/337180.337232**
3. Herb Krasner (2022) The Cost of Poor-Quality Software in the US: A 2022 Report", Consortium for Information & Software Quality, https://www.it-cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2022-report/. Accessed 10 Feb 2023
4. Anu V, Hu W, Carver JC, Walia GS, Bradshaw G. Development of a human error taxonomy for software requirements: a systematic literature review. Inf Softw Technol. 2018;103:112–24. https://doi.org/10.1016/j.infsof.2018.06.011.
5. Huang, Fuqun Liu, Bin Huang, Bing (2012) A taxonomy system to identify human error causes for software defects. In: Proceedings of the 18th ISSAT International Conference on Reliability and Quality in Design. https://doi.org/10.13140/2.1.4528.5445
6. Association S, et al. Systems and software engineering—vocabulary iso/iec/ieee 24765: 2010. Iso/Iec/Ieee. 2010;24765:1–418.
7. Chillarege R, et al. Orthogonal defect classification-a concept for in-process measurements. IEEE Trans Software Eng. 1992;18(11):943–56. https://doi.org/10.1109/32.177364.
8. Mays RG, Jones CL, Holloway GJ, Studinski DP. Experiences with defect prevention. IBM Syst J. 1990;29(1):4–32. https://doi.org/10.1147/sj.291.0004.
9. Gaitros DA. Common errors in large software development projects. J Defense Softw Eng. 2004;12(6):21–5.
10. Huang F, Liu B. Software defect prevention based on human error theories. Chin J Aeronaut. 2017;30(3):1054–70. https://doi.org/10.1016/j.cja.2017.03.005.
11. Bickley SJ, Torgler B. A systematic approach to public health—Novel application of the human factors analysis and classification system to public health and COVID-19. Saf Sci. 2021;140:105312. https://doi.org/10.1016/j.ssci.2021.105312.
12. Tang N, Hu H, Xu F, Zhu F. The combination of HFACS and context-aware technology for personalized safety management on construction sites. Int Conf Ind Eng Syst Manag (IESM). 2019;2019:1–6. https://doi.org/10.1109/IESM45758.2019.8948101.
13. Reason J. Human Error [M]. Cambridge: Cambridge University Press; 1990.
14. Shappell SA, Wiegmann DA. The Human Factors Analysis and Classification System (HFACS) (Report Number DOT/FAA/AM-00/7). Washington: Office of Aerospace Medicine; 2000.
15. Shappell S, Wiegmann D. Applying reason: the human factors analysis and classification system. Hum Factors Aerospace Saf. 2001;1:59–86.
16. Wiegmann D, Faaborg T, Boquet A, Detwiler C, Holcomb K, Shappell S (2005) "Human error and general aviation accidents: A comprehensive, fine-grained analysis using HFACS." Federal Aviation Administration, Office of Aerospace Medicine Technical Report No. DOT/FAA/AM-05/24. Office of Aerospace Medicine: Washington, DC
17. Kitchenham B. Procedures for performing systematic reviews. Keele Univ. 2004;33:28.
18. Philip Koopman (2010) Risk areas in embedded software industry projects. In Proceedings of the 2010 Workshop on Embedded Systems Education (WESE '10), Article 5. p 1–8. https://doi.org/10.1145/1930277.1930282**
19. Basili VR, Perricone BT. Software errors and complexity: an empirical investigation0. Commun ACM. 1984;27(1):42–52. https://doi.org/10.1145/69605.2085.
20. Ko AJ, Myers BA. A framework and methodology for studying the causes of software errors in programming systems. J Vis Lang Comput. 2005;16(1–2):41–84. https://doi.org/10.1016/j.jvlc.2004.08.003.
21. Nakajo T, Kume H. A case history analysis of software error cause-effect relationships. IEEE Trans Software Eng. 1991;17(8):830–8. https://doi.org/10.1109/32.83917.
22. Guilherme Travassos, Forrest Shull, Michael Fredericks, Victor R. Basili (1999) Detecting defects in object-oriented designs: using reading techniques to increase software quality. In Proceedings of the 14th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA '99). p 47–56. https://doi.org/10.1145/320384.320389
23. Lutz RR. Analyzing software requirements errors in safety-critical, embedded systems. Proc IEEE Int Sympos Requir Eng. 1993;1993:126–33. https://doi.org/10.1109/ISRE.1993.324825^**.
24. Silva N, Cunha JC, Vieira M. A field study on root cause analysis of defects in space software. Reliab Eng Syst Saf. 2017;158:213–29. https://doi.org/10.1016/j.ress.2016.08.016**.
25. Richard Wettel, Michele Lanza (2008) Visually localizing design problems with disharmony maps. In Proceedings of the 4th ACM symposium on Software visualization (SoftVis '08). p 155–164. https://doi.org/10.1145/1409720.1409745
26. Laura Falk, Atul Prakash, and Kevin Borders (2008) Analyzing websites for user-visible security design flaws. In Proceedings of the 4th symposium on Usable privacy and security (SOUPS '08). p 117–126. https://doi.org/10.1145/1408664.1408680
27. Small M. Design error and reusabilty. SIGCSE Bull. 2007;39(2):185–7. https://doi.org/10.1145/1272848.1272906**.
28. Sanja Aaramaa, Sandun Dasanayake, Markku Oivo, Jouni Markkula, and Samuli Saukkonen (2017) Requirements volatility in software architecture design: an exploratory case study. In Proceedings of the 2017 International Conference on Software and System Process (ICSSP 2017). p 40–49. https://doi.org/10.1145/3084100.3084105**
29. F. Zeng, A. Chen and X. Tao (2009) Study on software reliability design criteria based on defect patterns," 2009 8th International Conference on Reliability, Maintainability and Safety. p 723–727, doi: https://doi.org/10.1109/ICRMS.2009.5270095**
30. Lucena et al. (2016) IBM Design Thinking Software Development Framework, Conference: 7th Brazilian Workshop on Agile Methods (WBMA'2016).

31. Michael Shanks, An Introduction to Design Thinking Process Guide, (weblink: https://web.stanford.edu/~mshanks/MichaelShanks/files/509554.pdf), Hasso Plattner Institute of Design, Stanford, CA USA. Accessed 9 Mar 2023
32. Leonardo Sousa, Roberto Oliveira, Alessandro Garcia, Jaejoon Lee, Tayana Conte, Willian Oizumi, Rafael de Mello, Adriana Lopes, Natasha Valentim, Edson Oliveira, and Carlos Lucena (2017) How Do Software Developers Identify Design Problems? A Qualitative Analysis. In Proceedings of the XXXI Brazilian Symposium on Software Engineering (SBES '17). p 54–63. https://doi.org/10.1145/3131151.3131168
33. Lehtinen TO, Mäntylä MV, Vanhanen J, Itkonen J, Lassenius C. Perceived causes of software project failures—an analysis of their relationships. Inf Softw Technol. 2014;56(6):623–43. https://doi.org/10.1016/j.infsof.2014.01.015.
34. Kreimer J. Adaptive detection of design flaws. Electron Notes Theor Comput Sci. 2005;141(4):117–36. https://doi.org/10.1016/j.entcs.2005.02.059.