The Journal of Systems & Software 198 (2023) 111560

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

The vision of on-demand architectural knowledge systems as a decision-making companion ☆

Maryam Razavian [a,*], Barbara Paech [b], Antony Tang [c,d]

[a] Eindhoven University of Technology, Eindhoven, The Netherlands
[b] Heidelberg University, Heidelberg, Germany
[c] Swinburne University of Technology, The Netherlands
[d] VU University Amsterdam, The Netherlands

ARTICLE INFO

Article history:
Received 29 July 2021
Received in revised form 10 November 2022
Accepted 14 November 2022
Available online 25 November 2022

Keywords:
Software architecture knowledge
Knowledge management systems
Decision-making
Human aspects

ABSTRACT

Cobbler's children do not wear shoes. Software engineers build sophisticated software but we often cannot find the needed information and knowledge for ourselves. Issues are the amount of development information that can be captured, organizing that information to make them useable for other developers as well as human decision-making issues. Current architectural knowledge management systems cannot handle these issues properly. In this paper, we outline a research agenda for intelligent tools to support the knowledge management and decision making of architects. The research agenda consists of a vision and research challenges on the way to realize this vision. We call our vision on-demand architectural knowledge systems (ODAKS). Based on literature review, analysis, and synthesis of past research works, we derive our vision of ODAKS as decision-making companions to architects. ODAKS organize and provide relevant information and knowledge to the architect through an assistive conversation. ODAKS use probing to understand the architects' goals and their questions, they suggest relevant knowledge and present reflective hints to mitigate human decision-making issues, such as cognitive bias, cognitive limitations, as well as design process aspects, such as problem-solution co-evolution and the balance between intuitive and rational decision-making. We present the main features of ODAKS, investigate current potential technologies for the implementation of ODAKS and discuss the main research challenges.

1. Introduction

As society's reliance on software systems increases, so does the complexity of the software that enables and shapes the society. Software systems are ever-larger, more integrated, and more dynamic, thus, designing software applications is also becoming increasingly complex. This paper is about improving the current state of individual design decision-making and its support through architectural knowledge systems (AKS). A fundamental issue in AKS is the lack of information and documentation. Developers are unwilling to capture what they know, but researchers assume that developers are willing to capture and organize information in a structured manner to make retrieval possible and develop AKS tools based on this assumption. If we accept that information is distributed, unorganized and undocumented, then the organization of ad-hoc information into something searchable and meaningful is the problem. Other fundamental issues are human decision-making issues such as cognitive bias, cognitive limitations, as well as design process aspects, such as problem-solution co-evolution and the balance between intuitive and rational decision-making. All these issues may be mitigated by an intelligent AKS.

We create the vision of an *on-demand architecture knowledge system (ODAKS),* because we want to identify and address specific knowledge issues and needs of architects. These issues comprise what developers would not do, how they may be biased by information, and how a new generation of AKS tools might organize information to make knowledge useful. We address issues that are not human related by techniques such as artificial intelligence and ontology (local and distributed). What makes ODAKS unique, however, is that human DM with its facets and limitations is the core of ODAKS and essential in the way on-demand knowledge is provided to the architects. We provide a *research agenda* with summarized features and justifications of this vision, which includes ideas of how these features may be implemented as well as challenges in their implementation and usage.

During software development, many complex design decisions are made by many different stakeholders. Therefore, the sharing of up-to-date design knowledge is essential. Out-of-date or

☆ Editor: Uwe Zdun.

* Corresponding author.
E-mail addresses: m.razavian@tue.nl (M. Razavian), paech@informatik.uni-heidelberg.de (B. Paech), atang@swin.edu.au (A. Tang).

https://doi.org/10.1016/j.jss.2022.111560

incomplete design knowledge can cause sub-optimal design decisions and a consequent software project failure (Charette, 2005; Menezes et al., 2019; Nızam, 2022; Tamburri et al., 2020). Outdated requirements can lead to wrong design decisions (Aurum and Wohlin, 2003). Likewise, outdated information on the use of technology can lead to software with a short life span (Coelho and Valente, 2017) and ultimately software project failure. Many software projects fail due to poor or sub-optimal decisions in the software design process (Aurum and Wohlin, 2003; Nızam, 2022). Recent empirical studies (Tamburri et al., 2020) as well as experience reports (The Standish Group, 2014) suggest that software project failure rates have remained high. It is therefore crucial to support the making of better decisions with better information and knowledge to achieve high quality software with greater chances of success.

We interpret *design knowledge* broadly, by including design elements such as requirements, technical design, detailed designs, as well as the contexts of a system, assumptions, or rationale. Software architecture comprises many activities and design is one of the main activities (Kruchten, 2008). *Architecture knowledge* (AK) is the knowledge needed to support architecture design and decision-making (DM).

Research concerning AK sometimes wrongly assumes that inventing knowledge capture and reuse methods and theories is enough to solve real-world problems, but real-world cases are more tricky (Tang et al., 2011b). Software architects, or architects in short, can work and make decisions independently from each other in different geographic locations, making decisions at different times, and being in different teams. They can also make implicit assumptions, and these assumptions may not be articulated and communicated clearly even when in a face-to-face meeting.

When building a complex system, many factors are considered and many decisions are made. There are several problems that challenge DM due to information and knowledge availability: (a) architects may not have the mental capacity to gather and understand all these factors simultaneously because of cognitive overload (van Vliet and Tang, 2016); (b) as systems evolve over time, the architects who created the initial systems are not necessarily the architects who maintain and change the systems (Perry and Wolf, 1992), especially in organizations where staff turnover is high; (c) decisions can be framed and biased by what information is available to decision makers, meaning that unsuitable information may lead architects astray. Additionally, architects do not always make use of all available information and trade-offs are often ignored (Christiaans and Almendra, 2010); (d) generally human DM can be clouded by cognitive biases and other cognitive issues (van Vliet and Tang, 2016).

Altogether, there is an AK gap between decision makers in a software project, and preservation of AK for reuse is the essence to bridging this gap (Capilla et al., 2016). In addition, DM is a collaborative activity with many decision makers and room for misunderstandings and miscommunication (Rizvi et al., 2015). In this paper, we focus on how architects can be assisted in their *individual* DM by an *Architecture Knowledge System (AKS),* which provides them with relevant information and knowledge. In the current state, we do not deal with collaborative DM.

In this paper, we distinguish between information and knowledge. *Information* is facts and records about a software system such as a specified requirement, design rationale, the performance behavior of a piece of hardware, or a description of how a design looks like. *Knowledge* is the appropriate collection of information for the understanding, design, and construction of a software system. It was suggested that knowledge has three perspectives (Alavi and Leidner, 2001): (a) a hoard of indiscriminant information; (b) an understanding of the information and how they relate to each other; (c) personalization and interpretation of knowledge. These three perspectives lead to the question of relating information to form useable knowledge in an AKS. In the software architecture domain, many have argued that the current approaches to AKS have limitations (Avgeriou et al., 2007; Capilla et al., 2016; Babar et al., 2009). As such, we explore the possibilities and the challenges of *On-demand Architectural Knowledge* in this paper. *ODAK is generated when it is needed*.

In the following, we describe our research agenda in more detail. Section 2 outlines the scope of our vision in terms of work we build on, issues, solutions and suggestions identified by AKS researchers and our main ideas integrating AK management and human DM. In Section 3, we analyze the current AKS issues in more depth. Section 4 discusses human DM issues and how they influence ODAKS, as well as possible solutions to these issues. In Section 5, we present the features of ODAKS as a companion. In Section 6, we discuss technologies that have been developed or are under research and development for implementing ODAKS. We discuss the challenges of building ODAKS in Section 7, from the perspective of interacting with humans as well as the perspective of developing new technologies.

2. Scope of our vision

Robillard et al. have provided the vision of on-demand developer documentation (OD3) where documentation is generated when it is needed (Robillard et al., 2017). They argue that software documentation is costly to create and maintain, and its return on investment is low, so that information, which is needed afterwards, is not captured. Furthermore, curated documentation can only provide answers to some classes of questions, but many other questions would be hard to document and therefore hard to find. They discuss three main challenges for providing OD3: information inference, document request, and document generation. Information inference comprises the selection of relevant information and enhancement for decision support; in particular, establishing links between artifacts, inferring undocumented properties, discovering latent abstractions and rationale. For a document request, an OD3-system needs to support the developer in expressing knowledge needs, capture the design context, provide personalization, and be aware of available information. On-demand document generation comprises intelligent selection of information, summarization, synthesis, and presentation of the document according to the needs of developers. ODAKS share many of the observations made by Robillard et al. (2017). The issues of OD3 are exacerbated in the case of ODAKS because architects deal with a wider spectrum of issues than software developers (Kruchten, 2008). *Therefore, we want to extend OD3 for architects.*

ODAKS are not platforms for design collaboration where architects make design decisions by communicating ideas through the system. However, ODAKS can capture who had designed what in the past and thus assist in finding the designers who made past decisions. This kind of knowledge search facilitates personal communication and a personalization strategy (Babar et al., 2007).

Graaf et al. conducted a study to understand what kind of design questions architects typically ask and how to support knowledge inferencing (de Graaf et al., 2014). They found that there are different types of questions that architects usually ask, and that knowledge needs depend on the design context. *Design context* is defined as the conditions that influence design decisions that are not specified explicitly as requirements or environmental factors (Harper and Jiang, 2015). Examples of design contexts are developers' skills, familiarity with products, budget and time constraints.

Software knowledge management is a multi-faceted issue. First, due to the ill-structured (Simon, 1973; Kunz and Rittel, 1970) nature of software architecture design, design questions can vary widely depending on past personal knowledge. Second, different software organizations have different knowledge needs. Some organizations are technology focused with specializations. For instance, development of health-related devices requires meticulous design specifications, change control, and testing, so developers require complete traceability. Third, the mobility of workers in an organization influences what knowledge an organization wants to retain and serve. Workers in consultancy firms tend to be volatile and developers work with multiple clients. These developers tend to learn new projects and domains in a short time. Developers in specialized domains tend to stay in one area for longer periods and they accumulate personal knowledge, e.g. vehicle software, power industry, financial industries. Differences in the development environment mean that ODAKS need to be adaptable to different knowledge needs. Therefore, there can be no definitive list of design questions and no definitive knowledge dictionary. A knowledge system that is fixed in answering a set of questions can only serve limited needs. Graaf et al.'s study demonstrates example knowledge the architects in an equipment manufacturer seek. We generalize these questions:

Q1. What [module, subsystem] [impacts, realizes, satisfies] a [functional, non-functional requirement]?
Q2. What is the [rationale, reason] behind this [requirement, design]?
Q3. What decision [depends on, influences] a [functional, non-functional requirement, component, module]?
Q4. Is a [feature] of a [component, module] [relevant, useful, compatible] for constructing a [component, module]?

As mentioned for OD3, the information needed to answer these questions, is often not captured. Even when it is captured, it typically comes from different sources. Thus, ODAKS need to be able to relate the scattered information found from various sources in a meaningful way when a design question is asked. Many current AKS discussed in Capilla et al. (2016) can answer some of these information-related questions and limited knowledge questions, if the knowledge relationships are well-defined, and if the information and knowledge is already captured by the AKS, assuming that the knowledge needs of architects do not change overtime. Current AKS lack the flexibility to adapt to a wide variety of information sources and changing knowledge needs. The scattered information and the lack of flexibility *highlight the need for the on-demand aspect of ODAKS*, providing tailored and personalized knowledge through inferencing with the information base in a real-time manner.

In Table 1, we show examples of architects' questions and sketch what ODAKS may do to answer them. These questions refine the questions Q1–Q4 given above by examples we found on the internet or from the experiences of the third author as an architect. Many of these questions have a generic form, like tracing implementation from requirements, but depending on the needs of an architect, the questions can be about something that has never been asked of ODAKS before. Examining the questions and potential answers, we get a sense of the challenges that ODAKS may encounter due to the convoluted nature of design relationships and the challenges of reasoning with design knowledge.

We present the vision of ODAKS as a high-level solution to solve these issues. We extended the three OD3 areas information inferencing, document request, and document generation into four fundamental features to capture, reason, search, and disseminate information and knowledge (see Sections 5 and 6 for details). In particular, we distinguish information collection from knowledge inferencing and extend document to knowledge:

- **(F1) Architecture design information collection and organization:** ODAKS gather all information from all available sources for collation;
- **(F2) Knowledge inferencing and formation:** ODAKS organize and relate information to form knowledge;
- **(F3) Interactive knowledge inquiry support:** ODAKS understand and clarify what an architect is asking;
- **(F4) On-demand knowledge selection and presentation:** OADKS reason with the information and select relevant knowledge to present in a meaningful way to support architectural DM.

Furthermore, we enhance the features to address human DM issues. Human DM is a *mental* process and therefore prone to issues such as cognitive biases and cognitive limitations. Current AKS "dump" the information that they have found and let the architects sort out what can be useful. The information is not context-dependent and therefore very likely either too much or misleading. ODAKS can clarify architects' intents and narrow down the knowledge search space. Furthermore, the *decision process* itself is complicated as, for example, problem and solution co-evolve. ODAKS could mitigate these issues, but may also aggravate them if ODAKS are not designed carefully. Our vision is that ODAKS act as *companions for DM* supporting the architect with an *assistive design conversation*. Assistance here is not only about providing personalized knowledge being tailored for specific knowledge needs of the architects, but also about subtly providing opportunities for architects to engage in a systematic and informed DM process.

That means, ODAKS assist the architect through:

- **Probing**: ODAKS present questions to clarify the state of design and inquiry contexts.
- **Suggesting**: ODAKS present potential areas of interest at a conceptual level. Architects are asked to select what they are interested in to guide ODAKS to hone in on the detail results.
- **Reflecting**: ODAKS present reflective hints and questions that challenge the architect's DM.

Reflection and suggestion can comprise specific mechanisms to mitigate the human DM issues.

Altogether, the novelty of an ODAKS is the holistic approach to information collection, knowledge inference and selection integrated with the assistive design conversation. (a) it can be a comprehensive AKS that gathers information from wide variety of sources; (b) it can inference and reason with loosely-connected information to provide knowledge; (c) it can interact with architects on-demand to clarify their intents; (d) it can make reflections and suggestions to mitigate potential thinking issues. As such, an ODAKS tackles the AK gap as it supports architects with relevant information and knowledge for their DM, and at the same time, it helps to alleviate human DM issues such as cognitive biases and cognitive limitations.

3. Issues of managing architecture knowledge

Architects do not and cannot know everything. However, they need to know relevant information and knowledge to make decisions. Finding relevant information sources, i.e. retrieving information and using it as knowledge correctly is often problematic (Van Vliet, 2008). In this section, based on existing empirical evidence in literature, we discuss four major issues in the collection and retrieval of architectural information and the formation of knowledge.

Table 1
Examples of information and knowledge requests.

	Architect's questions	Answers/ Various sources	How ODAKS may get the answers and the implementation challenges
Information requests	What module(s) realize(s) user authentication? (Q1)	LDAP from design and implementation documents	Currently, architects search for the information in a UML model, wiki-pages, or semantic wiki-pages using the keyword "user authentication". ODAKS would search for "user authentication" and interpret the word "realize" and search a design model (e.g. UML).
	How did we quantify scalability in project X?	X transactions per sec under different loadings, it is defined in the performance specification of project X.	Architects currently look in the relevant section of the performance requirement specification. Similarly, ODAKS would search for "scalability" and scalability-related concepts in the areas of requirement, performance, and test reports.
Knowledge requests	Why do we use LDAP? (Q2)	Design rationale for choosing LDAP. The answer could be technical, conformance, budget related or a combination of other reasons.	Currently, architects manually search for the design rationale in design documentation. ODAKS would need to interpret "LDAP" and its inherent concept(s), namely "user authentication" and "security". Then, it would search in IT policy, requirement, design, implementation, and other places for anything that is related to "LDAP" and its related concepts. The challenge for ODAKS is to understand the underlying design concept of LDAP so that inference about its design rationale can be made.
	If we change requirement X, what components in the system would be affected and how? (Q3)	A trace of all the components and modules that are affected because they directly or indirectly implement requirement X.	System components can be interrelated for reasons such as how they interface with each other, how they affect each other's behavior, their compatibility, and other issues. Currently, an architect would search design documents, models, and implementation notes, or simply test the software to find out, or ask someone knowledgeable. With ODAKS, "affect" could mean different design concepts such as "implement", "interface", "software behavior", "quality attributes", "compatibility", "rationale" etc. ODAKS first interpret the relevant concepts semantically to define and guide the search, and then go to their knowledge base to gather and relate the information.
	How to apply a message polling design pattern without sacrificing code maintainability? (Q4)	Pros and cons, if they exist, of a message polling design pattern and its related maintainability quality attributes.	Currently, architects look into design documents, design notes, source code, object diagrams, and Q&A sites such as Stack Overflow to find if there are any pros and cons of maintainability when implementing message polling. This part of the query is to search for information. It may not yield any answer as a design pattern typically helps rather than impairs maintainability. ODAKS may also find that message polling is typically related to system performance, which is another quality attribute. ODAKS can ask if the architect would want to investigate this quality attribute. To perform such an act, ODAKS need to understand the taxonomy of quality attributes and to be able to reason with it.

Issue 1: Practitioners often have scattered and disorganized architectural information and knowledge. An observational case study with practitioners revealed that the amount of information, their diversity and scattering make it difficult to find knowledge (Graaf et al., 2014). Some issues with the current forms of architectural information and knowledge are:

- **Architectural information comes in different forms:** Codified information can be texts and diagrams contained in formal specifications, such as signed-off requirement specifications and design specifications. It can be structured and contained in databases, such as test cases issue registries. It can be standard notation such as UML diagrams. For open-source software development, they can be contained in discussion threads, posts, and markdowns, all of which are rich sources of development information.
- **Architectural information contains diverse contents**: Information can be about how a design is made, why it is made (e.g. rationale), circumstances under which decisions are made (e.g. design contexts and assumptions), the issues and uncertainties that designers face, evaluation of decisions (e.g. trade-offs, pros and cons, risk analysis), decision impacts (e.g. how one decision is related to other decisions) etc.. It can be contemplations of options,

discussions of issues, and reporting outcomes and behaviors of systems. Architectural information can virtually contain anything that affects a system design.

- **Architectural information has different sources:** Codified information can be drawn on a whiteboard, created in a tool, hand-written on a piece of paper, in meeting minutes, in wikis, in source code comments, in emails, chat messages, issue comments (in an issue tracker), commit messages, meeting notes. Each type of information comes from a different source.
- **Architectural information can be tacit or is lost:** Some information is tacit, that means it is not recorded anywhere and known only to the architects. Other information is lost, e.g. when an architect forgets it or when a person departs the organization.

Issue 2: **It is challenging to get architects to document AK, such as design rationale.** A survey with practitioners revealed that although architects see the value of documenting design rationale, it is challenging due to various reasons ranging from the lack of standards and processes to the time and budget constraints of projects (Tang et al., 2005). An extensive survey with open source software revealed that architects and designers do not capture AK as much as we think (Ding et al., 2014). The analysis of Capilla et al. has identified six difficulties or barriers of architectural knowledge management (AKM) (Capilla et al., 2016). We map the majority of these barriers to this issue 2, while we discuss barrier 2 and 3 in the next issue 3. There are **general barriers with respect to the motivation** for capturing. Barrier 1 (B1) is a lack of incentives for architects to make the effort to document AK; B4 is about the extra effort required to capture AK when the architects' workload is already high; B5 argues that documentation disrupts the architects' creativity; B6 argues that the architects do not see the benefits of AK capture given the costs. These barriers are general cultural and educational issues in software organizations, in which researchers can provide evidence to demonstrate and convince practitioners of the worthiness of knowledge capture. Such evidence is lacking because there are not yet any comprehensive AKS or approaches that can make AK documentation cost-effective.

Even when architects can spare the effort to document what they have done and how a design is implemented, it is unrealistic to expect developers to document trace links between requirements, design, decisions, and implementation artifacts. However, these trace links are necessary to assess change impacts. For instance, if a requirement changes, how would that affect quality attributes and implementation? Would there be any unknown side effects to another part of the system? The issues of documenting trace links are: (a) it is difficult to anticipate the side effects of how one element in a system affects another part of the system without a change scenario; (b) the effort of documenting trace links is huge and impractical. There are solutions for providing and maintaining software traceability, but only for specific development tools and artifacts, e.g. Hübner and Paech (2020) .

Issue 3: Barrier 3 (B3) suggests that there is a **lack of understanding of what information and knowledge to capture** (Capilla et al., 2016). Similarly, based on an expert survey with architects Weinreich et al. (2015) suggest that we do not know what knowledge is important and what formalism can be used to capture knowledge. It is almost impossible for architects who design a system to capture all the explicit and tacit architecture design information and knowledge in anticipation of how maintainers (and other persons who would like to use the information and knowledge) might want to use them. Generally, architects know more than they can tell (Senker, 1995), but they do not know what maintainers might or might not know.

Architects capture AK based on what they think is important and relevant, and do not necessarily know or document the knowledge that maintainers would find useful. The plethora of AK research has so far not been able to solve this knowledge barrier with the many meta-models proposed (Capilla et al., 2016). In current AKS, maintainers are limited by the questions they can ask that are outside of the pre-defined relationships (de Graaf et al., 2014). One reason is that researchers have assumed that *knowledge models and meta-models are fix with pre-defined knowledge elements and relationships*. Typical assumptions are: (a) the information gathered does not change overtime; (b) the knowledge needs of architects are well-known and do not change; (c) the information related to technology and software projects remains constant in AKS models. However, these assumptions do not hold. As examples, architects who used to write formal specifications find themselves writing up wiki-posts. Architects who used to be responsible for high-level designs now find themselves dealing with deployment and operational issues due to DevOps implementation. Architects may find themselves dealing with micro-services due to mergers with new company acquisitions. AKS that cannot grow and evolve through these technological changes can be outdated very rapidly.

Barrier 2 (B2) suggests that there is a lack of tools to actuate the capture. Study of software architects in real-world practice however suggests that this is not only about tool availability or knowledge models. Architects do not know how to capture knowledge, and they do not know what is the "best approach" to represent, retain, and organize "key" AK (de Graaf et al., 2014).

Issue 4: Application of information (and thus the formation of knowledge) is design context dependent. In In a case study Graaf et al. observed that architects who face specific architecture design situations require different information and knowledge to help them make decisions (de Graaf et al., 2014). Design contexts were described as conditions that influence design decisions that are not specified explicitly as requirements (Tang and Lau, 2014). Whilst architects may know design contexts intuitively, the concept is very hard to define due to its highly dynamic nature (Chattopadhyay et al., 2018), and also because there are many contextual factors and combinations (Dybå et al., 2012). Examples of design contexts are environmental factors or socio-economic aspects that influence system design (Bedjeti et al., 2017), as well as technical factors (Dybå et al., 2012). Contextual factors can dictate which information and knowledge is relevant to a design decision. They also are related to what an architect wishes to achieve and what problems are being solved at that time (Tang and van Vliet, 2012). For instance, an architect wants to upgrade the security software to fix vulnerability. She wishes to evaluate why this security measure was not chosen initially. The reason of the original choice may be due to a trade-off that is budget-related. Budget constraint was the design context of the decision then. Let us say that this budget constraint is no longer an issue and the security software is now available. She knows that this software can now be used because the constraining issue has been removed. Knowledge of the old design context helps to facilitate the decision (Tang and Lau, 2014). A typical AKS would use keyword search to find everything that matches exactly or closely to what is sought. An ODAKS, instead, would reason why a question is asked and which areas of design and development would contain the knowledge. Such intelligence involving design context comprehension is currently absent from AKS.

These four issues are interrelated and they create challenges. The lack of codified information and knowledge (i.e. issue 2 and 3), the unstructured and diversified sources of information and knowledge (i.e. issue 1), and the appropriate use of knowledge which is context dependent (i.e. issue 4) all contribute to the complexity of building an AKS. They detail why answering the example questions of Table 1 is challenging.

4. Human decision-making issues and processes

Bhat et al. performed a semi-systematic literature study on architecture DM (Bhat et al., 2020). They found that in 2004–2010, researchers mostly focused on modeling and tools; in 2010–2015 researchers shifted their focus to lightweight documentation and extraction from different sources but recent research has shifted to the human aspects of DM. We argue that this shift reflects the two fundamental issues identified by researchers over the past 15 years: the knowledge gap and human DM issues. In this section, we focus on the architect and DM as a *mental process*. In the following, based on the empirical evidence reported in literature on human DM, we discuss in Section 4.1. the issues arising during human DM and in Section 4.2. the factors influencing the DM process. In designing ODAKS, we carefully need to consider the effects of ODAKS as technical systems, as well as how they might aggravate or alleviate these human DM issues and factors. In Section 6, we discuss techniques to mitigate these issues and assist human DM.

4.1. Human decision-making issues

Design DM is an important *mental* element in every software architecture design (Jansen and Bosch, 2005). Making decisions is complicated, and errors are common because there are many human DM issues. A large body of research has demonstrated the many ways in which humans are irrational and easily biased when making decisions (Chen and Lee, 2003). Studies of DM in the fields of cognitive psychology, behavioral economy, and design studies are essential in understanding how architects make decisions and how they use relevant design information in DM (Arnott and Gao, 2019). The traditional view on DM advocated by philosophers and economists assumes that people are rational decision-makers. These assumptions were based on normative theories specifying that people should always choose the best option, having first evaluated all options on offer. Since the mid-1950s, research studies began to show that *limited mental processing and memory capacity* and other thinking issues affect rational DM (Thaler and Sunstein, 2008). People tend to rely on shortcuts in DM, such as *intuition* and *heuristics,* that make complicated decision problems manageable but that can also lead to error and bias.

The Information Systems field (IS) has a long history of studying the effects of technology-assisted DM such as decision support systems (DSS) on human DM (Arnott and Pervan, 2016). Research has shown that a DSS can influence users by either helping them or leading them to irrational or suboptimal decisions (Arnott, 2006). At the same time, DSS can be designed to help users avoid DM biases, such as a confirmation bias (Huang, Hsu, and Ku 2012) or an availability bias (George et al., 2000). Architectural DM through AKS is a type of technology-assisted DM. Hence, we take the research results that focus on the effects of DSS on human DM issues as an entry point to discuss the potential effects of AKS on an architect's DM.

In the following, we argue for prevalent research areas on understanding human DM. Grounding on the literature in these research areas, we sketch how human DM issues impede DM.

4.1.1. Major cognitive biases influencing architectural decision-making

Cognitive biases are considered deviations from rational DM versus the ideal where a decision-maker considers all the consequences of making a choice (Tversky and Kahneman, 2000). Cognitive biases are inherent in human reasoning. Based on the systematic literature review of cognitive biases in software development (Mohanani et al., 2018), we selected three relevant cognitive biases of architectural DM for our discussion, namely the confirmation bias, the availability bias, and the anchoring bias (Stacy and MacMillan, 1995).

Confirmation bias. Confirmation bias is the tendency to pay undue attention to sources that confirm our existing beliefs while ignoring sources that challenge our beliefs (Mohanani et al., 2018). With the confirmation bias, people are interested in information that fits their beliefs (Huang et al., 2012). Accordingly, the confirmation bias inhibits architects from evaluating all the available information and knowledge fairly and this may result in bad design decisions. Architects can have difficulty in imagining all possible ways in which events can unfold. Because they fail to envision important pathways in the complex net of future events, they can become unduly confident about predictions based on the few pathways they consider.

Availability bias. Another cognitive bias that architects are prone to is the availability bias i.e., a tendency that information which is easier to recall unduly influences preconceptions or judgments (Mohanani et al., 2018). People tend to assign more importance (weight) to recent and important events because these are easier to recall from memory (Kahneman, 2011). Architects rely heavily on their experience and knowledge to make decisions. Their limited ability to retrieve all relevant cases may cause them to make biased judgments and decisions.

Anchoring bias. The anchoring bias is about giving undue weight to initial information (Mohanani et al., 2018). When architects make decisions, they might anchor on past experience, the way information is formulated (Tversky and Kahneman, 2000), or first found solutions (van Vliet and Tang, 2016) without adjusting their decisions as new information is presented to them. Such anchors on experiences, information, or ideas, however, may restrict architects to adjust their thinking to arrive at a more appropriate design solution. Architects can anchor on pre-conceived ideas (Tang, 2011) and tend not to adjust their decisions when the design context has changed, and base judgments on outdated understanding of the design context. Even when they change their understanding, the adjustments are typically insufficient and the final decision can be biased by the initial information (Chen and Lee, 2003).

4.1.2. Cognitive limitations in problem solving

There are limitations on how human minds can solve problems (Simon and Newell, 1972). Such cognitive limitations apply to DM, design, and problem solving.

Working Memory Capacity. The reasoning ability is found to be strongly related to the storage capacity and the control efficiency of the working memory in our brains (Chuderski and Jastrzębski, 2018). The working memory briefly stores representations that the mind is currently working on, and people switch and retrieve memory actively to accommodate new pieces of information when needed. The capacity of the working memory has been shown to be highly correlated to both, the insight problem solving (intuition) and the reasoning (rationality) capabilities (Chuderski and Jastrzębski, 2018). Since the capacity of the working memory is limited, we sometimes seek ways to recall information from external sources. The invention of symbols and written language have helped humans to develop and advance their mental processing capacities (Nelson, 1998). For instance, with writing and exograms (i.e. different means of recordings) we can record experiences, which allows us to recall and associate information. Technologies and computer records provide extensions to human internal memory at a grand scale. These are tools that also change the way we think (Nelson, 1998). Still, humans with their limited memory capacity have to make sense

of all information with such extended memory. Large amounts of externally retrieved information may overwhelm the human's working memory capacity.

Bounded rationality. Herbert Simon first introduced the theory of bounded rationality because the ideal view of rationality was at odds with how decision-makers in real world make decisions (Simon, 1982). Bounded rationality theory stipulates that decision-makers cannot have perfect knowledge of a decision situation and, further, that they are limited in their cognitive and information processing abilities. In addition, they are normally subject to time pressure that means that perfect computations of utility functions (i.e. pros and cons) are not possible. Simon's key insight was that decision-makers' rationality is bounded rather than perfect. Instead of maximizing utility, decision-makers make decisions based on what they have. These decisions are good enough to *satisfice* the goals (Simon, 1996; Tang and van Vliet, 2015).

4.2. Design decision-making process

There are two main schools of thought explaining how designers go about design, each focusing on a certain facet of human DM: (i) intuitive and rational DM focusing on the cognitive processes involved in DM and (ii) problem-solution co-evolution describing DM in two interacting and evolving spaces – problem space and solution space. In what follows, we discuss these two DM process facets. It should be noted that both DM process facets occur in a certain design context distinguished by situational factors that influence the DM process, which include technical (e.g., complexity, technology), social (e.g., personal characteristics, organizational culture), and environmental factors (e.g., uncertainty, market) (Dybå et al., 2012). Since the DM that we are concerned with is about designing, contexts that influence design are also DM situational factors.

4.2.1. Intuitive and rational design decision-making

Like all humans, architects make decisions within and between two cognitive processes or systems (i.e. Dual Process Decision-Making). Kahneman called them System 1 and System 2. They can operate at the same time and can interact (Kahneman, 2011). System 1 is associated with intuition, expertise, and expert judgment, while System 2 is based on the careful evaluation of problems, options, tradeoffs, and rationale.

Naturalistic DM theory, which has similarities to intuitive decision-making (System 1), was proposed by Klein (2008, 2009). Naturalistic DM relies on an intuitive mode of thinking and provides an account of how architects make decisions, different from rationality: through expertise and experience instead of careful evaluation of design options and eventually selecting the optimal one. Using naturalistic DM, architects learn to recognize situations and match them with a set of solutions that they know would work; if the match is not so strong the architects simulate how the actual solution might play out in the new situation and make the necessary modifications (Klein, 2008, 2009).

Research shows that architects use both: intuitive and rational cognitive processes. When using their intuition, they follow a *gut feeling*, a strong sense of knowing or a hunch speaking for one option being the best choice in a given decision situation, while being unable to provide rationale for this gut feeling (Khatri and Ng, 2000). Far from being ineffective or second rate, the fast, intuitive DM, under some circumstances, may lead to superior outcomes compared to rational DM. A few studies have shown that intuitive DM holds the potential to improve software design DM performance (Pretorius et al., 2018; Arnott and Gao, 2019). Research in other disciplines, such as strategic management, cognitive and applied psychology, and new product development, has shown that applying intuitive DM is sometimes better than focusing on rationality and is therefore increasingly accepted as an important complement of rational DM (Hodgkinson and Sadler-Smith, 2018). Moreover, two generic types of factors indicate which cognitive process – being intuition, rationality or both – is beneficial : (i) the architects' dispositional style (Epstein et al., 1996), and (ii) the situational factors (Dijksterhuis and Meurs, 2006); (Dijksterhuis and Van Olden, 2006).

Dispositional style. The use of intuitive or rational DM is situated in and depends on the skills and experience of the architect (Kahneman, 2011). Both, rational and intuitive DM, can be used by a practitioner at any time, in any order, and even simultaneously. Still, people tend to rely on one or both styles in a specific way (Epstein et al., 1996), known as their *dispositional style*. In the field of psychology, there are reliable instruments to measure individual differences in the dispositional style. One example of such an instrument is the Rational Experiential Inventory (REI) questionnaire as developed by Epstein et al. (1996) which is a well-validated scale. An architect who is dispositioned to the rational cognitive process has the tendency to carefully analyze the problem at hand and consider all alternatives with the goal to choose the best option. An architect who is dispositioned to the intuitive cognitive style, on the other hand, has a tendency to follow his/her gut feeling speaking for one option being the best choice in a given situation, for instance based on past experience, while being unable to provide rationale for this feeling.

Situational Factors. The use of intuition and rationality in DM is situated and its potential benefits and drawbacks depend on many factors. For instance, *complex problems* involving a great deal of information (Dijksterhuis and Meurs, 2006; Dijksterhuis and Van Olden, 2006) or with *unstable environments* (Khatri and Ng, 2000) appear to especially benefit from the application of more intuitive DM (Klein, 2009). Likewise, it is preferable to use intuition when the designer is *experienced* with the problem (Kahneman and Klein, 2009). On the other hand, when assessing design ideas or design options, a more rational approach could be effective (Eliëns et al., 2018). Being part of the design context, situational factors can determine when the use of either approach is beneficial or detrimental.

4.2.2. Design decision-making as problem-solution co-evolution

Most design problems are ill-structured (Simon, 1973) and the architectural design process is about interpreting problems, generating solutions, and evaluating decisions (Ralph, 2015). This process of traversing between *problem space* and *solution space* is known as *problem-solution co-evolution* (Maher et al., 1996). The iterative problem-solution co-evolution means that architects understand and define the problem space based on what is possible in the solution space. Solution options and decisions are propositional (if…then…) statements, bridging a proposed problem and a proposed solution. When a solution is not good enough, the process iterates: back to solution alternatives, or even back to the problem space when there is a perceived need to introduce new problems or criteria (Dorst, 2019).

In their study of software developers in action, Purao et al. (2002) described the problem space as the metaphoric space that contains mental representations of the developer's interpretation of the user requirements, and the solution space as the metaphoric space that contains mental representations of the developer's specific solutions. Design reasoning elements used in QOC (Maclean et al., 1996), IBIS (Conklin and Begeman, 1988) and different design rationale models (Dutoit and Peach, 2000; Tang et al., 2010) also describe the problem and solution spaces. Zimmermann et al. created tools (ADMentor) and templates to capture information in the problem, solution, and decision spaces

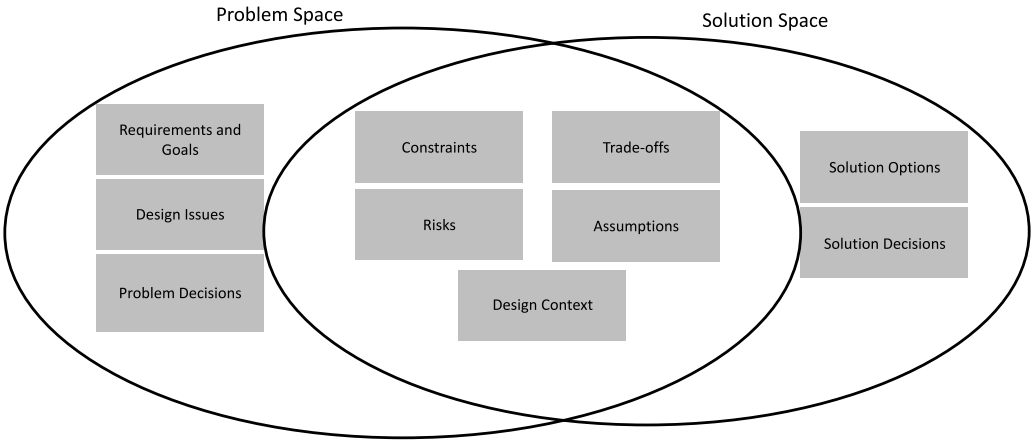

Fig. 1. Design reasoning elements residing in the problem space and the solution space.

(Zimmermann et al., 2015). The design reasoning elements, which architects need to consider and reason with, are generalized and summarized in Fig. 1.

As design DM is an iterative process (Tang et al., 2011a), it is also the process of traversing between the design reasoning elements in the problem space and the solution space (see Fig. 1): interpreting, framing, and reframing *Requirements and Goals*, *Design Issues*, and *Problem Decisions* (Schön, 1984) until they are sufficiently coherent to generate *Solution Options and eventually make Solution Decisions*. *Constraints, Risks, Trade-Offs, Assumptions, and Design Context* need to be considered in both spaces. This process can happen in an architect's mind or in a team where architects have conversations about the design issues, solution options, compare alternatives, ask more questions, raise doubts, and challenge various ideas with the aim of finding a solution for a problem.

5. On-demand architectural knowledge systems

In this section, we present ODAKS in detail. We outline in Section 5.1. how the features F1 to F4 of ODAKS deal with the AKM issues presented in Section 3. We discuss in Section 5.2. how ODAKS mitigate the human thinking issues described in Section 4, and in Section 5.3., we discuss challenges of our idea.

5.1. ODAKS' solution for architectural knowledge management

For ODAKS, we extend OD3's vision based on our distinction between information and knowledge and the specifics of AK. Therefore, we distinguish information retrieval and knowledge inference and formation. Furthermore, the focus is on knowledge generation, which does not necessarily come in a fixed document format. The difference between using a document-based information retrieval system and using ODAKS is that a document-based retrieval system retrieves fixed entities or documents to show what is recorded there. Retrieving from ODAKS that have inference capabilities, on the other hand, can yield incrementally and iteratively constructed knowledge through interaction with the architect. Fig. 2 shows the features F1-F4 of ODAKS which are described in the following. We reference the issues from Section 3 to motivate the features.

(F1) Architecture design information collection and organization: The sources of the information are specifications, design notes, code comments etc. created by architects and developers within a development organization, or open-source project mailing lists and Q&A forums such as Stack Overflow. Since information is scattered and incomplete (see Issue 1 in Section 3), architects may not know where the relevant information is stored and miss it.

To mitigate this problem, ODAKS need to extend the capturing of relevant information by automatically traversing and crawling through all potential sources, to form knowledge. In a traditional information system, such information is indexed and searched using keywords. However, architecture information can be kept in diverse formats, comprising not only written words but also source code, models, diagrams and voice (Issue 1). Auto-translating of hand-written drawings on digital whiteboard (Mangano et al., 2014) and commercially available automatic transcription software can also provide a rich source of information. Additionally, design rationale and decisions voiced by designers can be captured and incorporated into UML (Capilla et al., 2020). The diversity of document formats requires sophisticated processing to extract the information. First, ODAKS need to be able to find and retrieve available information through searching and crawling. It is common that some information is not recorded anywhere but knowing who was involved in the design and development and may know something allows the inquirer to go to the person or team to make inquiries. This is the personalization information retrieval strategy (Babar et al., 2007). ODAKS can capture the names of those who write the notes and are involved in the development discussions. Second, ODAKS need to have the capabilities to parse and extract information automatically. An example is to breakdown and capture individual entities and their relationships in UML models for knowledge representation and inferencing. Such information can then be used to construct the semantics for sense making (see F2 below).

(F2) Knowledge inferencing and formation: ODAKS organize and relate design information into knowledge. Design knowledge can be many things, it can be how a design is constructed (e.g. interfaces and modules), or the design rationale of a design, such as design concerns, potential design options, and trade-offs (Tang and Vliet, 2009), or it can be many cross-cutting concerns in a design (Nuseibeh, 2004); Nuseibeh, Kramer, and Finkelstein 2003). The needs for knowledge vary largely and cannot be fully anticipated. To-date, we understand and capture these design relations by relying on humans to annotate them. This is time-consuming and largely impractical (Issues 2, 3 and 4).

Alternatively, ODAKS can use ontologies and be trained to relate design relations from texts and models through some machine training algorithms. Collected design relations are partial and they provide knowledge of how and why a specific design is made. For instance, ODAKS knows that LDAP is related to authentication and authentication is a security issue, therefore it can infer that LDAP is about security implementation. ODAKS

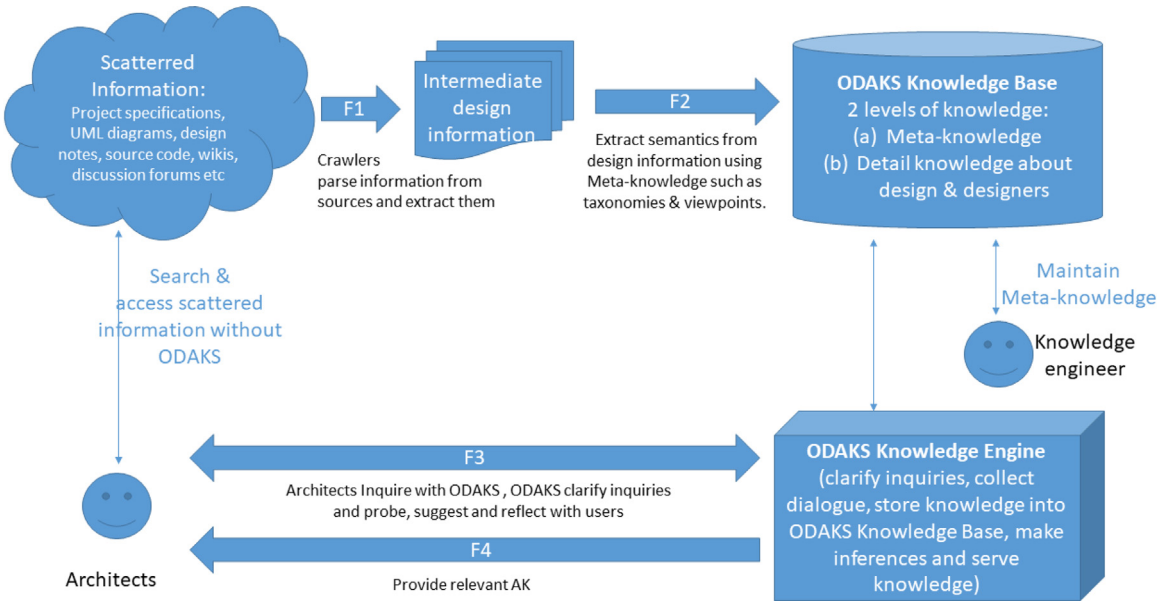

Fig. 2. ODAKS' solution concepts.

also know about a design relationship which represents a security requirement. When the two pieces of information are combined during inferencing, ODAKS can reason with the security requirement to LDAP implementation. Such design relations connecting different pieces of information and concepts can be established by explicitly modeling these relationships in a knowledge base. This is a simple example of allowing ODAKS to serve different queries through ontologies.

In order to support flexible *knowledge reasoning*, there is a need to represent useful relationships between pieces of information effectively. Ontologies provide an explicit formal specification of terms in a domain and specify relations among these terms (Noy and McGuinness, 2001). An ontological model contains related information that can be used for querying and inferencing. An analogy is the use of a relational database, the tables and the indexes of a database are defined and customizable, allowing different SQL queries to be made with the database model. Here the analogy ends. Ontological systems can do more, they can be used to store loosely related pieces of information and support querying through inferencing. The inference engine can follow information threads to find related knowledge. This capability provides flexibility in knowledge retrieval because it can reason with concepts without architects having to understand the data model and explicitly reference the model. An ontological model can be customized to contain relevant meta-models and information for each organization's knowledge needs.

ODAKS can provide the mechanism to support querying with ontological models. It can contain common design relationships such as architecture viewpoints, design rationale model, traceability model, quality attribute model, and so on. ODAKS can also implement domain- and technology-specific models defined for specific development organization knowledge needs (Issue 1). One can implement knowledge relationships which only ODAKS can gather, like information in a distributed team environment implementing microservices where more than one team collaborate in an architecture design. Locally structured information can be contained in local ontology models. Many design relationships already exist and known. Some of them are standard and well-understood design relationships called design viewpoints, and they represent cross-cutting design concerns (Nuseibeh et al., 2003; Bedjeti et al., 2017). Viewpoint representation of design concepts and relationships form the *meta-knowledge* in ODAKS. Such ontology models can be provided by third parties and distributed. Distributed ontology models such as standard viewpoints and common architectural knowledge can be integrated and used with local ontology models.

(F3) Interactive knowledge inquiry: When architects inquire an AKS, they typically enter keywords and key phrases but they do not provide the purpose and intent of an inquiry. The knowledge that the architects seek through such queries is tacit in several senses of the word (Schön, 1992): (i) AK is more than what architects can verbalize. Architects can describe what a design is but cannot *fully* describe the implications of a design unless specific questions are asked. (ii) architects tend to give inaccurate descriptions of what they know or seek, and (iii) architects can best recollect the knowledge by putting it in the mode of doing design for a specific context and having a conversation about it.

The search and inference results may therefore not be directly relevant to the intents of the architect. Issue 4 shows the need to understand the problems and the design contexts being addressed in order to help qualify a design inquiry. One way to achieve this is that ODAKS have intelligent interactions with the architect. A design context can be expressed as user goals and intents (Rose and Levinson, 2004; Hu et al., 2009) and hints at the nature of an inquiry (Cao et al., 2009). For example, if an architect inquires about memory management, ODAKS need to find out whether the architect is concerned about memory management of the data structure, or about the creation and destruction of objects, or about the operating system's memory management, or hardware memory management. This can be achieved because the ontological model is able to make explicit which class(es) of concept the term memory appears in.

ODAKS need to interact with the architect to determine about the design context of the inquiry (Tang et al., 2018; Razavian et al., 2016a). The additional design contexts allow ODAKS to narrow their focus to find the relevant design knowledge (see F4 below). To take care of the human DM issues described in Section 4, we stipulate that these interactions between ODAKS and architects should be an **assistive conversation,** which is described in detail in Section 5.2.

(F4) On-demand knowledge selection and presentation: In order to provide relevant knowledge and information to an architect, ODAKS need to perform the following tasks: (a) based on an

architect's inputs, ODAKS reason with their knowledge base and retrieve relevant information and knowledge. This task requires searches with machine reasoning. ODAKS' awareness of the query contexts can enable them to narrow the search and focus the search by selecting concepts and relationships that are relevant to the goals. In task (b), ODAKS prioritize the outputs by computing relevance and probability. In task (c), ODAKS present the results to the architect. ODAKS present results in the order of importance and are mindful of not overwhelming an architect with details. When the architect needs the details, they are displayed and eventually lead to the original document and models. In step (d), ODAKS can prompt the architect intelligently as well as seeking more inputs (see F3) such as asking for more design problems and design contexts to direct reasoning. This prompting of inputs can be based on the ODAKS' experience and knowledge of what other architects have often asked, or on the role of the inquirer.

In Section 6, we explore technologies and on-going research that potentially enable ODAKS.

5.2. ODAKS as companions through assistive design conversation

This section focuses on what ODAKS can do whilst interacting with architects, through F3 and F4, to provide better support for human DM. In our vision, ODAKS are *companions* to a human decision-maker, as illustrated in Fig. 3. By companion, we mean that ODAKS need to accommodate how architects make decisions in terms of *the DM process and issues.* To do so, ODAKS use *assistive design conversation* to support the architect's design DM.

Assistance is about providing personalized knowledge tailored for specific knowledge needs of the architects. Through the assistive design conversation, ODAKS can narrow down the search space to the state of design and inquiry contexts and systematically offer prioritized presentation of knowledge. Assistive conversation is also about subtly providing opportunities for architects to engage in a systematic and informed DM process. Examples are suggesting possible areas of interest and providing reflective hints and questions that challenge the architect's DM. As such, the assistive design conversation helps mitigating DM issues and assists architects in their exploration of problem space and solution space, therefore giving architects a better chance for good decisions. While ODAKS support architects' DM, they, however, are not DM agents making design decisions themselves. What makes ODAKS unique, however, is that human DM with its facets and limitations is the core of ODAKS and essential in the way on-demand knowledge is provided to the architects.

Three mechanisms have been introduced in the literature for mitigating human DM issues and assisting in the design DM process – *debiasing*, *nudging*, and *reflection* (Arnott and Gao, 2019).

Debiasing is a process whereby the negative consequences of a cognitive bias are reduced or mitigated. There are a number of techniques that can be used to undertake debiasing (Kahneman et al., 1982; Keren, 1990). Debiasing techniques require an understanding of the mechanisms underlying the particular bias.

Nudging is a mechanism suggested by Thaler and Sustein to lead the DM process taking into account the inherent issues of human DM, that is, the negative consequences of cognitive biases and cognitive limitations (Thaler and Sunstein, 2008). The idea behind nudging is to subtly provide opportunities to guide the decision maker towards preferred options without restricting the options or forcing certain outcomes or taking away the freedom of choice (Thaler and Sunstein, 2008). Thus, nudging is a gentle way to steer decisions and it can be employed to deal with debiasing.

Reflection. Reflection is a thinking process that challenges and revises the DM process. Donald Schön suggested that a reflective conversation such as *reflection-in-action* is done through asking relevant questions over the course of formulating a solution (Schön, 1983). McCall suggested that design is a critical conversation between ideation and evaluation (McCall, 2010). Van Manen suggested that reflection is a kind of deliberation and thinking at a higher and deliberate level and it is one way to achieve higher rationality (Van Manen, 1977). While (Weinmann et al., 2016) discuss reflection as the goal of nudging. This is not discussed in (Jesse and Jannach, 2021). In our view, reflection is a very important element of DM support.

In Section 6.2., we discuss mechanisms that potentially enable ODAKS's features. In the following, we sketch how the features can be integrated for an assistive design conversation.

Fig. 3 shows four different elements that have a role in the architect's DM with ODAKS. (i) *Human DM Issues* of architects including *Cognitive Biases*, *Cognitive Limitations,* and *Dispositional Style*; (ii) *Design DM* of architects that engage both DM process facets of *Intuitive and Rational DM* and *Problem-Solution Co-evolution*; (iii) *Design Context* that are inputs into the DM process; (iv) *Assistive Design Conversation* is what ODAKS add to assist DM. The facilities include *Probe*, *Suggest*, and *Reflect*.

Whilst ODAKS cannot act like a human architect to carry out a real design conversation, ODAKS may fake intelligence (Parnas and Clements, 1985) with assistive conversations. We propose three types of assistive design conversations:

A. **Probe**: ODAKS present questions to clarify the state of design and inquiry contexts as discussed for F3. Clarifications given by architects can help ODAKS scope their inference space (Issue 4). This helps to provide adequate suggestions and reflections.
B. **Suggest**: ODAKS structure potential areas of interest on a conceptual level (i.e. meta-knowledge or viewpoints about a design). Architects are asked to select what they are interested in to guide ODAKS to home in on desirable results. In particular, debiasing and nudging are an important part of suggesting to deal with the issues mentioned in Section 4.
C. **Reflect**: ODAKS present reflective hints that challenge design DM. The reflective hints can be inferred from the DM elements used in previous design decisions (see also Fig. 1 in Section 4.2.2). ODAKS can ask reflective questions to remind the architect to think in different ways and thus, also deal with issues mentioned in Section 4.

Table 2 Provides examples of each type of assistive conversations between an architect and ODAKS based on the examples given in Table 1. Such conversations can arise while architects develop a new system and want to learn from others, or when they maintain a given system and want to understand the rationale underlying the current system. With the appropriate meta-knowledge (F2), the implementation of an assistive design conversation is based on the interactions with architects through intelligent dialogue (F3) and useful knowledge presentation (F4). Each conversation illustrates how ODAKS incorporates human DM with its facets and limitations in the way knowledge is provided to the architects.

Conversation 1: Probing and clarification of contexts. When an architect asks an ODAKS for knowledge, a dialogue is started whereby the architect states what s/he wants. Based on the input, ODAKS infer from their knowledge space to either (a) serve knowledge that is specific to the inquiry, or (b) to clarify the contexts with the architect for guiding inference and search. In case of (b), ODAKS ask the architect to enter more information. The additional inputs provided by an architect could help to

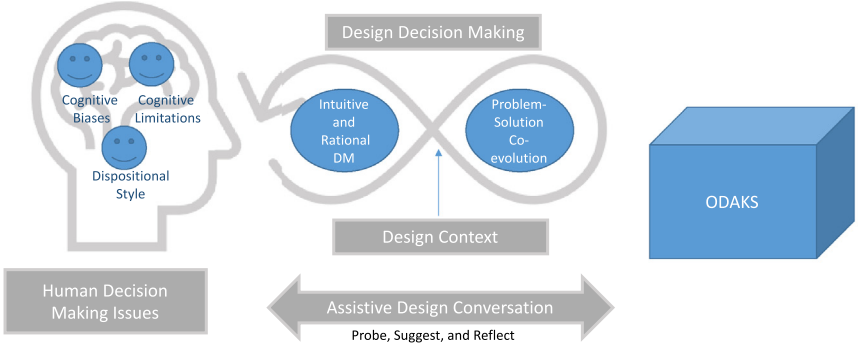

Fig. 3. ODAKS Assistive Human Decision-making by probing, suggesting and reflecting.

Table 2
Example Assistive Conversations between ODAKS (O) and Architect (A)

Conversation 1	**ODAKS' Intent – Probe**	**ODAKS' processing steps**
A: What modules would be affected if I change requirement X?		ODAKS (F4) search the knowledge base to identify requirement X. ODAKS also search for other similar terms that are of the type "requirements". They then uses related meta-knowledge such as "implementation" and "quality attributes" to find modules that may be directly or indirectly impacted.
O: Requirement X relates to a number of sub-systems. Which sub-system or quality attributes do concern you?	ODAKS need to clarify the context of the requirement so that it can narrow the scope to search and inference.	ODAKS ask user for more inputs (F3)
A: "Security"		ODAKS (F4) search for all security-related modules that are directly or indirectly related to X. They use all relevant (meta-knowledge) security aspects and viewpoints such as "implementation", "software dependency", "related requirements", to do its inferencing.
O: Displays a list of relevant viewpoints: "implementation", "software dependency", "related requirements". A click of any viewpoint by the architect would display modules directly/indirectly related to requirement X that are security-related.		ODAKS (F4) present relevant viewpoints that are related to security and affected by requirement X.
A: ⟨⟨browse and select knowledge⟩⟩	ODAKS collect user feedback for future use.	ODAKS monitor the browsing behaviors (F3) to see what information and knowledge the architect looks at. ODAKS can store what an architect has read.
O: What is useful to you?		ODAKS ask the architect to mark the useful answers found by ODAKS (F3).
A: ⟨⟨Marked selection⟩⟩	ODAKS collect user feedback for future use.	ODAKS store the original query, the useful answers identified, the meta-knowledge, such as contexts and quality requirements that was used in the inferencing to narrow the scope of future inquiry processing. Viewpoints such as "Useful Query", "Typical Concerns" relations can be used to guide searches.
Conversation 2	**ODAKS' Intent - Suggest**	**ODAKS' processing steps**
A: How did we quantify scalability in past project X?		ODAKS search knowledge base to identify the scalability metrics in project X (F4). They cannot find anything in project X. It proceeds to carry out a general search using the terms "quantify" and "scalability" in all projects and finds mentions in project B. ODAKS may also look into previous searches with meta-knowledge such as "Typical Concerns" to look for knowledge.
O: Scalability was not quantified in project X. Scalability is quantified in project B. Would you like to view information?	ODAKS suggest alternatives as it cannot directly answer the question	ODAKS suggest an alternative source of information to user. ODAKS expect architect to answer "Yes" or "No".

(*continued on next page*)

narrow down the search and inferencing scope when ODAKS reason with their meta-knowledge and knowledge.

Conversation 2: Suggestive conversation. ODAKS can make suggestions based on initial discoveries. ODAKS use inference and search capabilities to find potential alternatives that an architect seeks. ODAKS can structure and list those alternatives as suggestions for the architect to select. The selection of an alternative helps to shape the scope of the exploration, and ODAKS then target the details to find more information and knowledge specific for that user. Users of ODAKS are identifiable through login and thus, ODAKS may collect user information (e.g. project id and technical concerns) through the interactions. User activities can also be collected to help ODAKS understand a user's areas of interests and what knowledge is typically sought. Usage history

Table 2 (*continued*).

A: "Yes"		ODAKS start searching (F4) for "quantify scalability" in project B using meta-knowledge to broaden search in concepts that are related to scalability and measurements. Let us say ODAKS find a number of scalability related concepts. ODAKS present three conceptual alternatives (F3) to the architect to narrow down exploration. Each suggestion is an encapsulation of a design concept.
O: Scalability can be measured by (i) the numbers of concurrent users; (ii) the numbers of concurrent requests; (iii) virtual machine spin-ups. Which one would you like to see?	ODAKS want to reduce cognitive load compared to traditional AKS that unloads all information indiscriminately.	ODAKS present relevant knowledge to guide selection of relevant knowledge that matters most. ODAKS expect any or none of the options.
A: ⟨⟨i and ii⟩⟩	ODAKS collect user actions for future use.	ODAKS store selection identified by architect for future inquiry processing. Viewpoint such as "Useful Query" can be used to capture query and response topics between architects and ODAKS.
Conversation 3	**ODAKS' Intent - Reflect**	**ODAKS' processing steps**
A: How to apply message polling design pattern without sacrificing code maintainability and readability?		ODAKS search knowledge base to find "message polling" design. ODAKS also search for quality criteria, code maintainability and readability. During the search, ODAKS finds through meta-knowledge "Quality Attributes", that scalability and performance are often of concern when dealing with message polling.
O: Displays the results of (i) applying message polling mechanism as a design pattern, (ii) maintainability of code, (iii) readability of code and polling mechanism	ODAKS display knowledge directly relevant to the question	ODAKS present the knowledge
A: ⟨⟨Browse and select⟩⟩		ODAKS store selection identified by architect for future inquiry processing.
O: Have you considered design issues scalability and performance here?	ODAKS can at this juncture prompt reflection on other design issues such as scalability and performance. Such reflection may guide the architect to further explore solution space, or even back to problem space to further consider new problems of scalability. Such reflection is in-line with debiasing strategies that enable "alternative thinking" that in turn mitigate anchoring and confirmation biases.	ODAKS associate "message polling" to the solution option which belongs to the solution space. Maintainability, readability, scalability are design issues in the problem space. ODAKS know about these meta-knowledge design reasoning elements. Based on this knowledge, ODAKS infer that the architect is relating the solution space to the problem space. At this stage, ODAKS offer a reflection that is associated to the problem space which the architect may find useful. ODAKS frame a reflection in the form of a generic question "Have you considered design issues XXX?" with the relevant information on scalability and performance.
Conversation 4	**ODAKS' Intents - Probe and Suggest**	**ODAKS' processing steps**
A: How does the front-end of software X log event Y?		ODAKS (F4) search knowledge base to find "event Y" and "software X" and "front-end". ODAKS find two different categories of results, distinguished by their platform being Android and iOS.
O: Which platform do you want to know about: iOS or Android?	ODAKS need to clarify the context of platforms so that it can narrow the scope to search and inference (i.e. Probe).	ODAKS ask user for more inputs (F3). ODAKS expect user to answer "Android" and/or "iOS".
A: Android and iOS		
O: Feature X logs Event Y in the front-end. You can find the solution for iOS here and for Android here.	ODAKS provide implementation for iOS unified logger; and provide the syslog implementation for Android.	ODAKS retrieve the relevant designs of event logging for both iOS and Android. The search for logging finds that logging happens in more than one areas: front-end and back-end. Based on the meta-knowledge design reasoning elements ODAKS know that there are other related solution options about logging. ODAKS infer that this might be useful and present them as suggestions.
O: Are you interested in logging mechanisms in the back-end? ODAKS display the results of (i), (ii), (iii).	ODAKS can suggest the architect to consider other related knowledge to see if event Y logging has implications in the back-end (as opposed to requested front-end logging mechanisms). A comparison of similar mechanisms may stimulate the architect to consider different solution options. This is in-line with Case Memory debiasing technique for availability bias (i.e. Suggest), that in turn mitigate availability bias.	ODAKS associate "logging mechanism" to the solution option which belongs to the solution space. At this stage, ODAKS offer a suggestion that is associated to solution space which architect might find useful, namely solution options categorized as back-end mechanisms. ODAKS present relevant solution options and expect any or none of the options
A: ⟨⟨i and ii and iii⟩⟩	ODAKS collect user actions for future use.	ODAKS store selection identified by architect for future inquiry processing.

can be collected and made relevant for future inquiries. One part of the usage history are the areas of knowledge sought by a particular user, or whether the knowledge sought is in certain requirement areas or in specific projects. By presenting only knowledge that matters most to the architect, conversation 2 illustrates how ODAKS assist in cognitive load, compared to traditional AKS that unloads all information indiscriminately.

Conversation 3: Reflective conversation. ODAKS present reflective hints that challenge design thinking. The reflective hints can be inferred from the type of knowledge an architect seeks. For instance, when an architect asks ODAKS for knowledge about message polling, ODAKS infer from their knowledge space that this knowledge is about solution option and reflective questions can be searched in the solution space. This inference can be done based on design reasoning elements shown in Fig. 1. By prompting reflection, conversation 3 highlights the potentials of ODAKS in mitigating cognitive biases such as anchoring and confirmation. While such reflections can assist architects in their exploration of problem space and solution space, and support better DM, it does not mean that ODAKS knows that such cognitive biases have occurred. It merely implies that the debiasing mitigations are offered by the ODAKS.

Conversation 4: Combination of Probing and Suggestive conversations. In assistive design conversations ODAKS combine probing, suggestive and reflective conversation, to tailor the conversation based on the specific knowledge needs while at the same time providing opportunities to mitigate the DM issues. The suggestive conversation illustrated in Conversation 4 offers debiasing availability bias by providing a comparison of similar mechanisms which might stimulate the architect to consider different solution options. Conversation 4 shows an example of such combination. ODAKS can potentially provide much more complex conversations than what we present in Table 2 depending on the extent of the ontology models. Designing such conversations is however challenging, calling for further research.

5.3. Challenges in the vision of ODAKS

In this paper, we provide a research agenda of ODAKS. We are aware that there are several risks associated with the vision and that not all implementation technology is in place so that there are challenges to solve by further research. In this sub-section, we shortly sketch challenges with respect to the idea of ODAKS. Challenges with respect to the implementation of ODAKS, which imply future research, are discussed in Section 7.

The first challenge with the idea of ODAKS is that it rests on sufficient data to collect appropriate information and knowledge from (Issues 1, 2) and do the inferencing. As discussed in Section 3, developers often do not document their work with the necessary details (Issue 3). However, as online remote work has been the norm during COVID-19, electronic data is more available than ever. Even hard-to-capture knowledge such as meeting discussions can now be voice recorded and auto-transcribed and documented, thanks to meetings being online. This can expand the reach of captured knowledge as discussed in Section 5.1. Whilst design documentation by architects may remain a challenge, culturally and practically, we can now access previously untapped data sources through automatic scanning and capturing. This will ease the workload of architects and make more information available to show the value of ODAKS, and thus motivate architects to support automated capture.

The second challenge with the idea is that architects might not like the assistive conversation envisaged and thus, not accept ODAKS. The companion relationship between ODAKS and architects necessitates that architects act with autonomy. Examples of the autonomy of architects could be architects having control in the extent and the form in which the information and the reflective conversation is presented to them. The challenge here is how to create ODAKS in a way that it is not intrusive, respects the concerns and needs of practicing architects, supports them in their various activities, as well as enables architects in exploiting their creativity. First solution ideas are discussed in Section 6.2.

6. Enabling ODAKS

The key technological challenges of ODAKS are to automatically organize scattered information, make sense of them, and organize them such that it can answer architects' queries and to interact with the architect to support human DM. Features F1 to F4 outline the key functionalities of ODAKS, but there can be many ways to implement them. In this section, we examine technologies and research works that are relevant for ODAKS' features. In Section 6.1., we describe the technologies and approaches for the four features. In Section 6.2., we focus on assistive conversation. In order to realize ODAKS' features, one needs to leverage current research and envisage new research efforts in specific areas. The technologies that are reported here involve various types of intelligent technologies such as machine learning, natural language processing, clustering, automatic classification, ontology construction and machine inferencing. These technologies are only available as research tools. As such, they need to be further developed and integrated to realize ODAKS' features.

6.1. Technologies for ODAKS' features

First and foremost, ODAKS should not be seen simply as a store and retrieve solution. Intelligence will need to be built into each of the four features of the ODAKS. Since architects use natural language to describe design and exchange ideas all the time, technologies such as natural language processing (NLP), machine learning, and machine reasoning are needed to extract and process the semantics of such documents and conversations in ODAKS' implementation. Technologies, such as the semantic web, have been used extensively in managing knowledge (Antoniou and Van Harmelen, 2004). The semantic web uses the Resource Description Framework (RDF) (Horrocks et al., 2003) to record relationships between information. RDF is a simple way of relating things to form a labeled graph. Searches of the semantic web rely on traversing and reasoning with the RDFs to find knowledge.

F1 - Collecting scattered design information. Design information can come from many different sources and repositories: a single project depository containing specifications, notes, wikis, emails, design models etc.; from organization depositories that contain multiple IT projects, standards, and guidelines, and from software developer forums, such as Github[1] and Stack Overflow.[2] Vendor release documents and forums can also be a part of the design information. ODAKS need *crawlers* for the collection.

Design information gathered from text documents is simpler to collect and extract, but specialized documents, such as UML models, source code, and forum discussions require specialized crawlers. A crawler needs to interpret the documentation contents and translate it to some intermediate formats so that semantic analysis can be performed on the documents. ODAKS crawlers can parse and select relevant information incremental from previous parses. During the selection process,

[1] https://github.com/

[2] https://stackoverflow.com/

ODAKS crawlers can interpret the contents to see if a document is within a defined scope for selection (Balasubramanian et al., 2009). A scope defines where design information comes from and its purpose. This scope is defined for each specific ODAKS implementation. One content crawling approach is to label web documents by genre. Crawlers or agents may use some weighing mechanism to perform this task (Pappas et al., 2012). Automatic analyzers have been created to analyze and learn various types of design information from public sources such as Stack Overflow and Github (Ahmad et al., 2020). Analysis, consistency checks, and test case generation of diagrammatic models such as UML are extensive. In recent research, different NLPs techniques were tested to mine words and synonyms related to architecture tactics (Bi et al., 2021). This kind of techniques may be adapted to mine different types of design information from documents and communication. However, selecting the right NLP to use matters. Omran and Truede (Al Omran and Treude, 2017) found that the performance of an NLP depends on the source and the task, with a variety of text sources to mine. Thus, ODAKS' designers need to consider what technology to use for extraction in different circumstances.

Extraction from modeling tools such as UML requires special tools that understand the syntax of the modeling language. Xu et al.'s work on UML semantics extraction makes use of intermediate XML representation of UML and translating and importing them into an ontology (Xu et al., 2012). Whilst the syntax in UML and other modeling languages are well-defined, their use by developers may not strictly follow textbook definitions, e.g. use case diagrams can be used in different ways by developers casting different stereotypes. The precise meaning of the semantics extracted from diagrams can thus be challenging due to ambiguities.

F2 - Constructing meaningful semantic information and forming knowledge. As software developers generally do little to organize software architecture information due to Issues 1, 2 and 3, ODAKS need to *provide automated support to extract the semantics* of the information to organize them into knowledge. First, ODAKS have to understand, in the sense of a machine without comprehension, the meaning of the contents. There are many approaches to interpret contents. For instance, semantic approaches have been applied in social media sense-making (Bontcheva and Rout, 2014). NLP and part-of-speech tagging have been used to semantically and semi-automatically analyze requirements, and requirements can be clustered by responsibilities (Casamayor et al., 2012).

Classifying Information. Classifying information and concepts in knowledge management can be applied to different kinds of knowledge. It can be used to represent procedural knowledge (know-how), causal knowledge (know-why), declarative knowledge (know-about) and so on (Alavi and Leidner, 2001). Information can be classified by relevant taxonomies to represent its type(s), as in LDAP in our example belonging to authentication. Some of these taxonomies already exist as industry standards, e.g. quality requirements in the ISO/IEC 25010:2011 standard (ISO/IEC25010:2011, 2011). A taxonomy can be purpose-built. NLP and clustering techniques can help to create taxonomies and dictionaries to approximate the semantics of design information. Kuhn et al. use latent semantic analysis to analyze source code, so that it can classify and group together similar design concepts (Kuhn et al., 2007). The TREx (ToeskaRationaleExtraction) approach has been proposed to recover, represent, and explore design rationale information from unstructured text documents using pattern-based information technology and ontology and other mechanisms (López et al., 2012). In recent research, architecture tactics and quality attributes were mined from Stack Overflow using NLP and machine learning techniques. The purpose was to identify architecture tactics and show what quality attributes are related to them. This work demonstrated that dictionary training and machine learning algorithms, such as Support Vector Machine (SVM) and Bagging can be used to classify architectural knowledge from discussion forums (Bi et al., 2021). Different NLP techniques for extraction and categorization have also been used (Bakar et al., 2015; Bi et al., 2018). Kleebaum et al. use a text classifier to identify rational in issues tracking and version control systems (Kleebaum et al., 2021). Prana et al. developed an automatic SVM-based classifier to analyze Github README files, classify and label the contents by who, when, what/why, how and other groups. They showed that a machine can be taught to identify keywords and thus help to ease information discovery (Prana et al., 2019). Classifiers such as the ones described here require supervised learning of many base documents. Information can come from geographically, technically and culturally diverse environments, machine learning for the purpose of classification could be language dependent, and a machine would need to adapt to these variations. The above and many other research works demonstrate that unorganized textual information can be automatically mined and structured for knowledge reuse.

Flexible Knowledge Representation. The users' knowledge needs vary and are hard to anticipate up-front (Issue 4). In order to serve varying and changing needs, information and knowledge must be structured and organized in a *flexible* and *adaptable* way to *support reasoning*. Many ontology implementations use RDF (Horrocks et al., 2003). Ontological relationships support reasoning between classes and concepts, e.g. authentication is about security. Ontologies and the semantic web support reasoning, they can define what knowledge to keep and what reasoning it can do for architects (de Graaf et al., 2014). Happel and Seedorf suggested to use ontology in AKS to support software engineering development including requirements engineering, component reuse, architecting, implementation, testing, and maintenance (Happel and Seedorf, 2006). They argued that "most of the formal foundations of ontologies have been in place for a long time" and such standardization can be a basis for wide-spread adoption. Additionally, they argued that "ontologies are well-suited to combine information from various sources and infer new facts based on this". Local ontologies capture knowledge of local projects, distributed ontologies can capture common knowledge shared and maintained by the software communities. Combining them allows project relevant knowledge to be enriched by industry and standard knowledge bases. In an empirical study, researchers mined design contexts and how contexts are used with design patterns (Wijerathna et al., 2021), the mined concepts and their inter-relationships are subsequently imported into an ontological structure where SPARQL queries can be used to search the knowledge. In a similar vein, Abebe and Tonella used NLP to parse a bug-report website to extract concepts and then insert them into an ontology (Abebe and Tonella, 2010).

Machine learning and NLP techniques can enhance the μK ontology's construction-based approach to construct an ontology (Lonsdale et al., 2010). ReVU provides a semi-automatic interpretation of UML models to extract design specification and properties (Konrad et al., 2006). ADeX incorporates a classification mechanism to understand the context of specifications and uses machine learning mechanism to group design decisions (Bhat et al., 2019) and AKB encapsulates different architecture concepts to relate context-specific and project-specific architecture profiles and patterns for reuse (Brandner et al., 2019). DBPedia annotator uses phrase spotting and disambiguation to identify phrases that can be linked to entities in DBpedia ontology (Bhat et al., 2017). This is a process to match text strings to concepts encapsulated

in an ontology. Mining of other types of design relationships such as component interactions (Yuan and Malek, 2016) and requirements traceability are common.

From these research works, we can see that three-steps (i.e. NLP for words extraction, concept construction by classification methods, and mapping to ontological system for inferencing) have been commonly used by several researchers. These techniques appear to be a promising approach for automated knowledge system construction.

Knowledge Structures and Relationships. Inquiries for knowledge within an ontology can be carried out with semantic wikis and SPARQL queries, but anything that is outside the ontological model cannot be answered. To overcome this shortcoming, the organization of knowledge must be flexible so that it can adapt, learn, and enlarge its model as new concepts and knowledge are added. Automated machine learning to build and enhance ontological models has been researched. Some ontology construction examples are the following: Mikrokosmos (μK) can learn new ontological concepts from a pre-defined ontology (Mahesh et al., 1996). The learner learns through cumulative analogy (Chklovski, 2003). μK provides a methodology to automate ontology reuse (Lonsdale et al., 2010). QuABasedBD can capture, machine-assistedly curate, learn, and classify database features from documents using SVM (Gorton et al., 2017).

Knowledge relationships can be established through mining, classifying, and/or conceptualizing of information (Bi et al., 2018). To do so, the first step is to identify knowledge relationships using *architecture viewpoints*. An architecture viewpoint describes one aspect of related software architecture concepts and cross-cutting concerns of a system. There are many different existing viewpoints to describe the many different aspects in a software system (Nuseibeh et al., 2003; Kruchten, 1995). An example is the software architecture viewpoint to represent the quality model of ISO/IEC 25010:2011 (ISO/IEC25010:2011, 2011). Implementation of viewpoints in ODAKS can help to establish links between design concepts and design artifacts (Mooney and Bunescu, 2005). Ontology can represent design properties (Scanniello et al., 2010) and assist architects to discover latent abstractions and rationale (López et al., 2012). Ontology can model software implementation by sub-systems, modules, classes and subroutines, system modifiability relationships (Lassing et al., 2001), decision and design rationale (van Heesch et al., 2012; Kruchten, 2004) and design patterns (Kampffmeyer and Zschaler, 2007). Past architecture decisions, design issues and solution options can also be captured for references and reuse in to-be architectures (Zimmermann and Miksovic, 2013).

General architecture knowledge in ontologies can be shared and distributed to different organizations. DBpedia has an ontology to capture knowledge about different database systems and use that knowledge to answer queries and provide recommendation (Bhat et al., 2017). Different viewpoints can be introduced to add new *meta-knowledge relationships,* such as the design rationale model, the quality attribute model, the DevOps process, the architect responsibility model, the design context checklist, and typical architect design concerns. Since architectural viewpoints are also knowledge concepts, they can be used to define meta-knowledge for the mining of information. Design contexts can be mined and organized to remind architects areas of concerns in design (Bedjeti et al., 2017; Wijerathna et al., 2021). Specialized viewpoints such as security ontology can also be incorporated into ODAKS. For instance, Li and Chen built an ontology that encapsulates security requirements which can be incorporated into ODAKS (Li and Chen, 2020). Knowledge engineers can implement viewpoints with an ontology. An example is a design rationale model as shown in Fig. 1. Decision-making and reasoning models have been empirically tested to demonstrate that architects who use them achieve better results (Tang et al., 2018; Poort and van Vliet, 2011; Bosse et al., 2005). These general knowledge models may be implemented by ontology and may be shared and maintained by the communities and distributed in ODAKS knowledge bases.

Many AKS lack the capabilities of a *flexible* and adaptable meta-knowledge representation to adapt to evolving needs of architects (Capilla et al., 2016). The lack of an adaptable meta-knowledge makes it difficult for AKS to evolve to incorporate new types of knowledge. This lack also makes it hard for AKS to scope searches with new knowledge. There can be different ways to tackle the ODAKS' adaptability issue. Meta-knowledge itself may be implemented as annotated RDF to form m-dimensional records (Schueler et al., 2007). ODAKS equipped with adaptable meta-knowledge can prompt inquirers to point out the areas or dimensions an architect would want to pursue, which could allow ODAKS to grow. A knowledge engineer (see Fig. 2) can also manually update an ODAKS to enhance its meta-knowledge. However, the challenge remains to provide an ontological structure based on the needs of the knowledge workers and to mine appropriate repositories to build up the knowledge such that ODAKS have a basis to perform meaningful inferences. It is the issue of knowing the "typical questions" (de Graaf et al., 2014) of the architects such that ODAKS can prepare itself to answer them.

Search-based software engineering (SBSE) has the vision of using machine-based searches to find solutions to software engineering problems (Harman et al., 2010). To do so, the nature of the problem must be well-defined and the range of solutions must be computable and comparable in order for such an approach to work. Unlike SBSE, ODAKS do not aim to present optimal solution(s). Instead of computing a range of solutions for decision makers, ODAKS organize and present relevant knowledge to assist decision makers to formulate and select solutions. Despite this difference, ODAKS may borrow some ideas from SBSE, such as clustering and taxonomizing of the problem and solution spaces (Harman et al., 2010) and introducing human interactions (Souza et al., 2018) to facilitate query response and optimization.

F3 - Interactive knowledge inquiry. Traditional information search engines typically search for information through some word matching or indexing mechanisms. There are two issues with this approach. First, such a search engine does not understand the semantics of a search to help it cut down its search space. Second, such a search engine cannot reason with an inquiry to ask for contextual clarifications. For ODAKS, we propose to use an intelligent and interactive inquiry interface and reasoning capabilities. This capability requires ODAKS to interact with architects to understand what and why an inquiry is about. An architect needs to provide more context when prompted, and an ODAKS can then make use of the additional information to home in its search and inferencing, as in the memory example described in Section 5.1 about F3.

Some systems use pre-determined question types: CNLP-AIDE Question Answering System is a question and answer user interface for use within a collaborative learning environment (Diekema et al., 2004). It has a language-to-logic component that creates a query representation and generates a question focus for a search. The authors analyzed questions asked by their subjects and found that there are seventeen different types of questions asked. In de Graaf et al. (2014), researchers discovered typical questions asked by software architects through interviews. From these questions, the researchers constructed an ontology to reason with what the architects typically ask. Both approaches have limitations as not all the questions, stakeholders, and situations are known at the time of question mining. These preset questions may not provide

for the needs of the architects. Therefore, ODAKS need to be able to understand and handle new questions.

One approach to find out what queries are important to the architects, is to survey architects and developers to see what kind of design questions they would ask, and then use that to structure architecture knowledge according to knowledge needs. This was done in de Graaf et al. (2014) where architects were interviewed and their typical design questions were worked into an ontological model. However, architects may not be able to foresee all their questions, and their design needs can change. ODAKS can ask for the user's feedbacks during runtime to determine if search results are relevant or not. Marking the relevancy of result sets can enhance ODAKS' knowledge base to provide probabilistic relevance feedbacks to guide searches and presentation in the future (Cambridge, 2009). This approach allows knowledge engineers to improve the structure of the ontology (see Fig. 2). It also allows ODAKS to dynamically adjust query results to suit architects' needs.

In order to service complex questions, a machine may interpret supportive design contexts, ask background questions to clarify misunderstandings, ambiguous key words, or assumptions (Sonntag, 2010). iHUB (Oddy, 1977) implemented advanced dialogue management and a semantic mediator to interpret a search query and translates it into an ontological query such as SPARQL. In addition to understanding a search query, a machine needs to interact with the inquirer to seek clarification. DiscOnto (Discourse Ontology) is an ontology to provide concepts for a dialogical interaction with the user and the interaction with the Semantic Web access sub-system (Oddy, 1977). Another approach is a machine to automatically identify and predict the goals of a user through understanding a user's past search history (Lee et al., 2005). Whilst these approaches may allow ODAKS to interact with architects to gather hints to better interpret the knowledge needs of architects, they do not help to alleviate human thinking issues described in Section 4. We suggest that ODAKS may help to alleviate human thinking issues partially by anticipating and adapting through assistive conversations. This endeavor is discussed in more detail in Section 6.2.

Information search and service tools using semantic web have been popular in the commercial space. Commercial tools such as Alexa, Siri, Cortana, and Google Assistant use natural language processing to interpret user questions. The keywords in the questions are searched and inferenced with the semantic web to find appropriate answers. The approach that ODAKS use can be similar to these commercial tools: the semantic relations represented by ODAKS' ontology is specific to software development and the domain is well-known to experienced architects and developers.. One of the potential technologies to implement interaction is to use bot architecture where bot service is responsible for the dialogue and bot logic is integrated with the ontological engine.

F4 - Knowledge selection and presentation. Design reasoning using ontologies has been implemented to trace design impacts, requirement to system behavior etc. (Abebe and Tonella, 2010; de Graaf et al., 2016; López et al., 2012). However, these systems do not know the contexts of a search and typically return result sets that contain all the information they can find, which may also contain irrelevant information. In our example, a search of LDAP would find "LDAP *realizes* Authentication" and "Authentication *provides* Security". However, there can be many threads or dimensions about LDAP. An architect may be concerned with LDAP implementation, LDAP features, LDAP upgrade, LDAP integration and so on. If ODAKS present the search results from all these dimensions, an architect might be overwhelmed. Presenting search results indiscriminately can thus be more of a hindrance when the results are not only text-based but include other artifacts such as design diagrams and source code (Sim et al., 1999). One potential solution is to provide a visual semantic wiki that represents architecture knowledge (Su et al., 2009), which provides better navigation and is potentially less demanding cognitively. Kleebaum et al. show how to extend issue tracking and version control systems with views on interconnected software design knowledge including rationale (Kleebaum et al., 2021).

Another consideration is the order in which ODAKS present their results. Zamir et al. suggest that it is difficult for users to search through a long list of documents and results (Zamir and Etzioni, 1999). They suggest that clustering of results can improve the coherence and browsability of the result set. Grouping of results can also reduce the cognitive load of a user having to scan through individual results. This idea has been implemented in different search clustering engines developed to group search results (Carpineto et al., 2009). One of them is Hierarchical Faceted Categories (HFC). HFC is a category system to provide meaningful labels to reflect the concepts relevant to a domain. This approach helps to reduce the mental load of the inquirer by presenting the search results by a category structure as well as search result alternatives (Hearst, 2006). Truede et al. used NLP techniques to extract development tasks from software documents, and when developers queried the system, TaskNavigator directed developers to the right documentation based on its understanding of the contexts of a query and predicts what developers want to find (Treude et al., 2014). These are some initial ideas that can be worked into ODAKS's design.

6.2. Techniques to assist human thinking

This section sketches how assistive conversations may help to deal with the human DM issues and processes and what potential methods and technologies can be used to carry out assistive conversations. ODAKS cannot detect human DM issues or process state. ODAKS cannot know if an architect is cognitively biased or cognitively overloaded. However, ODAKS can systematically offer a prioritized presentation of knowledge and an assistive conversation through probing, suggesting, and reflection to subtly provide opportunities for architects to engage in a systematic and informed DM process, which gives architects a better chance for good decisions. Such a human-centered approach to AKM is unique. First, we collect existing techniques on debiasing, nudging, and reflection from the area of DSS and design DM. Then, we discuss how this can be incorporated into ODAKS's Probe, Suggest, and Reflect conversations, and ultimately support an architect's DM.

6.2.1. Techniques for debiasing, nudging, and reflection

There are techniques which can debias the **confirmatory** behavior of executives (Chen and Lee, 2003) and financiers (Huang et al., 2012). For example, Huang et al. show that *counter-arguments* provided by a DSS can effectively reduce the effects caused by a confirmation bias. Other debiasing techniques focus on encouraging people to routinely ask themselves how they could disconfirm what they believe to be true—often referred to as *consider the opposite thinking* (Maule, 2010). However, research shows that DSS can sometimes aggravate the confirmation bias, because the use of DSS increases the confidence level of the individuals even when the accuracy of information is worse (Kottemann et al., 1994). Moreover, the use of DSS might create the illusion of control and, therefore, increase irrational confidence, hence confirmatory behavior (Davis and Kottemann, 1994).

To mitigate an **availability bias,** DSS tools have provided a few debiasing techniques (Chen and Lee, 2003). An example is *case memory* (Chen and Lee, 2003), which is meant to reduce the user's tendency to use most recent information by making past events and cases equally accessible. As such, it aids the user in recalling

the circumstances under which an event occurred or a decision was made, which may help to avoid inappropriate association between current circumstances and past events. Another debiasing technique is a *rationale model* of various knowledge elements such as design issues, design alternatives, and decision rationale (Mohan and Jain, 2008). In addition, *retrospective reflection* may alleviate the effects of an availability bias on decision making (da Cunha et al., 2016). There are also risks in DSS tools aggravating an availability bias. Easily retrieved knowledge can dominate the decision makers' understandings and judgments (Dube-Rioux and Russo, 1988).

To mitigate an **anchoring** bias, DSS research suggests various debiasing techniques (George et al., 2000) that include: (i) triggering users to think about *multiple future scenarios* focusing on the unpredictability of the future (Chen and Lee, 2003), (ii) providing *model advice* formulating and computing suggestions based on historic data (Jones et al., 2006), (iii) providing *reflective messages* to adjust possible anchors (George et al., 2000). The use of a DSS alone may aggravate the anchoring effect by readily retrieving possible anchors such as past decisions, experiences, or estimates (George et al., 2000).

In a recent literature survey, Jesse and Jannach suggested 83 **nudging mechanisms** in digital recommender systems collected into four different categories (Jesse and Jannach, 2021): (a) *Decision Information*: Change the decision information shown without changing the option (42 mechanisms); (b) *Decision Structure*: Rearrange the options (23 mechanisms); (c) *Decision Assistance*: Provide decision assistance such as reminders and reflections to architects (8 mechanisms); (d) *Social Decision Appeal*: Instigate empathy and provide social appeal to convince users (11 mechanisms).

Caraban et al. gathered 23 nudging mechanisms in the area of human–computer interaction organized into six different categories (Caraban et al., 2019). *(a) Facilitate* - encourages people to intuitively pursue a predefined set of actions based on their best interests and goals. It helps to diminish an individual's efforts through mechanisms such as *default options* and *opt-out policies. (b) Confront* - attempts to pause unwanted action by instilling doubt. It breaks mindless behavior and prompts reflective choices. This can be done through mechanisms such as *providing multiple viewpoints* and *reminding of consequences. (c) Deceive* - attempts to affect how alternatives are perceived, or how activities are experienced, with the goal of promoting particular outcomes using *deception mechanisms. (d) Social influence* - social influence nudges take advantage of people's desire to conform and comply with what is believed to be expected from them through mechanisms such as *invoking feelings of reciprocity* and *enabling social comparisons. (e) Fear* - evokes feelings of fear, loss, and uncertainty to make the user pursue an activity using mechanisms such as *make resources scarce. (f) Reinforce* - attempts to reinforce behaviors through increasing their presence in the individuals' thinking using mechanisms such as *just-in-time prompts* and *subtle feedbacks.*

Concerning the nudging in the context of the design DM, there is also work on *priming intuition and rationality*. Priming refers to somewhat mysterious workings of the automatic system of the brain which is a kind of nudging (Thaler and Sunstein, 2008). It was found that merely asking questions can influence how people behave. In an experiment, researchers asked people if they would vote the next day and that increased their voting by as much as 25% (Greenwald et al., 1987). In lab experiments, intuition and rationality have been primed successfully. Intuition has been primed mainly through *distracting* participants from consciously thinking about a problem using a complex task such as word puzzles followed by uninterrupted work on the target problem (Dijksterhuis and Meurs, 2006). Rationality can be primed through *instructions.* In one experiment, researchers subtly changed an instruction from "following requirement specification" to "following list of ideas", and with that, many more options had been generated (Mohanani et al., 2014). In another experiment, using reminder cards resulted in subjects considering more than one option for a solution (rather than fixating on the first idea that comes to mind), *creating explicit arguments* based on requirements, and *considering the various constraints and trade-offs* that must be made (Tang et al., 2018).

Aside from the lab environment, research on priming and training intuition in real-world practice has been limited. In the DSS literature, practices such as *electronic brainstorming,* i.e., software for generating and sharing ideas among a group, have been used to prime intuition (Gallupe et al., 1992). Training rationality in practice has been researched in software engineering. Emphasis has been placed on facilitating such rational DM through *capturing design rationale*, *conventional AKS* (Capilla et al., 2016); (Babar et al., 2009), *design reasoning techniques* (Tang and Vliet, 2009; Burge and Brown, 2000; Kruchten et al., 2006), and *reflective questions* (Razavian et al., 2016a; Tang et al., 2018).

All these nudging mechanisms focus on the presentation of the knowledge to influence the decision maker to move in a certain direction. Sometimes, e.g. in the case of deceive and fear (Caraban et al., 2019), nudging follows the interest of the system owner rather than the system user. Many of these mechanisms try to expand the mind of a decision maker.

Reflection is considered a deliberative rationality of choosing amongst choice (Van Manen, 1977). Reflection is a conversation, with others and with oneself, to challenge one's thinking process (Schön, 1983). Schön's reflection-in-action theory suggests professionals to have a reflective conversation through asking relevant questions over the course of formulating a solution. Talby et al. propose developers in Agile team to reflect on specific problems to make incremental improvements (Talby et al., 2006). Babb et al. propose to embed reflection and learning into agile teams (Babb et al., 2014). In Razavian et al. (2016a,b), we discuss how to help design DM through asking **reflective questions**. We proposed reflective questions based on rational and intuitive DM to be used by architects in software teams (Razavian et al., 2016b). For example, when an architect inquires about a solution, a reflection such as "What are the alternatives to ... ?" may trigger the architect to reflect on *solution options* and as such explore the solution space. However, reflections such as "What are the most important requirements?" may trigger the architect to switch to the *problem space*, and it may nudge the architect to think better. Reflections can be made into generic statements based on the core concepts behind problem-solution co-evolution. Questions can be used to prompt architects to consider design reasoning elements in the problem space and solution space.

6.2.2. Implementing the ODAKS assistive conversation

In the following, we discuss how to mitigate the human DM issues from Section 4 through an assistive conversation using the technology on debiasing, nudging, and reflection sketched in the previous section. We further introduce existing research works that have incorporated such technologies to achieve similar ideas envisioned in ODAKS's Probe, Suggest, and Reflect conversations.

ODAKS and cognitive biases. With suggestive conversation, ODAKS can trigger the decision maker to consider more appropriate information and knowledge, e.g. through showing past similar design cases, which is the *case memory* (Chen and Lee, 2003; Zhong, 2008) debiasing technique. ODAKS can use *facilitate nudging mechanisms* (Caraban et al., 2019) to *simplify the presentation of knowledge* that ODAKS compute for the architects. Alslaity et al. found that in recommender systems by phrasing

information and knowledge shorter and simpler (i.e. simplification nudge (Jesse and Jannach, 2021) the cognitive effort needed to understand them is significantly reduced (Alslaity and Tran, 2019). ODAKS can also use *suggest other viewpoints nudge* in the suggestive conversations. *Suggest other viewpoints nudge* have been successfully embedded in recommender systems to provide users with alternatives that might not have otherwise been considered (Forwood et al., 2015). This nudge can enable ODAKS' suggestive conversation to provide multiple design perspectives for architects to assess their design problems.

With reflective conversation, ODAKS can remind the users to consider other possibilities and embedding mechanisms (Mussweiler et al., 2000) to *consider the opposite thinking* (Bauer and Schedl, 2017). *Confront nudging mechanisms* can help to remind architects of the potential consequences of certain options or decisions through reflective conversation. Wang et al. embeds the confront nudging mechanism in a web-plugin that aims at mitigating impulsive disclosures on social media through reminding users' of the audience (Wang et al., 2014). Similarly, Harbach et al. redesigned the permissions dialogue of the Google Play Store to incorporate personalized scenarios that disclosed potential risks from app permissions (Harbach et al., 2014). If the app required access to one's storage, the system would randomly select images stored on the phone along with the message "this app can see and delete your photos". Researchers in fields such as human-technology interaction are beginning to understand how confront nudges can encourage more careful decision makers when they prompt a reflective choice (Caraban et al., 2019).

ODAKS and cognitive limitations. One of the purposes of ODAKS is to mitigate limited working memory, bounded rationality, and satisficing behaviors. Whilst ODAKS cannot extend human cognitive processing capabilities, they may be able to select the most relevant and available knowledge to present to architects and thereby reduce the size of potentially explosive solution space for consideration. Dorneich et al. identified cognitive overloads of users when using user interfaces (Dorneich et al., 2003), one of them is requiring short term memory when looking for information on a visual display. To address this issue, Taylor et al. studied user attention and concluded that it is "more efficient to eliminate [user] prediction, and define domain-general probability utility over the visual display, such that the display is transparent, non-interfering, and assistive" (Taylor et al., 2015). In other words, ODAKS need to work out what is important and be selective in the presentation of knowledge in the right contexts in suggestive conversations. *Facilitate nudging techniques* can include *visual arrangement of knowledge* to prioritize certain knowledge by *positioning* and providing *alternative viewpoints* to extend the cognitive and memory limitations of architects. Guidelines for digital nudges have been developed and experimented to influence online users. Researchers are beginning to understand how users tend to behave when digital nudges are presented to them (Schneider et al., 2018).

ODAKS and dual process decision-making. Similar to treating cognitive biases, ODAKS cannot know what cognitive processes occur in an architect's mind. However, based on an architect's dispositional style and situational factors, ODAKS can use facets of assistive conversations to prime (i.e. a kind of nudging) his or her cognitive process, being intuition, rationality, or both: priming can be considered as positioning of information to nudge the subjects (Caraban et al., 2019). While the effects of priming on DM have not yet been empirically verified, we imagine the following priming steps: (a) By probing an architect's dispositional style through the REI instrument (Dijksterhuis and Meurs, 2006), ODAKS may be able to detect whether an architect tends to rely on one or both, intuition and rationality cognitive processes; (b) With regard to context, ODAKS can collect decision situational factors such as the complexity of the problem, past experience of the architect in similar problems and solutions, or the stage of design in problem-solution co-evolution and with these contexts' conjecture whether priming is beneficial. Based on both, dispositional style and contextual factors, ODAKS can potentially prime intuition and rationality respectively through practices such as (i) distraction and brainstorming (intuition), and (ii) design rationale, design reasoning techniques, and reflective questions (rationality).

Researchers in DSS field have started to customize decision support systems based on the dispositional style of the decision maker (Engin and Vetschera, 2017). Engin et al. found that a poor fit of information representation to decision maker characteristics not only impedes the performance in the decision problem at hand, but also causes a depletion of cognitive resources. This has a negative impact on the solution of subsequent problems. Engin et al. suggest mechanisms to customize representation of information based on the decision maker's dispositional style (Engin and Vetschera, 2017). Schneider et al. propose to analyze users' past decisions or demographic data and use big-data to find their behavior patterns in real-time (Schneider, Weinmann, and Vom Brocke 2018). Through assistive conversations, ODAKS can collect the dialogue with architects and carry out data analysis to obtain interaction patterns. Research works on software engineers' personality, emotions, communication style, cognitive style or thinking style (Jia et al., 2016; Sánchez-Gordón and Colomo-Palacios, 2019; Mendes et al., 2019) can also help ODAKS' assistive conversations. Research in nudging mechanisms have proposed nudges that have a reflective strategy embedded into the design of a nudge based on the dual process decision-making theory. In these nudging mechanisms, the intuition nudge is theorized to correct failures of the intuitive cognitive processes, while the rationality nudge facilitates reflection in the decision makers (Banerjee and John, 2020).

ODAKS and problem-solution co-evolution. Although ODAKS cannot know the exact stage of problem-solution co-evolution of a design, based on the queries that architects pose and the dialogue with ODAKS, ODAKS may be able to approximate the stage of the problem-solution co-evolution. Such approximation can potentially be performed based on the analysis of similar questions being asked in the past through big-data analysis (Schneider et al., 2018) and sentiment analysis techniques (Medhat et al., 2014; Feldman, 2013). Sentiment techniques make use of lexicon-based approaches or machine-learning approaches to treat the opinions, sentiments, and subjectivity of text. In this case, some of the techniques such as lexicon-based methods (Taboada et al., 2011) and corpus-based semantic analysis using SVM (Moraes et al., 2013) could be used to classify and annotate questions, classify responses and selections by architects for ODAKS to learn about the intents of architects in terms of problem space or solution space explorations. With the support of these techniques, ODAKS can attach a semantic to the queries and infer if they are at the state of (i) "*what problems to solve?*", (ii) "*what solution to choose?*" and (iii) if they go back and forth between problem and solution space. ODAKS can offer a reflective dialogue with architects to remind them to check if the core design reasoning elements in the problem space and the solution space are sufficiently explored (see Fig. 1).

7. Summary and challenges

For the past two decades, the realization of AKS faces a number of issues. First, information and knowledge are scattered and disorganized. Second, it is challenging to collect the information because developers often do not document them systematically. Third, there is a lack of understanding of what information and

knowledge to capture. Finally, the application and the use of knowledge depends on the current contexts of the subject matter and the architect's intents, which may not match what an AKS contains. These issues have limited the implementation and adaptation of AKS. Additionally, human thinking issues complicate the use of knowledge from AKS. Humans are rationally bound and AKS can help extend this bound by providing extended memory and computational power. However, human architects are at the same time prone to cognitive biases, cognitive limitations, and other thinking issues. An AKS implementation will need to consider these issues such that it organizes the information and knowledge to work with architects instead of misleading them.

To cope with these issues holistically, we have suggested a research agenda with a vision of ODAKS with four major features. The research agenda also comprises suggested functionalities of the features, the research works that potentially can be used to realize the features, as well as the technical and human challenges of the implementation. In terms of the features, F1 is a set of crawlers that can collect different types of information, some of which are unstructured and written in natural language, others can be in architecture models such as UML. F2 extracts collected information and organizes it according to the meta-knowledge using semantic web or ontology implementations. F3 interacts with architects and F4 searches the knowledge base to provide knowledge. Between F3 and F4, ODAKS act as a companion to architects through Assistive Design Conversation to provide support in three ways: (a) probe architects to provide contexts and other relevant information to assist knowledge searches; (b) suggest areas and solutions that may be relevant, and present the knowledge accordingly; (c) provide reflections to nudge architects in their thinking. We synthesize the ODAKS features and high-level functionalities as our contributions to address the knowledge deficiency issues architects face today. The implementation of ODAKS features is challenging but viable. While this is only a vision paper and many research challenges are left open, we are convinced that architects will discover the benefits of ODAKS when they get to use first increments and these increments will be improved through their feedback.

ODAKS' goals are beyond the traditional AKS search capabilities. They should be companions to a human architectural decision maker, allowing creative and cyclical design DM. The ODAKS vision presented in this article is a *thought experiment* based on extensive literature research, analysis, and synthesis. Some of the features described in the ODAKS solution have been individually and separately implemented or experimented. These research works have provided a foundation for ODAKS' feasibility. The novelty of ODAKS vision is that it is the first time anyone has brought these AKS issues and solution options together.

In the process of developing the vision of ODAKS, we realize that many research challenges exist in this vision, including further development of intelligent technologies to handle meta-knowledge, human interactions as well as semantic analysis. There are still many unknowns in human DM and the application of assistive conversations to deal with human thinking issues. In the following, we discuss the challenges of ODAKS' research concerning the architects' DM, the knowledge collection, organization, query and presentation of the ODAKS, and concerning the interaction with the ODAKS:

Research on Architects' decision-making. First, we still need more research on architects' DM as such, **n**amely to what extent do cognitive biases and limitations, dispositional style, situational factors, and status of DM influence an architect's DM? Given the issues described in Section 3, we also need to understand better, how architects record and collect information. Both questions are fundamental not only to ODAKS' development but to software architecture design in general. We must study in detail **the influences of ODAKS on the DM of the architect.** We need to avoid aggravation of human DM issues created by ODAKS and understand how the conversation with a machine is different from the conversation with a human, and how the conversation with a machine changes the conversation in the team. Furthermore, we need to find out how to measure the effectiveness of ODAKS in supporting DM (including quality and relevance of presented knowledge, acceptance by the architect, better decisions, and project success).

Research on ODAKS' knowledge collection and organization (feature F1 and F2) to prepare the knowledge to answer inquiries by architects with features F3 and F4. Here we need to understand how ODAKS can cope with information recording and collection of architect interaction behavior (cf. the issues described in Section 3). The first challenge is the **complexity of mining information** from diverse sources, some with well-defined structures such as UML and some with written natural languages stored in different repositories, and differences in semantics. The second challenge is how to **structure the information efficiently and effectively** to serve the architects. Efficiency can come from automating ODAKS to "understand" information contents, relating, and storing them. Effectiveness can only be achieved, if the information is structured and stored meaningfully by useful meta-knowledge and viewpoints. This meta-knowledge will depend on contexts such as organizations, purposes and domains. Particularly interesting and challenging is how the ODAKS can **capture and organize the historical data** about an architect's DM. Unlike humans with limited memory, ODAKS can have access to the complete history of requirements, designs, implementations, conversations, and their evolutions.

Whilst viewpoints and software architecture models are generic enough to provide basic ontological models, specialized models for special needs due to design processes or industry domains, need to be handcrafted. This is a task for knowledge engineers who maintain ODAKS' ontological models and their evolution. We learn from experience that architects and developers cannot document trace links extensively, and we need to rely on machines to help us trace and assess change impacts. The challenge here is (a) to be able **to evolve an ontological model** that suits the needs of architects and the development organization over time; (b) to be able **to maintain the model** automatically or semi-automatically by a machine. Whilst there is plenty of research in data mining and machine learning as described in Section 6, automatic sense-making and auto-structuring of ontological models has not been studied in-depth as far as we know. One task is to figure out how newly crawled information that does not belong to any existing concepts, like new technologies and jargons, can be identified and incorporated within ODKAS. Another task is to figure out a process where these concepts can be incorporated into the ontological model through (un)supervised machine learning.

Building ontologies in ODAKS can be challenging, it may be larger than what one development organization can achieve. An organization may organize its own local project knowledge, but it needs to incorporate general architectural knowledge, which can be challenging for an organization to construct and maintain. The creation and maintenance of such ontological models may be conducted by research and standard organizations, and they can share and distribute the general architecture knowledge models in open-source ontological knowledge bases. Such a facility would provide flexibility and save efforts in building general architecture knowledge ontologies. A further challenge is to define the technology to support suitable interfaces for interoperability and ontological consistency.

Research on ODAKS' knowledge inquiry and presentation (F3 and F4) with assistive conversations. Here we need to research how and how well ODAKS can serve knowledge in the assistive conversation that matches the mental knowledge organization of the architect and the problems the architect wants to solve. The first challenge is to **analyze the historical data** about the architect to guess the dispositional style, situational factors, and status of DM of the architect when providing assistance and developing appropriate probing mechanisms. An additional challenge is the **semantic analysis** of data to identify the problems and solutions the architect is dealing with. This is particularly important for a reflective conversation. A further challenge is how ODAKS can **steer the DM behavior** of the architect based on useful design knowledge. This comprises further research on debiasing, nudging, and reflection mechanisms and their integration, for example how to use visual arrangement, position, and alternative viewpoints. In particular, future research is needed to effectively understand who (e.g., dispositional style), when (situational factors), and how to prime intuition and/or rationality in design DM.

Research on human and ODAKS interaction challenges. We need to research the challenges stemming from the companionship and the interaction of two counterparts: humans (architects) and ODAKS. As ODAKS require interacting with the architect in numerous ways, it is important that the interaction is effective with the human counterpart. It needs to exhibit characteristics typically associated with human behavior, namely understanding language and human decision-making (Cohen and Feigenbaum, 2014). The challenge, here on the one hand, is the **form and language in which the Assistive Design Conversation is presented**. On the other hand, the human counterpart of an ODAKS is also changing, becoming more conscious of the impact that such interactive systems have on his/her work and everyday life. As a consequence, we need to research how to enable a more **trustful and beneficial relationship** between architects and ODAKS, with a focus on supporting and empowering architects, ensuring at the same time their control and autonomy. This comprises further research on debiasing, nudging, and reflection mechanisms and their integration such that it is not too distracting or too forceful, leading to friction and less effectiveness. ODAKS need to achieve them without being intrusive, giving rise to the need for future research on the patterns of architects' behavior in the interaction with ODAKS.

The presented research agenda outlines the vision as well as the major future research challenges. In particular, we bring together the issues of management of architectural knowledge, human DM, and the intelligent support by information collection, knowledge inference and assistive design conversation. The research agenda as such has dual aim of understanding human DM and creating intelligent AKS. Research works are needed to, on one hand, understand and explain human DM and on the other hand to build better AKS that are necessarily intertwined and co-evolve. Many of the hardest problems of communication lie at the intersection of the social aspects of understanding and explaining and the technical aspects of designing and implementing. Only by targeting the fundamental interconnections among the social and technical aspects can we address our communication and collaboration challenges.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

Abebe, Surafel Lemma, Tonella, Paolo, 2010. Natural language parsing of program element names for concept extraction. In: 2010 IEEE 18th International Conference on Program Comprehension. IEEE, pp. 156–159.

Ahmad, Arshad, Feng, Chong, Khan, Muzammil, Khan, Asif, Ullah, Ayaz, Nazir, Shah, Tahir, Adnan, 2020. A systematic literature review on using machine learning algorithms for software requirements identification on stack overflow. Secur. Commun. Netw. 2020.

Al Omran, Fouad Nasser A., Treude, Christoph., 2017. Choosing an NLP library for analyzing software documentation: A systematic literature review and a series of experiments. In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories. MSR, pp. 187–197.

Alavi, Maryam, Leidner, Dorothy E., 2001. Knowledge management and knowledge management systems: Conceptual foundations and research issues. MIS Q. 107–136.

Alslaity, Alaa, Tran, Thomas, 2019. Towards persuasive recommender systems. In: 2019 IEEE 2nd International Conference on Information and Computer Technologies. ICICT, IEEE, pp. 143–148.

Antoniou, Grigoris, Van Harmelen, Frank, 2004. A Semantic Web Primer. MIT Press.

Arnott, David, 2006. Cognitive biases and decision support systems development: A design science approach. Inf. Syst. J. 16, 55–78.

Arnott, David, Gao, Shijia, 2019. Behavioral economics for decision support systems researchers. Decis. Support Syst. 122, 113063.

Arnott, David, Pervan, Graham, 2016. A critical analysis of decision support systems research revisited: The rise of design science. In: Enacting Research Methods in Information Systems. Springer.

Aurum, Aybüke, Wohlin, Claes, 2003. The fundamental nature of requirements engineering activities as a decision-making process. Inf. Softw. Technol. 45, 945–954.

Avgeriou, Paris, Kruchten, Philippe, Lago, Patricia, Grisham, Paul, Perry, Dewayne, 2007. Architectural knowledge and rationale – issues, trends, challenges. In: ACM SIGSOFT Software Engineering Notes. pp. 41–46.

Babar, Muhammad Ali, de Boer, Remco C., Dingsøyr, Torgeir, Farenhorst, Rik, 2007. Architectural knowledge management strategies: Approaches in research and industry. In: Second Workshop on SHAring and Reusing Architectural Knowledge - Architecture, Rationale, and Design Intent. SHARK/ADI 2007.

Babar, Muhammed Ali, Dingsøyr, T., Lago, P., van Vliet, Hans (Eds.), 2009. Software Architecture Knowledge Management: Theory and Practice. Springer-Verlag Berlin Heidelberg.

Babb, Jeffry, Hoda, Rashina, Norbjerg, Jacob, 2014. Embedding reflection and learning into agile software development. IEEE Software 31, 51–57.

Bakar, Noor Hasrina, Kasirun, Zarinah M., Salleh, Norsaremah, 2015. Feature extraction approaches from natural language requirements for reuse in software product lines: A systematic literature review. J. Syst. Softw. 106, 132–149.

Balasubramanian, Srinivasan, Chavet, Laurent, Qi, Runping, 2009. System, method, and service for collaborative focused crawling of documents on a network. Google Patents.

Banerjee, Sanchayan, John, Peter, 2020. Nudge plus: Incorporating reflection into behavioral public policy. Behav. Public Policy 1–16.

Bauer, Christine, Schedl, Markus, 2017. Introducing surprise and opposition by design in recommender systems. In: Adjunct Publication of the 25th Conference on User Modeling, Adaptation and Personalization. pp. 350–353.

Bedjeti, Adriatik, Lago, Patricia, Lewis, Grace, de Boer, Remco C., Hilliard, Rich, 2017. Modeling context with an architecture viewpoint. In: 1st International Conference on Software Architecture. IEEE, Gothenburg, Sweden.

Bhat, Manoj, Shumaiev, Klym, Biesdorf, Andreas, Hohenstein, Uwe, Hassel, Michael, Matthes, Florian, 2017. An ontology-based approach for software architecture recommendations.

Bhat, Manoj, Shumaiev, Klym, Hohenstein, Uwe, Biesdorf, Andreas, Matthes, Florian, 2020. The evolution of architectural decision making as a key focus area of software architecture research: A semi-systematic literature study. In: 2020 IEEE International Conference on Software Architecture. ICSA, IEEE, pp. 69–80.

Bhat, Manoj, Tinnes, Christof, Shumaiev, Klym, Biesdorf, Andreas, Hohenstein, Uwe, Matthes, Florian, 2019. ADeX: A tool for automatic curation of design decision knowledge for architectural decision recommendations. In: 2019 IEEE International Conference on Software Architecture Companion. ICSA-C, IEEE, pp. 158–161.

Bi, Tingting, Liang, Peng, Tang, Antony, Xia, Xin, 2021. Mining architecture tactics and quality attributes knowledge in stack overflow. J. Syst. Software 111005.

Bi, Tingting, Liang, Peng, Tang, Antony, Yang, Chen, 2018. A systematic mapping study on text analysis techniques in software architecture. J. Syst. Softw. 144, 533–558.
Bontcheva, Kalina, Rout, Dominic, 2014. Making sense of social media streams through semantics: A survey. Semantic Web 5, 373–403.
Bosse, Tibor, Jonker, Catholijn M., Treur, Jan, 2005. Reasoning by assumption: formalisation and analysis of human reasoning traces. In: International Work-Conference on the Interplay Between Natural and Artificial Computation. Springer, pp. 427–436.
Brandner, Klaus, Mayer, Benjamin, Weinreich, Rainer, 2019. Software architecture knowledge sharing with the architecture knowledge base (AKB). In: Proceedings of the 13th European Conference on Software Architecture-Volume 2. pp. 30–33.
Burge, Janet, Brown, David C., 2000. Reasoning with design rationale. In: Artificial Intelligence in Design'00. Springer.
Cambridge, U.P., 2009. Introduction to information retrieval.
Cao, Huanhuan, Hu, Derek Hao, Shen, Dou, Jiang, Daxin, Sun, Jian-Tao, Chen, Enhong, Yang, Qiang, 2009. Context-aware query classification. In: Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 3–10.
Capilla, Rafael, Jansen, Anton, Tang, Antony, Avgeriou, Paris, Ali Babar, Muhammad, 2016. 10 Years of software architecture knowledge management: Practice and future. J. Syst. Softw. 116, 191–205.
Capilla, Rafael, Jolak, Rodi, Chaudron, Michel R.V., Carrillo, Carlos, 2020. Design Decisions By Voice: The Next Step of Software Architecture Knowledge Management. Springer International Publishing, Cham, pp. 166–177.
Caraban, Ana, Karapanos, Evangelos, Gonçalves, Daniel, Campos, Pedro, 2019. 23 Ways to nudge: A review of technology-mediated nudging in human-computer interaction. In: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems. Association for Computing Machinery, Glasgow, Scotland Uk, Paper 503.
Carpineto, Claudio, Osiński, Stanislaw, Romano, Giovanni, Weiss, Dawid, 2009. A survey of web clustering engines. ACM Comput. Surv. 41, 1–38.
Casamayor, Agustin, Godoy, Daniela, Campo, Marcelo, 2012. Functional grouping of natural language requirements for assistance in architectural software design. Knowl.-Based Syst. 30, 78–86.
Charette, Robert N., 2005. Why software fails [software failure]. IEEE Spectr. 42, 42–49.
Chattopadhyay, Souti, Nelson, Nicholas, Nam, Thien, Calvert, McKenzie, Sarma, Anita, 2018. Context in programming: An investigation of how programmers create context. In: Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering. pp. 33–36.
Chen, Jim Q., Lee, Sang M., 2003. An exploratory cognitive DSS for strategic decision making. Decis. Support Syst. 36, 147–160.
Chklovski, Timothy Anatolievich, 2003. using Analogy to Acquire Commonsense Knowledge from Human Contributors. Massachusetts Institute of Technology.
Christiaans, Henri, Almendra, Rita Assoreira, 2010. Accessing decision-making in software design. Des. Stud. 31, 641–662.
Chuderski, Adam, Jastrzębski, Jan, 2018. Much ado about aha!: Insight problem solving is strongly related to working memory capacity and reasoning ability. J. Exp. Psychol. [Gen.] 147 (257).
Coelho, Jailton, Valente, Marco Tulio, 2017. Why modern open source projects fail. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. pp. 186–196.
Cohen, Paul R., Feigenbaum, Edward A., 2014. The Handbook of Artificial Intelligence, 3. Butterworth-Heinemann.
Conklin, J., Begeman, M., 1988. gIBIS: A hypertext tool for exploratory policy discussion. In: Proceedings of the 1988 ACM Conference on Computer-Supported Cooperative Work. pp. 140–152.
da Cunha, José Adson O.G., da Silva, Fabio Q.B., de Moura, Hermano P., Vasconcellos, Francisco J.S., 2016. Decision-making in software project management: A qualitative case study of a private organization. In: 2016 IEEE/ACM Cooperative and Human Aspects of Software Engineering. CHASE, IEEE, pp. 26–32.
Davis, Fred D., Kottemann, Jeffrey E., 1994. User perceptions of decision support effectiveness: Two production planning experiments. Decision Sci. 25, 57–76.
de Graaf, K.A., Liang, P., Tang, A., van Hage, W.R., van Vliet, H., 2014. An exploratory study on ontology engineering for software architecture documentation. Comput. Ind..
de Graaf, K.A., Liang, P., Tang, A., van Vliet, H., 2016. How organisation of architecture documentation affects architectural knowledge retrieval. Sci. Comput. Program. 121, 75–99.
Diekema, Anne R., Yilmazel, Ozgur, Chen, Jiangping, Harwell, Sarah, He, Lan, Liddy, Elizabeth D., 2004. Finding answers to complex questions.
Dijksterhuis, Ap, Meurs, Teun, 2006. Where creativity resides: The generative power of unconscious thought. Conscious. Cogn. 15, 135–146.
Dijksterhuis, Ap, Van Olden, Zeger, 2006. On the benefits of thinking unconsciously: Unconscious thought can increase post-choice satisfaction. J. Exp. Soc. Psychol. 42, 627–631.
Ding, W., Liang, P., Tang, A., van Vliet, H., Shahin, M., 2014. How do open source communities document software architecture: An exploratory survey. In: 9th International Conference on Engineering of Complex Computer Systems. ICECCS, Tianjin.
Dorneich, Michael C., Whitlow, Stephen D., Ververs, Patricia May, Rogers, William H., 2003. Mitigating cognitive bottlenecks via an augmented cognition adaptive system. In: SMC'03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme-System Security and Assurance (Cat. No. 03CH37483). IEEE, pp. 937–944.
Dorst, Kees, 2019. Co-evolution and emergence in design. Des. Stud. 65, 60–77.
Dube-Rioux, Laurette, Russo, J. Edward, 1988. An availability bias in professional judgment. J. Behav. Decis. Mak. 1, 223–237.
Dutoit, A., Peach, B., 2000. Rationale management in software engineering. In: Handbook of Software Engineering and Knowledge Engineering.
Dybå, Tore, Sjøberg, Dag I.K., Cruzes, Daniela S., 2012. What works for whom, where, when, and why? On the role of context in empirical software engineering. In: Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. pp. 19–28.
Eliëns, Ramon, Eling, Katrin, Gelper, Sarah, Langerak, Fred, 2018. Rational versus intuitive gatekeeping: Escalation of commitment in the front end of NPD. J. Prod. Innov. Manage. 35, 890–907.
Engin, Ayşegül, Vetschera, Rudolf, 2017. Information representation in decision making: The impact of cognitive style and depletion effects. Decis. Support Syst. 103, 94–103.
Epstein, Seymour, Pacini, Rosemary, Denes-Raj, Veronika, Heier, Harriet, 1996. Individual differences in intuitive–experiential and analytical–rational thinking styles. J. Personal. Soc. Psychol. 71 (390).
Feldman, Ronen, 2013. Techniques and applications for sentiment analysis. Commun. ACM 56, 82–89.
Forwood, Suzanna E., Ahern, Amy L., Marteau, Theresa M., Jebb, Susan A., 2015. Offering within-category food swaps to reduce energy density of food purchases: A study using an experimental online supermarket. Int. J. Behav. Nutr. Phys. Activity 12, 1–10.
Gallupe, R. Brent, Dennis, Alan R., Cooper, William H., Valacich, Joseph S., Bastianutti, Lana M., Nunamaker, Jr., Jay F., 1992. Electronic brainstorming and group size. Acad. Manag. J. 35, 350–369.
George, Joey F., Duffy, Kevin, Ahuja, Manju, 2000. Countering the anchoring and adjustment bias with decision support systems. Decis. Support Syst. 29, 195–206.
Gorton, Ian, Xu, Rouchen, Yang, Yiming, Liu, Hanxiao, Zheng, Guoqing, 2017. Experiments in curation: Towards machine-assisted construction of software architecture knowledge bases. In: 2017 IEEE International Conference on Software Architecture. ICSA, IEEE, pp. 79–88.
Graaf, Klaas Andries de, Liang, Peng, Tang, Antony, van Vliet, Hans, 2014. The impact of prior knowledge on searching in software documentation. In: Proceedings of the 2014 ACM Symposium on Document Engineering. ACM, Fort Collins, Colorado, USA, pp. 189–198.
Greenwald, Anthony G., Carnot, Catherine.G., Beach, Rebecca, Young, Barbara, 1987. Increasing voting behavior by asking people if they expect to vote. J. Appl. Psychol. 72 (315).
Happel, Hans-Jörg, Seedorf, Stefan, 2006. Applications of ontologies in software engineering. In: Proc. of Workshop on Sematic Web Enabled Software Engineering(SWESE) on the ISWC. Citeseer, pp. 5–9.
Harbach, Marian, Hettig, Markus, Weber, Susanne, Smith, Matthews, 2014. Using personal examples to improve risk communication for security & privacy decisions. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 2647–2656.
Harman, Mark, McMinn, Phil, De Souza, Jerffeson Teixeira, Yoo, Shin, 2010. 'Search based software engineering: Techniques, taxonomy, tutorial. In: Empirical Software Engineering and Verification. Springer.
Harper, K.E., Jiang, Zheng, 2015. Exploring software architecture context. In: Software Architecture (WICSA), 2015 12th Working IEEE/IFIP Conference on. pp. 123–126.
Hearst, Marti A., 2006. Clustering versus faceted categories for information exploration. Commun. ACM 49, 59–61.
Hodgkinson, Gerard P., Sadler-Smith, Eugene, 2018. The dynamics of intuition and analysis in managerial and organizational decision making. Acad. Manag. Perspect. 32, 473–492.
Horrocks, Ian, Patel-Schneider, Peter F., Van Harmelen, Frank, 2003. From SHIQ and RDF to OWL: The making of a web ontology language. Web Semant.: Sci., Serv. Agents World Wide Web 1, 7–26.
Hu, Jian, Wang, Gang, Lochovsky, Fred, Sun, Jian-tao, Chen, Zheng, 2009. Understanding user's query intent with wikipedia. In: Proceedings of the 18th International Conference on World Wide Web. pp. 471–480.
Huang, Hsieh-Hong, Hsu, Jack Shih-Chieh, Ku, Cheng-Yuan, 2012. Understanding the role of computer-mediated counter-argument in countering confirmation bias. Decis. Support Syst. 53, 438–447.
Hübner, Paul, Paech, Barbara, 2020. Interaction-based creation and maintenance of continuously usable trace links between requirements and source code. Empir. Softw. Eng. 25, 4350–4377.

ISO/IEC25010:2011, 2011. Systems and Software Engineering—Systems and Software Product Quality Re-Quirements and Evaluation (SQuaRE)—System and Software Quality Models. International Organization for Standardization, Geneva.

Jansen, Anton, Bosch, Jan, 2005. Software architecture as a set of architectural design decisions. In: Proceedings 5th IEEE/IFIP Working Conference on Software Architecture. pp. 109–120.

Jesse, Mathias, Jannach, Dietmar, 2021. Digital nudging with recommender systems: Survey and future directions. Comput. Hum. Behav. Rep. 3, 100052.

Jia, Jingdong, Zhang, Pengnan, Capretz, Luiz Fernandos, 2016. Environmental factors influencing individual decision-making behavior in software projects: A systematic literature review. In: Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering. pp. 86–92.

Jones, Donald R., Wheeler, Patrick, Appan, Radha, Saleem, Naveed, 2006. Understanding and attenuating decision bias in the use of model advice and other relevant information. Decis. Support Syst. 42, 1917–1930.

Kahneman, Daniel, 2011. Thinking, Fast and Slow. Penguin.

Kahneman, Daniel, Klein, Gary, 2009. Conditions for intuitive expertise: A failure to disagree. Am. Psychol. 64, 515.

Kahneman, Daniel, Tversky, Amos, Slovic, Paul, 1982. Judgment under Uncertainty Heuristics and Biases. Cambridge University Press, New York.

Kampffmeyer, Holger, Zschaler, Steffen, 2007. Finding the pattern you need: The design pattern intent ontology. In: International Conference on Model Driven Engineering Languages and Systems. Springer, pp. 211–225.

Keren, Gideon, 1990. Cognitive aids and debiasing methods: Can cognitive pills cure cognitive ills? Adv. Psychol. 68, 523–552.

Khatri, Naresh, Ng, H. Alvin, 2000. The role of intuition in strategic decision making. Hum. Relat. 53, 57–86.

Kleebaum, Anja, Paech, Barbara, Johanssen, Jan Ole, Bruegge, Bernd, 2021. Continuous rationale identification in issue tracking and version control systems. In: 9th Working Conference on Software Visualization. VISSOFT 2021.

Klein, Gary, 2008. Naturalistic decision making. Hum. Factors: J. Hum. Factors Ergon. Soc. 50, 456–460.

Klein, Gary, 2009. Streetlights and Shadows: Searching for the Keys to Adaptive Decision Making. The MIT Press.

Konrad, Sascha, Goldsby, Heather, Lopez, Karli, Cheng, Betty H.C., 2006. Visualizing requirements in UML models. In: 2006 First International Workshop on Requirements Engineering Visualization. REV'06-RE'06 Workshop, IEEE, p. 1.

Kottemann, Jeffrey E., Davis, Fred D., Remus, William E., 1994. Computer-assisted decision making: Performance, beliefs, and the illusion of control. Organ. Behav. Hum. Decision Process. 57, 26–37.

Kruchten, P., 1995. The 4+1 view model of architecture. IEEE Software 12, 42–50.

Kruchten, P., 2004. An ontology of architectural design decisions in software-intensive systems. In: 2nd Groningen Workshop on Software Variability Management.

Kruchten, P., 2008. What do software architects really do? J. Syst. Softw. 81, 2413–2416.

Kruchten, P., Lago, P., van Vliet, H., 2006. Building up and reasoning about architectural knowledge. In: Quality of Software Architecture. QoSA, Springer-Verlag, pp. 43–58.

Kuhn, Adrian, Ducasse, Stéphane, Gírba, Tudor, 2007. Semantic clustering: Identifying topics in source code. Inf. Softw. Technol. 49, 230–243.

Kunz, W., Rittel, H., 1970. Issues As Elements of Information Systems. Center for Planning and Development Research, University of California at Berkeley.

Lassing, N., Rijsenbrij, D., Van Vliet, H., 2001. Viewpoints on modifiability. Int. J. Software Eng. Knowl. Eng. 11, 453–478.

Lee, Uichin, Liu, Zhenyu, Cho, Junghoo, 2005. Automatic identification of user goals in web search. In: Proceedings of the 14th International Conference on World Wide Web. pp. 391–400.

Li, Tong, Chen, Zhishuai, 2020. An ontology-based learning approach for automatically classifying security requirements. J. Syst. Softw. 165, 110566.

Lonsdale, Deryle, Embley, David W., Ding, Yihong, Xu, Li, Hepp, Martin, 2010. Reusing ontologies and language components for ontology generation. Data Knowl. Eng. 69, 318–330.

López, Claudia, Codocedo, Víctor, Astudillo, Hernán, Cysneiros, Luiz Marcio, 2012. Bridging the gap between software architecture rationale formalisms and actual architecture documents: An ontology-driven approach. Sci. Comput. Programm. 77, 66–80.

Maclean, A., Young, R., Bellotti, V., Moran, T., 1996. Questions, options and criteria: Elements of design space analysis. In: Moran, T., Carroll, J. (Eds.), Design Rationale - Concepts, Techniques, and Use. Lawrence Erlbaum, New Jersey.

Maher, Mary Lou, Poon, Josiah, Boulanger, Sylvie, 1996. Formalising design exploration as co-evolution: A combined gene approach. In: Key Centre of Design Computing. (29), University of Sydney, Sydney.

Mahesh, Kavi, Helmreich, Stephen, Wilson, Lori, 1996. Ontology Development for Machine Translation: Ideology and Methodology. Citeseer.

Mangano, Nicolas, LaToza, Thomas D., Petre, Marian, van der Hoek, André, 2014. Supporting informal design with interactive whiteboards. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 331–340.

Maule, A. John, 2010. Can computers help overcome limitations in human decision making?. Intl. J. Hum.–Comput. Interaction 26, 108–119.

McCall, Raymond, 2010. Critical conversations: Feedback as a stimulus to a creativity in software design. Hum. Technol. 6, 11–37.

Medhat, Walaa, Hassan, Ahmed, Korashy, Hoda, 2014. Sentiment analysis algorithms and applications: A survey. Ain Shams Eng. J. 5, 1093–1113.

Mendes, Fabiana Freitas, Mendes, Emilia, Salleh, Norsaremah, 2019. The relationship between personality and decision-making: A systematic literature review. Inf. Softw. Technol. 111, 50–71.

Menezes, Júlio, Gusmão, Cristine, Moura, Hermano, 2019. Risk factors in software development projects: A systematic literature review. Softw. Qual. J. 27, 1149–1174.

Mohan, Kannan, Jain, Radhika, 2008. Using traceability to mitigate cognitive biases in software development. Commun. ACM 51, 110–114.

Mohanani, Rahul, Ralph, Paul, Shreeve, Ben, 2014. Requirements fixation. In: Proceedings of the 36th International Conference on Software Engineering. ACM, pp. 895–906.

Mohanani, Rahul, Salman, Iflaah, Turhan, Burak, Rodríguez, Pilar, Ralph, Paul, 2018. Cognitive biases in software engineering: A systematic mapping study. IEEE Trans. Softw. Eng..

Mooney, Raymond J., Bunescu, Razvan, 2005. Mining knowledge from text using information extraction. ACM SIGKDD Explor. Newsl. 7, 3–10.

Moraes, Rodrigo, Valiati, João Francisco, Neto, Wilson P. GaviãO, 2013. Document-level sentiment classification: An empirical comparison between SVM and ANN. Expert Syst. Appl. 40, 621–633.

Mussweiler, Thomas, Strack, Fritz, Pfeiffer, Tim, 2000. Overcoming the inevitable anchoring effect: Considering the opposite compensates for selective accessibility. Pers. Soc. Psychol. Bull. 26, 1142–1150.

Nelson, Katherine, 1998. Language in Cognitive Development: The Emergence of the Mediated Mind. (Cambridge University Press).

Nızam, Alı, 2022. Software project failure process definition. IEEE Access 10, 34428–34441.

Noy, N.F., McGuinness, D.L., 2001. Ontology Development 101: A Guide to Creating Your First Ontology. Stanford University.

Nuseibeh, Bashar, 2004. Crosscutting requirements. In: Proceedings of the 3rd International Conference on Aspect-Oriented Software Development. pp. 3–4.

Nuseibeh, Bashar, Kramer, Jeff, Finkelstein, Anthony, 2003. ViewPoints: Meaningful relationships are difficult!. In: The 25th International Conference on Software Engineering. pp. 676–681.

Oddy, R.N., 1977. Information retrieval through man-machine dislogue. J. Doc. 33, 1–14.

Pappas, Nikolaos, Katsimpras, Georgios, Stamatatos, Efstathios, 2012. An agent-based focused crawling framework for topic-and genre-related web document discovery. In: 2012 IEEE 24th International Conference on Tools with Artificial Intelligence. IEEE, pp. 508–515.

Parnas, D., Clements, P., 1985. A rational design process: How and why to fake it. IEEE Trans. Softw. Eng. 12, 251–257.

Perry, D.E., Wolf, A.L., 1992. Foundation for the study of software architecture. ACM SIGSOFT Software Eng. Not. 17, 40–52.

Poort, Eltjo R., van Vliet, Hans, 2011. Architecting as a risk- and cost management discipline. In: Proceedings of the Ninth IEEE/IFIP Working Conference on Software Architecture. pp. 2–11.

Prana, Gede Artha Azriadi, Treude, Christoph, Thung, Ferdian, Atapattu, Thushari, Lo, David, 2019. Categorizing the content of github readme files. Empir. Softw. Eng. 24, 1296–1327.

Pretorius, Carianne, Razavian, Maryam, Eling, Katrin, Langerak, Fred, 2018. Towards a dual processing perspective of software architecture decision making. In: 2018 IEEE International Conference on Software Architecture Companion. ICSA-C, IEEE, pp. 48–51.

Purao, Sandeep, Rossi, Matti, Bush, Ashley, 2002. Towards an understanding of the use of problem and design spaces during object-oriented system development. Inf. Organ. 12, 249–281.

Ralph, Paul, 2015. The sensemaking-coevolution-implementation theory of software design. Sci. Comput. Programm. 101, 21–41.

Razavian, Maryam, Tang, Antony, Capilla, Rafael, Lago, Patricia, 2016a. In two minds: How reflections influence software design thinking. J. Software: Evol. Process 28, 394–426.

Razavian, M., Tang, A., Capilla, R., Lago, P., 2016b. Reflective approach for software design decision making. In: 2016 Qualitative Reasoning About Software Architectures. QRASA, pp. 19–26.

Rizvi, Buturab, Bagheri, Ebrahim, Gasevic, Dragan, 2015. A systematic review of distributed agile software engineering. J. Software: Evol. Process 27, 723–762.

Robillard, Martin P., Marcus, Andrian, Treude, Christoph, Bavota, Gabriele, Chaparro, Oscar, Ernst, Neil, Gerosa, Marco Aurélio, Godfrey, Michael, Lanza, Michele, Linares-Vásquez, Mario, 2017. On-demand developer documentation. In: 2017 IEEE International Conference on Software Maintenance and Evolution. ICSME, IEEE, pp. 479–483.

Rose, Daniel E., Levinson, Danny, 2004. Understanding user goals in web search. In: Proceedings of the 13th International Conference on World Wide Web. pp. 13–19.
Sánchez-Gordón, Mary, Colomo-Palacios, Ricardo, 2019. Taking the emotional pulse of software engineering—A systematic literature review of empirical studies. Inf. Softw. Technol. 115, 23–43.
Scanniello, Giuseppe, D'Amico, Anna, D'Amico, Carmela, D'Amico, Teodora, 2010. Using the kleinberg algorithm and vector space model for software system clustering. In: 2010 IEEE 18th International Conference on Program Comprehension. IEEE, pp. 180–189.
Schneider, Christoph, Weinmann, Markus, Brocke, Jan Vom, 2018. Digital nudging: Guiding online user choices through interface design. Commun. ACM 61, 67–73.
Schön, Donald A., 1983. The Reflective Practitioner : How Professionals Think in Action. Basic Books, Nueva York, EUA.
Schön, Donald A., 1984. Problems, frames and perspectives on designing. Des. Stud. 5, 132–136.
Schön, Donald A., 1992. Designing as reflective conversation with the materials of a design situation. Knowl.-Based Syst. 5, 3–14.
Schueler, Bernhard, Sizov, Sergej, Staab, Steffen, 2007. Management of meta knowledge for rdf repositories. In: International Conference on Semantic Computing. ICSC 2007, IEEE, pp. 543–550.
Senker, Jacqueline, 1995. Tacit knowledge and models of innovation. Ind. Corp. Change 4, 425–447.
Sim, S. Elliott, Clarke, Charles L.A., Holt, Richard C., Cox, Anthony M., 1999. Browsing and searching software architectures. In: Proceedings IEEE International Conference on Software Maintenance-1999 (ICSM'99).'software Maintenance for Business Change'(Cat. No. 99CB36360). IEEE, pp. 381–390.
Simon, Herbert A., 1973. The structure of ill structured problems. Artificial Intelligence 4, 181–201.
Simon, Herbert, 1982. Models of Bounded Rationality. MIT Press Cambridge, Mass.
Simon, Herbert A., 1996. The Sciences of the Artificial. The MIT Press.
Simon, Herbert, Newell, Allen, 1972. Human Problem Solving: The State of the Theory in 1970. Carnegie-Mellon University.
Sonntag, Daniel, 2010. Ontologies and Adaptivity in Dialogue for Question Answering. IOS Press.
Souza, Jerffeson, Araújo, Allysson Allex, Saraiva, Raphael, Soares, Pamella, Maia, Camila, 2018. A Preliminary Systematic Mapping Study of Human Competitiveness of SBSE. Springer International Publishing, Cham, pp. 131–146.
Stacy, Webb, MacMillan, Jean, 1995. Cognitive bias in software engineering. Commun. ACM 38, 57–63.
Su, Moon Ting, Hirsch, Christian, Hosking, John, 2009. Kaitorobase: Visual exploration of software architecture documents. In: 2009 IEEE/ACM International Conference on Automated Software Engineering. IEEE, pp. 657–659.
Taboada, Maite, Brooke, Julian, Tofiloski, Milan, Voll, Kimberly, Stede, Manfred, 2011. Lexicon-based methods for sentiment analysis. Comput. Linguist. 37, 267–307.
Talby, David, Hazzan, Orit, Dubinsky, Yael, Keren, Arie, 2006. Reflections on reflection in agile software development. In: AGILE 2006. AGILE'06, IEEE, pp. 11–112.
Tamburri, Damian A., Palomba, Fabio, Kazman, Rick, 2020. Success and failure in software engineering: A followup systematic literature review. IEEE Trans. Eng. Manage. 68, 599–611.
Tang, Antony, 2011. Software designers, are you biased? In: Proceeding of the 6th international workshop on SHAring and Reusing architectural Knowledge. ACM, Waikiki, Honolulu, HI, USA, pp. 1–8.
Tang, A., Avgeriou, P., Jansen, A., Capilla, R., Babar, M. Ali, 2010. A comparative study of architecture knowledge management tools. J. Syst. Software 352–370. http://dx.doi.org/10.1016/j.jss.2009.08.032.
Tang, Antony, Babar, Muhammad Ali, Gorton, Ian, Han, Jun., 2005. A survey of the use and documentation of architecture design rationale. In: Software Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conference on. IEEE, pp. 89–98.
Tang, Antony, Bex, Floris, Schriek, Courtney, van der Werf, Jan Martijn E.M., 2018. Improving software design reasoning–A reminder card approach. J. Syst. Softw. 144, 22–40.
Tang, Antony, Lau, Man F., 2014. Software architecture review by association. J. Syst. Softw. 88, 87–101.
Tang, Antony, Liang, Peng, Clerc, Viktor, van Vliet, Hans, 2011a. Supporting co-evolving architectural requirements and design through traceability and reasoning. In: Avgeriou, Paris, Lago, Patricia, Grundy, John, Mistrik, Ivan (Eds.), Relating Software Requiremens and Software Architecture.
Tang, Antony, Liang, Peng, Van Vliet, Hans, 2011b. Software architecture documentation: The road ahead. In: 2011 Ninth Working IEEE/IFIP Conference on Software Architecture. IEEE, pp. 252–255.
Tang, Antony, Vliet, Hans, 2009. Software architecture design reasoning. In: Babar, Muhammad Ali, Dingsøyr, Torgeir, Lago, Patricia, Vliet, Hans (Eds.), Software Architecture Knowledge Management. Springer Berlin Heidelberg.
Tang, A., van Vliet, H., 2012. Design strategy and software design effectiveness. IEEE Softw. 29, 51–55.
Tang, Antony, van Vliet, Hans, 2015. Software designers satisfice. In: Weyns, Danny, Mirandola, Raffaela, Crnkovic, Ivica (Eds.), Software Architecture. Springer International Publishing.
Taylor, P., He, Ze, Bilgrien, Noah, Siegelmann, Hava T., 2015. Human strategies for multitasking, search, and control improved via real-time memory aid for gaze location. Front. ICT 2, 15.
Thaler, Richard, Sunstein, Cass, 2008. Nudge. Penguin, New York, USA.
The Standish Group, 2014. Chaos report on software projects. In: Project Smart.
Treude, Christoph, Robillard, Martin P., Dagenais, Barthélémy, 2014. Extracting development tasks to navigate software documentation. IEEE Trans. Softw. Eng. 41, 565–581.
Tversky, A., Kahneman, D., 2000. Judgement under uncertainty: Heuristics and biases. In: Connolly, T., Arkes, H.R., Hammond, K.R. (Eds.), Judgement and Decision Making: An Interdisiplinary Reader. Cambridge University Press.
van Heesch, Uwe, Avgeriou, Paris, Hilliard, Rich, 2012. Forces on architecture decisions-a viewpoint. In: Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on. IEEE, pp. 101–110.
Van Manen, Max, 1977. Linking ways of knowing with ways of being practical. Curric. Inq. 6, 205–228.
Van Vliet, Hans, 2008. Software architecture knowledge management. In: 19th Australian Conference on Software Engineering. ASWEC 2008, IEEE, pp. 24–31.
van Vliet, Hans, Tang, Antony, 2016. Decision making in software architecture. J. Syst. Softw. 117, 638–644.
Wang, Yang, Leon, Pedro Giovanni, Acquisti, Alessandro, Cranor, Lorrie Faith, Forget, Alain, Sadeh, Norman, 2014. A field trial of privacy nudges for facebook. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 2367–2276.
Weinmann, Markus, Schneider, Christoph, Vom Brocke, Jan, 2016. Digital nudging. Bus. Inf. Syst. Eng. 58 6 (Dec. 2016), 433–436.
Weinreich, Rainer, Groher, Iris, Miesbauer, Cornelia, 2015. An expert survey on kinds, influence factors and documentation of design decisions in practice. Fut. Gener. Comput Syst. 47, 145–160.
Wijerathna, Laksri, Aleti, Aldeida, Bi, Tingting, Tang, Antony, 2021. Mining and relating design contexts and design patterns from stack overflow. Empir. Softw. Eng. 27, 8.
Xu, Zhuoming, Ni, Yuyan, He, Wenjie, Lin, Lili, Yan, Qin, 2012. Automatic extraction of OWL ontologies from UML class diagrams: A semantics-preserving approach. World Wide Web 15, 517–545.
Yuan, Eric, Malek, Sam, 2016. Mining software component interactions to detect security threats at the architectural level. In: 2016 13th Working IEEE/IFIP Conference on Software Architecture. WICSA, IEEE, pp. 211–220.
Zamir, Oren, Etzioni, Oren., 1999. Grouper: A dynamic clustering interface to web search results. Comput. Netw. 31, 1361–1374.
Zhong, Yinghong, 2008. The framework of total decision support based on knowledge management. In: 2008 International Seminar on Future Information Technology and Management Engineering. IEEE, pp. 516–520.
Zimmermann, Olaf, Miksovic, Christoph, 2013. Decisions required vs. decisions made: Connecting enterprise architects and solution architects via guidance models. In: Aligning Enterprise, System, and Software Architectures. IGI Global.
Zimmermann, Olaf, Wegmann, Lukas, Koziolek, Heiko, Goldschmidt, Thomas, 2015. Architectural decision guidance across projects-problem space modeling, decision backlog management and cloud computing knowledge. In: Software Architecture (WICSA), 2015 12th Working IEEE/IFIP Conference on. IEEE, pp. 85–94.