It's a Complete Haystack: Understanding Dependency Management Needs in Computer-Aided Design

KATHY CHENG, University of Toronto, Canada
ALISON OLECHOWSKI, University of Toronto, Canada
SHURUI ZHOU, University of Toronto, Canada

In today's landscape, hardware development teams face increasing demands for better quality products, greater innovation, and shorter manufacturing lead times. Despite the need for more efficient and effective processes, hardware designers continue to struggle with a *lack of awareness* of design changes and other collaborators' actions, a persistent issue in decades of CSCW research. One significant and unaddressed challenge is understanding and managing dependencies between 3D CAD (computer-aided design) models, especially when products can contain thousands of interconnected components. In this two-phase formative study, we explore designers' pain points of CAD dependency management through a thematic analysis of 100 online forum discussions and semi-structured interviews with 10 designers. We identify nine key challenges related to the traceability, navigation, and consistency of CAD dependencies, that harm the effective coordination of hardware development teams. To address these challenges, we propose design goals and necessary features to enhance hardware designers' awareness and management of dependencies, ultimately with the goal of improving collaborative workflows.

CCS Concepts: • **Human-centered computing** → **Empirical studies in collaborative and social computing**; • **Applied computing** → **Computer-aided design**.

Additional Key Words and Phrases: awareness, collaboration, hardware development, distributed work

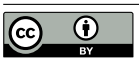
ACM Reference Format:
Kathy Cheng, Alison Olechowski, and Shurui Zhou. 2025. It's a Complete Haystack: Understanding Dependency Management Needs in Computer-Aided Design. *Proc. ACM Hum.-Comput. Interact.* 9, 7, Article CSCW436 (November 2025), 32 pages. https://doi.org/10.1145/3757617

1 Introduction

The technologies we rely on in our daily lives are constantly evolving, but the fundamental need for physical products remains unchanged. The process of designing physical products (i.e., hardware development) requires collaboration among multiple engineers and teams. In today's landscape, however, the hardware development process is increasingly challenging, due to the growing complexity of products and the distributed nature of hardware development work [22, 90].

A significant challenge in collaborative hardware development is a *lack of awareness*, a persistent issue highlighted in decades of CSCW research [87, 95] across various domains, like software development [136], data science [96], and collaborative writing [83]. Specifically in hardware development, it is challenging for designers to synchronize design activities and maintain an awareness of what others are working on, especially because products can be immensely complex. For instance, the Boeing 787 Dreamliner included 300,000+ CAD (computer-aided design) models [12] that must cohesively integrate for the final airplane to function properly. Even student design projects can

Authors' Contact Information: Kathy Cheng, University of Toronto, Toronto, Canada, kathy.cheng@mail.utoronto.ca; Alison Olechowski, University of Toronto, Toronto, Canada; Shurui Zhou, University of Toronto, Toronto, Canada.



ACM 2573-0142/2025/11-ARTCSCW436
https://doi.org/10.1145/3757617

involve 1,000+ models parametrically linked by 40,000+ dependency relationships [53]. Without dependency management tools, it is impossible to navigate these intricate dependency relationships [44], locate design errors [70, 105], or prevent unintended impacts on others' work [22]. Researchers, therefore, strongly advocate for the development of dependency awareness tools that can passively track these complex relationships [19, 24, 79], thereby facilitating collaboration.

Managing dependencies is not limited to hardware development; modern software development also requires careful coordination of activities, people, and artifacts. Software systems, like code, libraries, or frameworks, are built from existing systems, creating *technical dependencies* [37, 56]. Software developers must be aware of these dependencies to understand how their work will impact each other and to ensure that the project progresses in the right direction. Researchers have developed tools to analyze and visualize technical dependencies to decrease developers' cognitive load and facilitate communication among development teams [37].

While dependency management has been explored in software development, the unique nature of hardware development makes dependency management especially challenging, due to the geometric data, the granularity of CAD dependencies, and the lack of maturity in the tools. First, hardware designers work with 3D CAD models that represent intricate geometric and topological data [48], and the models' dependencies are inherently linked to their physical architecture. Unlike software, where dependencies are typically expressed in text-based code and can be traced through direct imports or function calls, CAD dependencies are often implicit [17], which are difficult to trace or search for [70]. Second, dependencies in CAD can exist at multiple levels of granularity, such as between 2D sketches, or assemblies (composed of multiple interconnected 3D models), as well as between levels (e.g., a top-level assembly can reference a low-level sketch). Finally, collaborative infrastructure for CAD (e.g., awareness tools, version control) is generally less mature than in software development [22]. Therefore, we can draw inspiration from the software development literature; however, we must recognize that the unique characteristics of hardware design necessitate tailored approaches to effectively manage CAD dependencies.

Dependency management tools tailored to CAD artifacts are crucial for supporting efficient collaboration in hardware development teams. While existing literature has developed visualizations or models of CAD dependencies [99, 111], the user requirements for such tools remain poorly understood due to the absence of a systematic exploration with real CAD practitioners. Therefore, we draw from similar research targeting awareness challenges in collaborative work [1, 34, 65, 78, 103, 139], conducting formative studies to better understand designers' needs regarding dependency management. Our work is guided by the research question: **What challenges do design engineers face with managing technical dependencies in CAD projects?** Gaining empirical insights into these challenges will enable us to devise appropriate strategies for addressing CAD dependency management, including the development of targeted tools.

Our approach to answering this question is a two-part formative study, wherein we gathered insights from actual CAD users through online forum discussions and semi-structured interviews. In the first phase, we mined 100 popular forum posts to obtain an overview of current practices and challenges. In the subsequent phase, we conducted interviews with 10 professional mechanical design engineers from a large technology organization that has established advanced dependency management practices, to identify rich scenarios and examples of persistent pain points. By conducting a thematic analysis of both data sources, we engage with a diverse array of CAD users through forum discussions, while simultaneously gaining an in-depth understanding of challenges from the interviews. This comprehensive approach enables us to thoroughly assess the existing gaps in CAD dependency management, as well as propose design implications for CAD platforms. Our contributions to CSCW are summarized as follows:

(1) A systematic identification of nine dependency management challenges in CAD, integrating insights from online forums and professional mechanical design engineers.
(2) A set of design goals, features, and initial tool concepts aimed at enhancing awareness and coordination in CAD work, focusing specifically on improving the traceability and navigation of technical dependencies.
(3) A discussion of the broader implications of improved technical dependency management, including its potential to enhance coordination of people and planning of design activities.

2 Background & Related Work

In this section, we first provide a background on awareness in CSCW to motivate our study. We then explore the specific awareness needs in CAD and the role of dependency management in addressing these needs. Finally, we review the relevant literature on CAD dependency management, as well as in the related field of software development, where dependency management has been studied more thoroughly.

2.1 Awareness in CSCW

Awareness is defined as the *"understanding of the activities of others, which provides a context for your own activity"* [41], and is a fundamental concept to the field of CSCW [57, 61, 85, 122]. In their seminal work, Gutwin and Greenberg [59] developed a conceptual framework for *workspace awareness*, which refers to the up-to-the-moment understanding of another person's interactions with a shared workspace. Simply put, awareness is understanding who is in the workspace, where they are working, and what actions they are taking [58], which involves both displaying one's actions and monitoring others' actions [66]. More recently, Tenenberg et al. introduced *we-awareness*, which extends beyond simply displaying one's actions and monitoring those of others [135]. We-awareness refers to the socially recursive knowledge that each participant of a collaborative effort has of the other (i.e., the action of partner A is perceivable by partner B, partner B's perception is perceivable by partner A, and so on) [135] such that all collaborators are mutually aware of each other's awareness [55]. This recursive nature of we-awareness is crucial for collaborative work, especially in distributed teams, because it enables individuals to anticipate conflicts, coordinate activities, and build common ground [28].

There are various levels of collaborative work (*coordinated*, *cooperative*, *co-constructive*) [7], but at the most basic level, collaboration requires coordination. As defined by Malone and Crowston, coordination is *"the act of managing dependencies between activities performed to achieve a goal"* [92]. An example of a dependency is a *transfer* dependency, which occurs when one activity produces something (e.g., a product, information) that another activity needs; this requires transferring from the "producer" activity to the "consumer" activity [93]. For coordination to occur, these dependencies must be managed, and the actors must have awareness of these dependencies. Without adequate awareness, teams are more likely to encounter misalignments, redundancies, or conflicts in their work [61]. Thus, awareness and coordination are deeply interconnected: awareness provides the contextual information necessary for managing dependencies, while coordination mechanisms [123] aim to structure and handle those dependencies efficiently [42, 121].

Maintaining awareness, however, is challenging due to the dispersed nature of knowledge in collaborative work [60] – what Hutchins refers to as *distributed cognition* [69]. The information needed for coordination is distributed across team members and their workspace. This dispersion imposes a cognitive load on individuals to track and process all relevant information [68]. Researchers have developed various mechanisms (e.g., notifications [62, 88], shared dashboards [136], real-time collaborative tools [143]) to reduce the user's cognitive load by providing visibility of other people's actions and changes to shared artifacts [33]. For instance, change awareness tools

track asynchronous modifications to collaborative documents, summarizing what was changed, who made the change, where it occurred, how it differs from previous versions, and why it was made [134]. Importantly, awareness mechanisms are not about delivering all information to users; rather, they are about disseminating contextually relevant information based on the specific work needs [124]. Thus, designing effective awareness tools requires answering key questions like: *What information should be provided? When should it be provided? To whom and in what form?*

Overall, the CSCW literature underscores the critical role of awareness and coordination in cooperative work. Our research specifically focuses on enhancing awareness in the field of hardware development by helping designers manage dependencies in CAD. By identifying CAD dependency management challenges, we highlight specific areas to improve designers' workspace awareness, ultimately fostering more effective coordination and collaboration for hardware development teams.

2.2 Awareness Needs in Cooperative CAD Work

Awareness is key in all cooperative work domains, but it is especially difficult to maintain in CAD work, due to the nature of developing hardware products. For instance, CAD artifacts (e.g., 3D models) represent complex geometric and topological data [48], making it challenging to use and develop traditional coordination tools, like version control [21]. As such, CAD designers often struggle to identify the changes to a shared model, identify conflicts between models, and integrate contributions among collaborators [22].

In cooperative CAD work, multiple designers or teams work together to design a physical product. Typically, each designer or team is responsible for different parts or subsystems, which are later integrated to complete the overall product. To illustrate this process, imagine the following scenario: Designer A will model the wheels of a car, and Designer B will model the axle concurrently [5]. For their designs to function cohesively, Designers A and B must communicate about design considerations, e.g., the wheel dimensions, materials, heat transfer, sequence of design decisions [107]. Additionally, they must coordinate with Designer C, who may be designing the car's frame, and agree on the interfaces between components [127]. The CAD models act as intermediary objects that enable coordination between designers [106, 137]. Therefore, it is crucial for designers to be aware of their collaborators' changes to shared CAD models [22].

To ensure interfacing parts work together, designers use *parametric CAD* [4], which relies on parameters to define and constrain a model's geometry. Revisiting our car design example, Designer B may define the axle's position relative to the center of the wheels. If Designer A changes the wheel diameter, the axle will automatically update – preserving fit and structural integrity without requiring manual rework [129]. While this approach works in theory, managing dependencies in complex products, comprising thousands of interacting parts, can quickly become a colossal task. Designers not only need awareness of the dependencies between CAD models to avoid unintentionally impacting another designer's model [22], but they also need to modify the dependencies accurately as the design changes – for example, deleting dependencies when they are no longer needed. Maintaining accurate and consistent parametric relationships is essential to prevent costly manual rework and integration issues [15].

In summary, awareness in the CAD context entails not only understanding what others are working on but also recognizing how design changes affect the overall product. Due to the parametric aspect of CAD models, the dependencies within a product can become highly complex and, if not managed properly, can result in significant issues such as integration failures, misaligned components, and rework. Therefore, our work aims to shed light on the user needs for managing CAD dependencies, ultimately aiming to help improve awareness and coordination in CAD projects.

2.3 Dependency Management

In this section, we review related work, beginning with the study of dependency management in software development, a domain that has long recognized and supported these challenges, providing frameworks and tooling support. We then highlight how dependency management in CAD presents unique and significant challenges beyond those encountered in software development.

2.3.1 In Software Development. Dependency management has been studied widely in the software development field [35, 36]. In cooperative software development, dependencies arise among activities, artifacts, and different parts of the same artifact (like program dependencies [109]) [45]. While there are many types of dependencies (e.g., knowledge, process, resource) [130, 131], software development projects generally have: *technical dependencies*, which exist between artifacts (e.g., components of a software system [132]) [38], and *work dependencies* [18], among developers. Often, technical dependencies imply work dependencies [56] – creating *socio-technical dependencies* [118] – which requires developers to coordinate and communicate to maintain a shared understanding of the dependency [56].

Dependencies are necessary since modern software is built on the foundation of existing code, libraries, and frameworks [71]. However, dependencies can introduce challenges, such as breaking changes [13, 72], incompatibility with other dependencies [40], and abandonment, where a package is used but no longer maintained [102]. Developers must keep updated with the latest versions of libraries that their code depends on to maintain *dependency freshness* [32], but it can be laborious to constantly track new updates [81], prompting the development of tools for automatic detection and updating [64]. When dependencies are not properly maintained, common errors can occur, such as missing dependencies (where necessary links are absent) and redundant dependencies (where a dependency exists but serves no purpose) [46, 126]. Tools to automatically identify these errors have also been of interest in related work [46, 104, 114], and are widely used in open-source communities to facilitate the maintenance of dependencies [9].

Due to the complexity of managing dependencies, software development researchers have investigated various strategies, finding "formal" approaches, like impact analysis tools to identify entities that will be affected by a change [84], and also "informal" approaches, like extensive email communication to broadcast the update [36]. Visualization tools have been built to help developers better understand the technical dependencies at different granularity levels, between files [54], features [25], and code blocks [14]. For instance, De Souza et al. created network graph visualizations to illustrate dependencies among developers for various scenarios, such as identifying developers working on similar tasks [37]. Another tool, Palantír, monitors artifacts in different developers' workspaces and visually exchanges information about how dependent artifacts are being modified, aiming to avoid conflicting changes [119]. Gori et al. developed FileWeaver, a system that automatically detects dependencies among files, tracks their history, and lets users view the downstream and upstream dependencies [54]. The aim of all of these tools is to improve software developers' awareness of technical dependencies.

The software development literature has extensively explored various aspects of dependency management, such as types of dependencies, dependency-related errors, and awareness tools. However, managing dependencies in CAD is inherently more complex due to the nature of 3D models, which are tightly coupled with the physical architecture of the product, unlike text-based code [48]. Although software engineering provides mature frameworks and tooling for dependency management, comparable solutions remain underdeveloped in the CAD domain. Our work aims to fill this gap by providing a deeper understanding of dependency challenges in CAD, which is essential for developing effective management strategies.

2.3.2 In CAD. As introduced in Section 2.2, the key idea of parametric CAD is that parameters are used to define all geometry, whether the shape of a single part, or the positions of parts relative to each other [52]. When designers are not careful about how they define these parameters (e.g., the direction of the dependency), they can introduce *architecture debt* [115], harming CAD model maintainability and reusability [15]. This becomes especially challenging in complex product development, where a single product (e.g., an airplane) may consist of hundreds of thousands of CAD models, all interconnected through intricate dependencies [12]. Consequently, there is a fundamental need for robust dependency management tools and strategies in CAD that can track relationships between different models, support impact analysis, and manage product variants [111].

Dependency management is particularly complex in hardware development due to three main factors: (1) the geometric nature of the data; (2) the granularity of dependencies; and (3) the lack of maturity in existing tools. In contrast to software – where functions, variables, and modules convey explicit semantic meaning and clearly defined relationships – CAD models are constructed through sequences of geometric operations or *features*, that lack inherent semantics [17]. Features do not typically carry textual information about their design intent or functional role within the product [101]. As a result, errors or changes are not as obvious; for example, a change could occur within the interior of a model, invisible from the outside view, whereas with software, code changes are more easily detected, highlighted, and summarized. This lack of visibility and semantic clarity makes it difficult to build tools that support effective change impact analysis in CAD.

Dependency management in CAD is further challenging because dependencies may span multiple granularity levels, ranging from low-level sketches and features to parts and assemblies (see Figure 1). These hierarchical relationships complicate both local edits and global coordination.

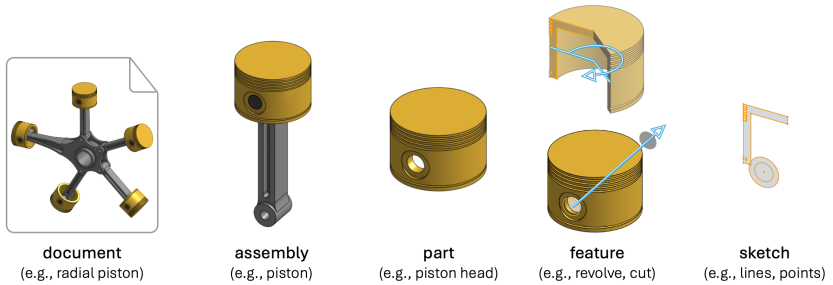


Fig. 1. Granularity levels of CAD artifacts, organized from highest to lowest. A **document** is the top-level container for CAD artifacts and often represents a product. However, due to technical limitations in some CAD platforms, large or complex products may be split across multiple documents, creating "external references." An **assembly** is a model composed of multiple interacting parts. A **part** represents a single physical object and is constructed from one or more features. A **feature** is an operation that manipulates geometry (in the figure, two features, *revolve* and *cut*, create the piston head part). At the lowest level, a **sketch** is a set of 2D line segments that defines a model's geometry. Models adapted from [39].

Furthermore, dependencies can exist between different documents, known as "external references," a functionality supported by major CAD platforms, such as SolidWorks,[1] Fusion360,[2] and Onshape.[3] External references can exist between artifacts at different granularity levels; for example, a circle sketch in one document could define the diameter of a pipe in another document, and any change to the sketch (e.g., increasing the diameter) will automatically update the dependent document.

[1] https://help.solidworks.com/2021/english/SolidWorks/sldworks/c_External_References.htm
[2] https://www.autodesk.com/products/fusion-360/blog/reference-objects-sync-all-contexts/
[3] https://cad.onshape.com/help/Content/updating-references.htm

It can be difficult for designers to trace the connections between different components [44], so best practices have been developed for systematically setting up dependencies. One common approach is "top-down" modelling [26, 27], whereby key parameters are defined in a central "master sketch", ensuring that dependent parts conform to the overall design constraints [20, 108]. Figure 2 illustrates this approach, where 2D sketches in the X-, Y-, and Z- planes define the reference geometries for the overall product. With master sketches, there is one central place where all of the dependency relationships are managed, and in the ideal scenario, changing the relevant element in the master sketch propagates the changes downstream accordingly [27].

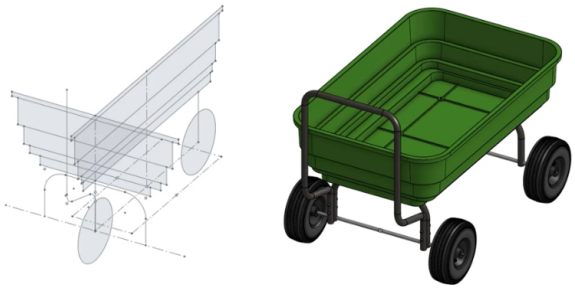


Fig. 2. Complex products typically follow a *top-down* architecture, where parameters are defined in (a) a master sketch, which is used to create (b) the final CAD model. Adapted from [113].

Since multiple collaborators work across different dependent documents, researchers emphasize the importance of tracking dependencies between these external references. Existing efforts have developed automated measures for identifying these dependencies, such as through mapping links [99], applying network approaches to identify independent modules [127], and using algorithms to detect redundant dependencies [47]. Various researchers have also developed graph-based visualizations to help designers understand the dependencies between CAD models [73, 80, 97]. Kozlova et al. interviewed six designers for feedback on their graph visualization of dependencies, finding that users desired interactivity, reduced visual clutter, and support for various levels of granularity [80]. Despite these advances, existing work focuses primarily on visualization and lacks support for managing the dependencies, such as enabling designers to modify dependencies directly within the interface. Beyond dependency management tools, awareness tools in CAD are generally less mature compared to software development. For instance, cloud-based architecture, though emerging, remains underutilized in many professional design organizations [91], which limits collaboration. Sophisticated version control in CAD is still nascent [21, 24], which hinders many aspects of dependency management, such as propagating changes across related documents or models [22]. Given the current limitations of CAD collaboration systems, it is timely to guide the development of more advanced dependency management capabilities.

Existing research on CAD dependency management has introduced various methods to identify and visualize dependencies. However, a critical gap exists in understanding the challenges that must be addressed before effective solutions can be developed. With the exception of Kozlova et al. [80], most studies do not draw insights from real CAD users, and no studies take a systematic approach to understanding the problem. Our research aims to fill this gap by identifying user requirements for dependency management and proposing strategies grounded in empirical findings.

3 Methods

To gain a comprehensive understanding of the challenges and user requirements surrounding CAD dependency management, we conducted an exploratory sequential two-phase formative study [117], and used Braun and Clark's reflexive thematic analysis (RTA) approach [10]. In Study 1, we mined and analyzed 100 user discussions from online CAD forum sites to build an initial understanding of the current challenges experienced by design engineers. This forum thread analysis revealed preliminary themes, which informed the analysis for Study 2, where we interviewed 10 professional mechanical design engineers. By following this multi-modal approach, we aim to overcome the limitations of a single data source. The forum discussions provide access to diverse discussion topics [21] and a large pool of CAD users, enhancing the generalizability of our findings. In contrast, with interviews, we naturally reach fewer CAD users, but we gain in-depth insights that highlight the challenges and opportunities for CAD dependency management. Figure 3 summarizes our research methods.

To determine appropriate sample sizes for both studies, we followed Malterud et al.'s guidelines for *information power* in qualitative research [94], which suggests that the number of participants required depends on the richness and relevance of the data rather than a fixed threshold. While either study alone may not hold sufficient information power, we conjecture that by supplementing and triangulating these two samples [141], we collect the necessary evidence to answer our research question. We discuss further details on study design and sample sizes in the following subsections.

In both Study 1 and Study 2, we investigated dependency management in the context of *Onshape*,[4] a cloud-based CAD and data management platform. Like all parametric CAD software, Onshape enables users to develop 2D sketches into 3D geometries and assemble these parts to create complex models. However, Onshape is particularly advanced in its support for dependency management for several reasons: (1) multiple designers can work on CAD documents simultaneously, removing restrictions on collaborative design; (2) the version control system enables branching and merging, supporting the creation and management of dependencies across different document versions; and (3) the cloud-based architecture eliminates common broken file path issues [129]. Given these capabilities, Onshape provides more sophisticated support than most CAD platforms, and is considered a state-of-the-art system for dependency management. Nonetheless, designers still report persistent challenges, therefore motivating the present study. Although we are interested in understanding dependency management in CAD overall, for this paper, we scoped our data collection to the Onshape platform, following the approach taken in similar studies, such as Kiani et al.'s investigation of help-seeking behaviours in the context of *Autodesk Fusion360* [77].

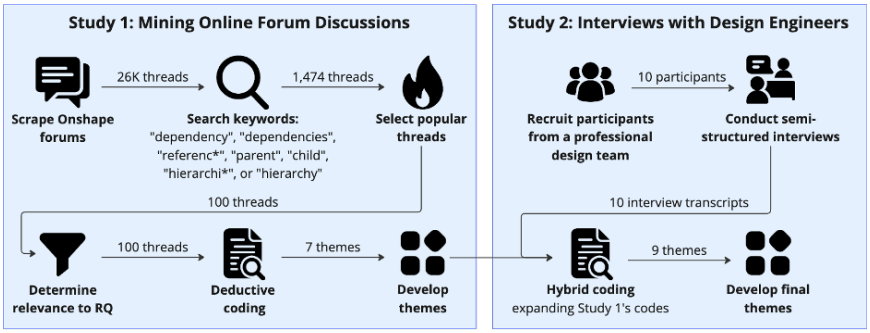


Fig. 3. Overview of research methods for Study 1 and Study 2.

[4] https://www.onshape.com/en/

3.1 Study 1: Mining Online Forum Discussions

The objectives of Study 1 are twofold: (1) to identify pain points and design goals for CAD dependency management tools, and (2) to guide the theme development for Study 2. To achieve this, we analyzed user discussions on online forums, a common data source used in CSCW research for understanding current practices and challenges from a wide range of perspectives [3, 51]. Online forums are important data sources for many domains, but especially for CAD-related topics, since they are one of the most frequently used resources among professional engineers and serious CAD users [77, 100, 112]. In our previous investigation of CAD version control practices [21], we found that forums can be considered communities of practice, offering rich discussions, user complaints, and needs that can inform the design of new features and improvements.

3.1.1 Data Collection. For this study, we collected data from Onshape forums,[5] which are officially affiliated with Onshape and hosted by its parent company, PTC Inc. At the time of our data collection (July 2024), Onshape forums comprised over 26K posts, over 121K comments, and over 10K members. Discussions on the forums fall into various predefined categories, with the largest being *Community Support*, where users seek troubleshooting assistance, and *Improvement Requests*, where users provide product feedback.

To find forum posts relevant to our research question, we first did keyword searching using the following keywords (stemmed): *"dependency"*, *"dependencies"*, *"referenc*"*, *"parent"*, *"child"*, *"hierarchi*"*, OR *"hierarchy"*. We purposely did not stem the word *dependency* to *depend** because it created a significant noise in the data, retrieving each instance of phrases like "it depends". Instead, to be comprehensive, we searched for both *dependency* and *dependencies*. We included the keyword *referenc** to capture discussions of external referencing, and *parent*, *child*, *hierarchi**, and *hierarchy* as these are common terms used in the CAD context to describe artifact dependencies [15, 23].

To collect the forum post content, we used a custom web-scraping tool using Python, built with packages *Selenium*[6] and *BeautifulSoup*.[7] Using our keyword searching criteria, we retrieved any forum posts that contained one or more of these keywords in the title, initial post, or any comments. The data we collected includes each post's title, posting date, the content of the post, and the content in comments. Here, we define *post* as the initial contribution by the original author and *thread* as the full discussion, comprising the post and all subsequent comments.

In total, our keyword search yielded 1,474 threads. Given the heavy workload of manually coding all threads, we narrowed down the dataset by selecting the top 100 most popular threads, defining popularity by the number of comments, as these threads had the highest user engagement, and provided a rich source of content for analysis. The average comment count for popular threads was 63 (ranging from 35 to 401). The first author then reviewed each thread for relevance to our research question; irrelevant threads were tagged as such and excluded. When a thread was excluded, the first author revisited the original larger dataset ($n = 1,474$) and retrieved the next most popular thread, repeating this process until 100 relevant threads were identified. In the end, 14 threads were deemed *irrelevant*, thus the total number of forum threads we reviewed was 114.

The decision to analyze 100 threads was guided by Malterud et al.'s concept of *information power* [94], which states that the larger the information power the sample holds, the lower the number of participants needed, and vice versa. A sample's information power depends on five dimensions: study aim, sample specificity, use of established theory, quality of dialogue, and analysis strategy (case vs. cross-case). For our study, we ask an exploratory research question with a broad aim to understand CAD dependency management challenges, however, our study context is of a

[5] https://forum.onshape.com/
[6] https://www.selenium.dev/
[7] https://pypi.org/project/beautifulsoup4/

specialized field (of hardware development using mechanical CAD) with participants of a specific skillset. While forum posts lack direct interaction between researcher and participant – reducing their information power – they involve rich peer-to-peer dialogue, particularly in lengthy threads with multiple responses, where participants build on each other's contributions. To maximize information power, we prioritized discussions with the most comments, which are more likely to reveal diverse perspectives and rich ongoing exchanges. Across the 100 threads analyzed, there were 78 unique post authors and 841 unique members represented in the comments. We do not claim to capture complete insights on dependency management from all 841 users, but their excerpts indeed contributed to our theme development. Since the forums offer a diverse range of users, we need a bigger sample size (than with interviews) to provide sufficient information power. Given these considerations, we believe this dataset encompasses diverse perspectives from CAD practitioners, and that 100 forum threads provide sufficient data to *"tell a rich story"* [29, p. 56].

3.1.2 Data Analysis. Our analysis followed Braun and Clark's method for reflexive thematic analysis (RTA) [11]. The strength of RTA lies in the active, *reflexive* role of the researcher in the knowledge production process, where the researcher's subjectivity shapes theme development. In this work, the researchers are experts in modern CAD and software engineering, which is key to RTA, since it is our background that provides a unique perspective to investigate dependency management within the CAD context. During the thematic analysis process, the researcher's subjectivity deepens the interpretation of the data [11]. Our analysis approach was *inductive*, given the exploratory nature of our research question. We focused on both *semantic* and *latent* levels to reflect the explicit content of the data while leveraging our expertise to inform our interpretation of the data.

There are six phases of RTA, which follow a recursive process [29]. The first phase, *familiarization*, began with the first author reading through all 100 threads to determine their relevance to the research question, making casual observational notes throughout. The next phase, *generating codes*, involved assigning short descriptive labels or "codes" to data excerpts to identify themes without following a predefined framework, resulting in 290 codes. In the third phase, *constructing themes*, we collated similar codes to develop themes; e.g., a recurring theme was that CAD designers struggled to anticipate how changes to an artifact would affect its dependents. Throughout the theme construction phase, all authors iteratively refined the candidate themes to ensure that each was distinct and had a central organizing concept, resulting in 14 candidate themes and 73 candidate sub-themes. In the fourth and fifth phases, *revising* and *defining themes*, all authors collaboratively reviewed each theme's codes, coded data, and relation to the research question to identify the most meaningful themes. We then defined the following candidate themes: *difficulty in tracing dependency chains*, *lack of overview of project structure*, *poor impact analysis*, *ambiguous dependency freshness*, *difficulty reorganizing models within the hierarchy*, *broken dependencies*, and *disorganized design history*. These seven themes provided an initial framework for the thematic analysis of the interview transcripts generated in Study 2, detailed in the following section.

3.2 Study 2: Interviews with Design Engineers

For our second formative study, we conducted 10 semi-structured interviews with professional mechanical design engineers who use CAD. The semi-structured format allowed us to ask open-ended questions that captured a broad range of topics, while also providing an opportunity to triangulate the themes identified in Study 1. The aim of Study 2 is three-fold: (1) to validate the findings from the forum analysis; (2) to gather rich scenarios and nuances that highlight why dependency management remains a persistent issue in CAD, and (3) to gather feedback on tool features that could improve engineers' awareness and management of dependencies.

3.2.1 Participants. We recruited participants from a large technology organization that produces software and hardware products – for anonymity, we refer to them hereafter as *Company X*. While Company X has a substantial workforce with many departments and engineering teams, we focused our participant recruitment on a particular team (hereafter: *Team Y*) that specializes in the design and development of autonomous robots. We chose this particular company and team for a few reasons. First, Company X is one of the largest organizations (>10K employees) that use Onshape, with a substantial workforce and a diverse range of complex products. This complexity requires sophisticated workflows and robust data management strategies for effective coordination, which allows us to draw more meaningful insights for studying dependency management. Second, Team Y has established a significant history of using Onshape for over four years. Consequently, their practices regarding regular Onshape usage are well-established. Finally, Team Y, makes extensive use of the external referencing features, relying on these workflows to design their flagship robot; thus, we are capturing persistent challenges with using state-of-the-art dependency management tools, not challenges due to a lack of familiarity. All of these factors make Team Y an ideal pool to recruit participants to understand dependency management challenges.

We used purposive sampling [16] to recruit participants from Team Y. We recruited online through a screening questionnaire posted in Team Y's relevant Slack channels. In total, we interviewed 10 participants (represented by ID1-ID10 in Table 1), of which four were women, and six were men. All interviewees have at least two years of experience working in Team Y at Company X and using Onshape professionally. All participants were experienced in multiple CAD softwares, and nine had experience using CAD software other than Onshape in a professional setting. Five participants were senior mechanical engineers, three were mechanical design engineers, and two were team leads. Each member of Team Y works on all the subsystems within the robot, so each interview participant has experience managing dependencies from both sides of the dependency relationship, which was an important experience for our interviews.

In determining a suitable sample size, we used Malterud et al.'s *information power* [94]. We purposefully selected interview participants based on specific characteristics of the organization, team, and individual, to ensure that their insights were highly relevant to our research question. As researchers with extensive experience conducting interviews and studying CAD collaboration, we engaged in in-depth discussions with participants, further enhancing the quality of dialogue. Additionally, because all interview participants belonged to the same company, a smaller sample size was sufficient for identifying patterns within this organizational context. Throughout recruitment and data collection, we continuously assessed the relevance and quality of the data to appraise the information power iteratively, as recommended by Malterud et al. [94] and Braun and Clark [11]. Based on this ongoing evaluation, we determined that 10 participants provided sufficient information power, and concluded data collection at that point.

3.2.2 Procedure. All 10 interviews were conducted remotely via Zoom between August and September 2024. The sessions were audio- and screen-recorded and automatically transcribed using Zoom's transcription service. Each interview lasted an average of 57 minutes, ranging from 47 to 65 minutes. The first author conducted all the interviews.

We began by asking participants about their role, job responsibilities, and experience at their organization. We refrained from asking about specific challenges to avoid biasing their responses based on themes identified in the forums. Instead, we asked them to describe a recent design project and walk us through the entire process – from starting the CAD work to delivering the final output. We encouraged participants to provide examples to ground their responses. For each scenario they described involving the management, modification, or understanding of dependencies in CAD, we asked follow-up questions, such as what artifacts were dependent, what information they needed to

Table 1. Participants in Study 2. Columns with Yrs. indicate the number of years of experience.

Participant	Job Title	Gender	Yrs. using CAD	Yrs. using Onshape	Yrs. at Company
ID1	Senior Mechanical Engineer	M	7 - 9	1 - 3	1 - 3
ID2	Senior Mechanical Engineer	M	10 +	1 - 3	1 - 3
ID3	Senior Mechanical Engineer	M	10 +	4 - 6	4 - 6
ID4	Mechanical Design Engineer	M	10 +	1 - 3	1 - 3
ID5	Senior Mechanical Engineer	M	10 +	4 - 6	4 - 6
ID6	Mechanical Design Lead	M	10 +	1 - 3	7 - 9
ID7	Engineering Team Lead	W	10 +	4 - 6	4 - 6
ID8	Mechanical Design Engineer	W	1 - 3	1 - 3	1 - 3
ID9	Mechanical Design Engineer	W	4 - 6	1 - 3	1 - 3
ID10	Senior Mechanical Engineer	W	7 - 9	4 - 6	4 - 6

understand the dependency relationships, and what challenges they faced. As participants recounted scenarios, they shared their screens while using Onshape, walking us through the features they typically use and demonstrating their design and navigation processes. The study was approved by the University of Toronto's Ethics Review Office.

3.2.3 Data Analysis. To begin data analysis, the interview transcripts were downloaded from Zoom, anonymized, and imported into the qualitative data analysis platform, NVivo-12.[8] Following Study 1's approach, we conducted the six phases of reflexive thematic analysis (RTA) [10] and coded both semantic and latent levels of the data. However, we employed hybrid coding to analyze the interview transcripts, combining deductive and inductive coding [31]. Like in Study 1, the first author began with the *familiarization* phase by reviewing and cleaning the automatically transcribed data while noting initial ideas. Since the data from Study 1 had already been coded, the first author marked instances where interview data aligned with existing candidate themes, along with new ideas that stood out against the previous analysis.

During the *generating codes* phase, the first author systematically analyzed each interview, generating around 360 codes. In the *constructing themes* phase, we built upon the seven candidate themes of dependency management challenges developed in Study 1, organizing codes that matched these existing themes accordingly. When we identified new challenges related to dependency management, we created additional themes to capture these insights. The first author independently coded the interview transcripts, while all authors collaboratively discussed and refined the themes throughout the process. This iterative refinement involved comparing the themes against each other and checking them against the original codes and dataset, per the Braun and Clark method [10]. At this stage, the thematic analysis included 20 candidate themes and 57 candidate sub-themes.

During *revising* and *defining themes*, all authors examined the codes within each theme to confirm they follow a coherent pattern, and revisited the dataset to determine that the themes accurately reflect the participants' original meanings. This interview analysis confirmed five of the seven themes from the forum threads and found two additional themes: *messy navigation of the master sketch* and *dependency conflicts across versions*. Across both formative studies, we identified nine challenges of CAD dependency management.

Finally, in the last phase of RTA, *writing the report*, we developed a deeper sense of how the themes fit together to create a cohesive picture of CAD dependency management challenges. Through this process, all authors collaboratively developed overarching themes that group the nine themes into challenges related to: (1) traceability; (2) navigation; and (3) consistency.

[8] https://lumivero.com/products/nvivo/

4 Findings: Challenges of CAD Dependency Management

This section synthesizes the findings from Study 1 and Study 2. We found nine challenges that design engineers face with managing technical dependencies in CAD projects, summarized in Table 2. To present our results, we have grouped the challenges into three overarching themes: traceability-, navigation-, and consistency-related. Quotes from forum threads are denoted with *F* (for "forum") and the year it was posted in brackets – for example, "(F24)" for a thread posted in 2024. Quotes from interviews are denoted with the participant ID – for example, "(ID1)".

Table 2. Overview of CAD dependency management challenges, as identified through thematic analysis of forum threads and interviews conducted in Study 1 and Study 2, respectively. We summarize the frequency of each challenge based on the number of relevant forum threads (out of 100) and the number of interviews (out of 10) that reported these challenges. A dash (—) indicates that the challenge was not identified.

Category	Challenge	# of Threads	# of Interviews
Traceability-related	Difficulty in tracing dependency chains	36	10
	Poor impact analysis	19	8
	Broken dependencies	11	—
Navigation-related	Lack of overview of project structure	23	4
	Difficulty reorganizing models within the hierarchy	5	2
	Disorganized design history	8	—
	Messy navigation of the master sketch	—	5
Consistency-related	Ambiguous dependency freshness	13	4
	Dependency conflicts across versions	—	1

4.1 Traceability-related

Traceability-related challenges focus on the difficulty of understanding how different artifacts are interconnected, including tracing the origins and histories of designs. Additionally, inadequate traceability hampers designers' ability to understand how artifacts are impacted by changes.

4.1.1 Difficulty in tracing dependency chains. Before beginning any modelling work – whether modifying an existing design or creating a new one – CAD designers need to know how design artifacts are interconnected. This requirement is central to parametric CAD [50], which requires thoughtful consideration of how dimensions drive the overall design. However, tracking all these relationships becomes challenging in complex product development, where products consist of numerous parts and subsystems with intricate dependencies. As one forum user expressed, *"external references are problematic to humans, because we do not, and can not, scan through everything we ever knew which bears on the problem at hand, every time we make decisions"* (F15).

Existing tools typically only allow users to trace dependencies for one feature at a time (see Figure 4). Users must manually select a feature, inspect its dependencies, and then repeat this process for each related feature. Currently, there is no way to generate a comprehensive list that spans multiple levels of granularity (e.g., across features, parts, assemblies). This process becomes especially challenging when several layers of hierarchy exist (as described in Section 2.3.2), since dependency tracing is only possible within a single document. As a result, designers must trace the model's history across multiple documents to understand the design intent. Often, this involves discovering that one model depends on another in a separate document, which itself may depend on yet another model, leading to a laborious and manual tracing process. As interviewee ID5 explained, "*you could track something from one part to the next part, and you will go into Master Sketch, and that Master Sketch derived another Master sketch, etc. [...] You would have traversed 8 different places*

before you actually found out where the change was." Participants identified two primary scenarios in which they perform dependency tracing: (1) to understand a model's history before making modifications, and (2) to investigate the root cause of a design error. It is important to note that both scenarios involve tracing *upstream* across documents, whereas it is *"almost impossible to track downstream"* (ID2). ID2 further explained that if a tool could list all downstream features linked to a sketch, this added visibility would influence their design decisions. For example, *"if [the sketch] points to a million different stuff, then probably I'll leave it alone, or be really, really careful"* (ID2).

The difficulty of tracing dependency chains was the most common challenge, mentioned in 36% of forum threads and by all 10 interviewees.

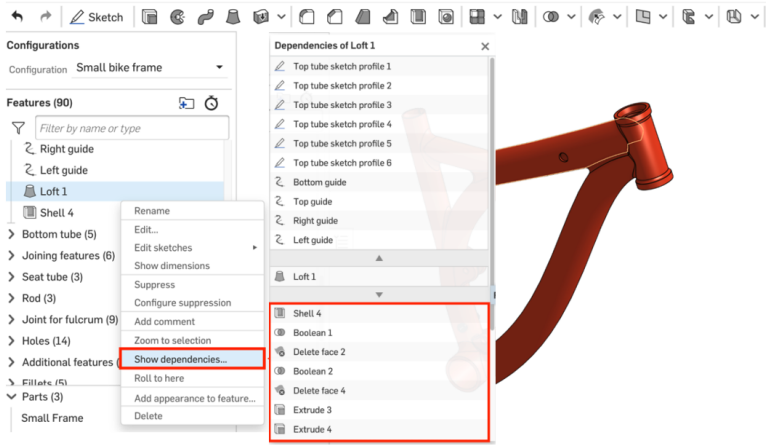


Fig. 4. Onshape's "Show dependencies" feature enables traceability within a document. Adapted from [74].

4.1.2 Poor impact analysis. Impact analysis is understanding how changes to a parent artifact propagate downstream, which, in CAD, is often a difficult and manual task. Onshape – compared to other CAD platforms – improves this process by allowing change propagation for models within the same document, as one interviewee noted, *"if the part geometries were just defined in the same document, you will never have to do [change propagation]"* (ID5). However, a significant challenge remains: while the software can manage change propagation for a single dependency layer, it cannot efficiently handle multi-level hierarchies that span across multiple documents. As interviewee ID1 said, *"it takes a lot of time and effort to cascade a change down into the hierarchy. It might be 5 levels deep or something."* Although manually reviewing and approving each propagated change in every affected document is laborious, some users find this stepwise process purposeful. One forum user explained that it allows designers to manually check each modified document *"to ensure that each parent assembly is referencing the correct/latest released version of a child part"* (F23).

Given the tedious nature of this task, it can be impractical for designers to verify the impact in each document, which can cause unintended changes to models without the designer's awareness. Interviewee ID3 described a common scenario: *"every time there is an update of the master sketches, you need to go to all the documents, update that reference, and then cross-validate."* Without tools to systematically track these impacts, engineers rely on heuristics and a deep understanding of their design to navigate dependencies selectively, since checking every child artifact for changes is impractical. As ID3 further emphasized, *"you need to be a little bit selective. But that is based basically [based] on the knowledge that we have. We don't have a bulletproof method here. It's just too much."*

The idea of automatic change propagation is appealing to streamline this process, but users have expressed caution about this approach. One forum user stated, *"I'm interested in the possibility of automated version updating in the assemblies, though I suspect that will get me in big trouble when at some point when I forget to check an assembly that's been auto-updated"* (F21). With or without automation, users highlight the importance of having a real engineer review the changes, stating, *"don't forget that the user is the ultimate judge of their intent. Please make sure we can override any magic if the magic isn't working"* (F15). Overall, CAD users find manual change propagation tedious, while fully automated propagation can be prone to errors. Thus, the ideal solution may be to streamline the review and approval process to improve efficiency, while maintaining design integrity by systematically identifying changes.

The challenge of poor impact analysis was mentioned in 19% of forum posts and by 8 interviewees.

4.1.3 Broken dependencies. As part of refactoring practices, designers may clean up or archive old designs without realizing these are still linked to other parts, unintentionally creating broken dependencies. One forum user shared a typical scenario: *"I have a document with a sketch [derived] from another document which I trashed thinking it was not needed. [...] It really seems like this shouldn't be possible, at least without a warning"* (F22). In this case, the designer was unaware of the dependency and received no warning before creating the error.

Another frustration users face is the lack of detailed information about the root cause of the error. For instance, Onshape's generic message *"Some dependencies are missing"* leaves users uncertain about how to fix the issue. One user drew the comparison, *"This is a bit like a compiler getting to the end of the compilation and saying, 'Your source program has some errors. Please fix them.' Not useful"* (F22). Without knowing what the dependency originally referenced, designers cannot repair or reroute the relationship to another artifact.

The challenge of broken dependencies was mentioned by 11% of forum threads but by no interviewees.

4.2 Navigation-related

Navigation-related challenges involve the efficient retrieval and organization of information. These challenges often arise from disorganized data, requiring designers to invest significant time and effort to make sense of the cluttered dependency information.

4.2.1 Lack of overview of project structure. During CAD modelling, designers often have multiple documents open to review the relevant information, but they typically focus on design work within a single document at a time. Working predominantly within one document can create silos, making it challenging for designers to understand how the design of the product is progressing at a high level; this is especially the case when multiple collaborators are making changes to different subsystems that may not immediately impact one another. Consequently, the inability to see the bigger picture of these interconnected artifacts presents a significant challenge. One forum user expressed, *"with design information distributed between documents, branches, versions, revisions [...], it's very hard to look at a project and quickly get an understanding of what's going on"* (F19).

Feature request: visualizing external references. Designers have suggested that a visualization would be useful to address the lack of overview problem. Interviewee ID9 stated, *"the only way for us to find out how [designs are] used is really by right-clicking and "Show dependency". It's all text information [...], and we actually don't have a visualization about how documents are linked together"* (ID9). This sentiment was echoed in the forum posts, where one user wrote, *"I think a visual representation of the files makes it easier for a visual program"* (F14), indicating that such a tool would align with the way design engineers think and work, given the inherently visual nature of CAD. Additionally, since the products being designed are physical, some users expressed that

the dependency structure should be *"ideally based on the assembly structure"* (F19), mirroring the architecture of the intended product.

This lack of overview challenge was mentioned in 23% of forum posts and by four interviewees.

4.2.2 Difficulty reorganizing models within the hierarchy. As external references are created between CAD documents, these documents form a hierarchical structure. During the initial design phases, designers often struggle to determine where certain dimensions or sketches should be placed within the hierarchy; for example, in our previous car design scenario (Section 2.2), designers need to decide the hierarchy of the axle, wheel, and frame dimensions. ID5 explained, *"if I have one Part [A] here and another Part [B] here, and then they have some sort of relation in common, then I don't want to be designing this [Part A] in isolation and this [Part B] in isolation. I want them to have a common ancestor, a common source of truth."* To achieve this, the sketch that defines the interaction between the two documents is moved to an upper-layer document, so that any change will propagate to both lower-layer documents. This approach works in theory, but as the design progresses, it can become necessary to reorganize the hierarchy. For example, lower-layer dimensions might need to be moved higher if they will now drive multiple documents, or vice versa, as ID5 noted: *"if a sketch is no longer needed, then we shouldn't define that in an upper document, because that just creates an unnecessary step."* However, interviewees expressed that changing the hierarchy is not that simple, because, *"when moving the sketches, everywhere these are referenced, you have to reintroduce that reference somewhere"* (ID4). For instance, in our car example, if the frame depends on the axle, which depends on the wheel, and now the frame needs to be the driving model, designers must carefully adjust all related references. Identifying which dependency relationships need to be edited, removed, or rerouted is not obvious, and there is no systematic or automated way to inform the user of which references require changes.

Similarly, interviewees described difficulties in decomposing one large document into smaller ones. ID9, the designer who originally developed the master sketch architecture, explained: *"it was a pain to split this master sketch [...] into 3. We needed to think about how we define the boundaries."* This process becomes more error-prone when done collaboratively, as multiple people may be rerouting these dependency relationships across various product subsystems. Without full awareness of each other's actions, errors such as circular references can occur. ID9 warned: *"you need to plan through like which part is dependent on which and how to avoid a loop reference because it's really easy to chase your own tail when you all work together."*

In both scenarios – reorganizing the hierarchy or decomposing documents – moving references must be handled carefully to avoid errors when the references are not properly redefined. This challenge was mentioned in 5% of forum posts and by two interviewees.

4.2.3 Disorganized design history. In CAD, the feature tree is a hierarchical representation of all the features in a 3D model or assembly, reflecting the design history of the CAD document. As designs become more complex, designers report increasing difficulty navigating the tree. One forum user explained that when *"starting to deal with bigger part studios (Still relatively small at 65 parts), [they are] really feeling the need for list organization"* (F14). A common frustration is the inability to easily reorder features so that they are displayed in a more intuitive order. For example, designers expressed the desire to group dependent features. As one user explained, *"I often want to reorganize my tree so that features are as far up as they can be without breaking. I'd be so happy if I could right-click a feature and just pick 'Move Below Next Parent,' and it would push it as far up as possible in the tree"* (F18). The ability to reorganize the feature tree would provide designers with a clearer overview of how certain features relate. One user highlighted the need for collapsing certain features together, stating, *"one of the main reasons for organizing a feature tree via grouping (like*

folders) is to condense it so that the sequence of actions can be more easily understood by a newcomer or your future self" (F15).

This challenge was mentioned in 8% of forum threads, but did not appear in the interviews.

4.2.4 Messy navigation of the master sketch. In complex product development, designers often employ a "master sketch" architecture to define the critical dimensions used throughout the product (as described in Section 2.3.2). However, as more sketch elements accumulate, the master sketch can quickly become overwhelming and difficult to read. ID10 captured this issue, stating, *"there's about a billion things. And there's dots and lines over everything. It's a complete haystack, [and] very rapidly, because we're working in 3 dimensions, [the master sketch] becomes unreadable."* This messiness of the master sketch not only hinders effective navigation but also increases the risk of errors. For example, densely packed sketches make it challenging to select the correct reference geometry, as ID9 noted: *"if two lines are 3mm apart from each other, it's easy to get things wrong."*

To improve readability, designers often want to *"clean it up"* (ID10) by reorganizing or removing unnecessary sketch elements. One common scenario is that *"some sketches, like lines [or] reference points, are no longer used, as the design changes and updates"* (ID9). However, Onshape does not alert the user that a sketch element is no longer needed and can be safely removed. ID9 described their two options: *"the first way is to read it, think about it, and understand it, which takes so much time that it's impossible. The second way is just to delete it and see which document goes red [i.e., has an error], which is also not very optimal."*

If designers are unaware of a sketch's downstream dependencies, modifying or removing it can lead to unintended consequences (Section 4.1.2). As ID10 explained: *"If I delete [a sketch element], I'll break [all these] parts, and I won't know about it until I update all references. And if I don't know where those references are, I can't fix them. Someone might find that out 2 months later when they update for some completely other reason, and then everything breaks, and they don't know why."* This uncertainty makes designers hesitant to delete anything for fear of causing broken dependencies. As ID4 said, *"stuff stays in [the master sketch] for a lot longer than it needs to, because the safe thing to do is not to delete it. You don't know what chaos you'll cause if you get it wrong."* Consequently, outdated features remain in the sketch, further adding to the clutter and increasing the risk of accidentally selecting an outdated element to reference later on.

The challenge of messy master sketches was significant, mentioned by 5 out of 10 interviewees.

4.3 Consistency-related

Consistency-related challenges arise when versions of dependent documents are not synchronized. This leads to two main issues: designers may be unsure whether dependencies are up-to-date (i.e., the freshness of the dependency), and conflicting versions can occur.

4.3.1 Ambiguous dependency freshness. External references in CAD are intended to allow parts and subsystems to be developed in parallel, facilitating collaboration and design efficiency. As design work progresses, new versions of documents are created, and dependent artifacts need to be updated to ensure changes transfer correctly across documents. When a document has a new version, Onshape sends notifications to all downstream documents, indicating that the dependency is outdated and requires updating. However, it is often unclear whether the change actually impacts other designs. This ambiguity becomes particularly challenging in the CAD context because each document may contain various artifact types (e.g., sketches, features, parts, assemblies), and a change to any one artifact will trigger a new version for the entire document – even though the specific dependent artifact did not change. Assessing the relevance of this 'new version' notification across numerous dependent documents is a time-consuming and complex task. One forum user described this frustration: *"I noticed that ANY change to information referenced in a drawing, whether*

the change affects what appears on the face of the drawing or not, requires the drawing to be updated. [...] I spend a lot of time hitting the 'update' button and waiting for the drawing to regenerate" (F19).

To avoid missing critical updates, some designers err on the side of caution, creating new versions and updating references frequently. However, this approach has its drawbacks, as another forum user explained: *"Let's say I have a Document with 10 tabs and I change one of them. I'm happy with the change, so I create a new version... well what just happened is that I just versioned 9 Elements that DID NOT CHANGE. To me, that is not a good Data Management practice and creates a lot of overhead keeping track of what's changing and what's not"* (F19).

Beyond substantive design changes, even minor metadata updates in a document can trigger notifications, further adding to the noise. As interviewee ID1 observed, *"Onshape is meant to only signal version updates when they have an impact. But, Onshape's definition of an impact might differ from what we think. [The software] might think that changing some description is a version change."* This notification system's ambiguity and the lack of granularity in alerts make it challenging for designers to distinguish between meaningful updates and unnecessary distractions.

The challenge of ambiguous dependency freshness appeared in 13% of forum posts and four interviews. Due to the frustration this issue caused, Team Y developed an in-house tool to automate this version update process. This tool generates a list of all "out-of-date" dependencies for any document, enabling users to select and update these references in bulk, which significantly streamlines the process and *"kills all the noise of the blue dots [i.e., update icons]"* (ID4).

4.3.2 Dependency conflicts across versions. Dependency conflicts in CAD arise when documents reference different versions of another document. If dependencies fall out of sync, design components can become incompatible. For example, ID1 highlighted, *"if I [reference] revision B in a product and revision F in a later product, how do you actually ensure that they work together?"* Moreover, this issue extends beyond the CAD environment to physical, real-world products. ID1 noted that *"we're gonna have multiple versions of the bot running around [...] with multiple references to these master sketches. A physical product out there won't ever have a trigger to update, right? So how do we keep track of all of those interweaving versions?"* The challenge becomes even more complex when trying to maintain consistency across various product lines, each tied to different states of the same master sketch.

This challenge was only mentioned by ID1 and did not appear in any forum discussions; however, we include it in our findings because of the significant negative consequences of version conflicts.

5 Discussion

Here, we summarize the identified challenges and outline key design goals for a CAD dependency management tool. We then present potential features, design mockups, and a user scenario to demonstrate how the proposed tool can address these challenges. Additionally, we discuss our study's main implications, concluding with limitations and future work directions.

5.1 Design Implications for CAD Dependency Management Systems

Our empirical investigations provide a comprehensive identification of dependency management challenges in cooperative CAD work. As the next step, we have distilled these findings into design goals and developed initial tool concepts to address the key challenges outlined in Section 4. We focus on seven of the nine challenges identified, excluding the challenges of *Disorganized design history* (4.2.3) and *Messy navigation of the master sketch* (4.2.4). These two challenges each pertain to a very specific window in CAD – namely, the version control panel for design history, and the sketch interface for the master sketch – and would benefit from specialized tool support integrated within these specific windows. We, therefore, leave these two challenges for future work, while

focusing in this paper on the dependency management challenges that apply more broadly to external references. Based on our formative study findings, we outline four design goals and ten features for a CAD dependency management tool, summarized in Figure 5 and discussed below.

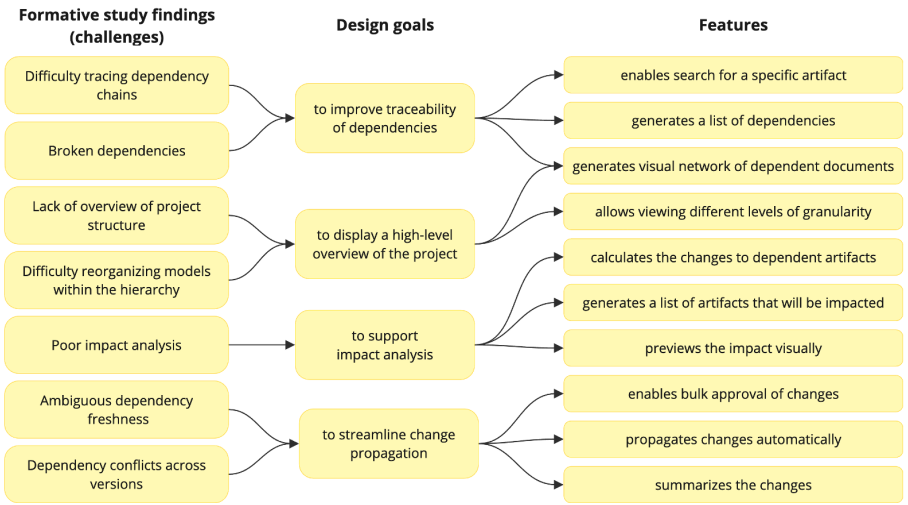


Fig. 5. Overview of findings, design goals, and features for a dependency management tool.

5.1.1 Goal 1: To improve traceability of dependencies. CAD designers struggle to trace dependency chains (4.1.1), making it crucial for the tool to enhance awareness of how artifacts are linked. The tool should first allow users to search for a specific artifact (e.g., a document) and quickly locate it within the project. Next, the tool should generate a list of dependencies, showing both upstream (parent) and downstream (child) relationships. This list should be adjustable to show different levels of granularity, such as sketches, parts, and documents. By incorporating these features, designers can avoid the time-consuming task of manually searching across multiple documents. This improved traceability would also mitigate the risk of broken dependencies (4.1.3), as users can confirm that no artifacts depend on a design before archiving or deleting it.

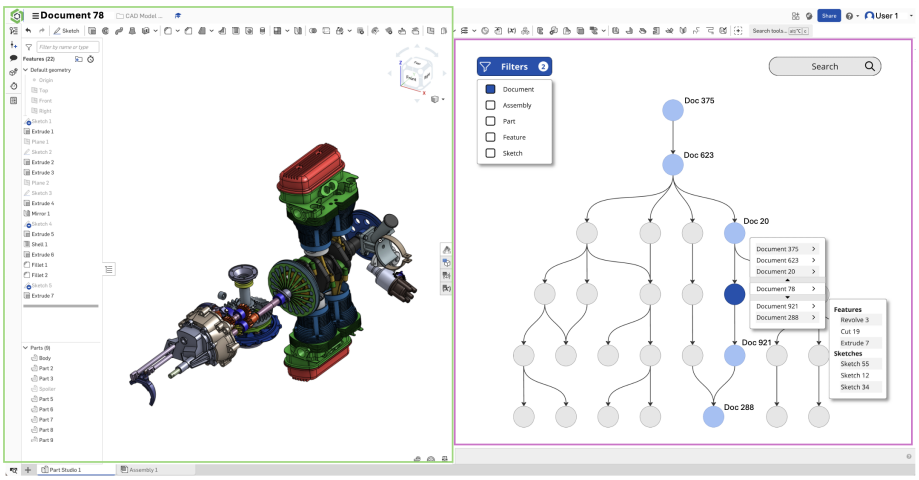
5.1.2 Goal 2: To display a high-level overview of the project. Designers lack a clear overview of the project structure (4.2.1), which makes it difficult to understand how documents relate to and impact each other. To support this need, the tool should generate a high-level visual network graph that maps the dependencies between documents. Users should be able to filter this view by level of granularity and expand or collapse nodes to enhance visibility. To further enhance awareness, users should also be able to select a specific document and visually trace the associated dependency chains. This visualization can also support decisions about document placement within the project hierarchy (4.2.2); for example, if a document appears high in the graph but has few or no dependents, it may prompt users to reconsider its placement.

5.1.3 Goal 3: To support impact analysis. Another challenge to target is the difficulty in identifying and understanding the effects of a change on related components (4.1.2). Manually verifying that changes have been correctly propagated is a tedious and error-prone task. To address this, the tool should preview the changes to the user in two ways: (1) generating a text-based list of artifacts that will change and (2) visually previewing the impact in the network graph. Once the user makes the desired changes to the model, the tool should calculate and display the impact of these changes

on the graph; for example, if the wheel diameter of a car is increased by 5 cm, the tool should update the position of the axle accordingly and indicate that the axle document has been modified. Impacted nodes should be highlighted, with nodes with errors or conflicts flagged (perhaps in red). This visual preview enables users to anticipate the changes before committing them.

5.1.4 Goal 4: To streamline change propagation. The fourth goal addresses the challenges of ambiguous dependency freshness (4.3.1) and version conflicts (4.3.2). To mitigate these issues, the tool should support bulk approval of changes after reviewing the preview. Once approved, the tool should automatically propagate the changes, create new versions of the affected documents, and produce a summary report outlining all changes. This workflow ensures that design changes remain synchronized throughout the project. This idea is inspired by Team Y's internal tool (described in Section 4.3.1), which detects a document's out-of-date references and enables batch version updates. Considering how critical this issue was for their team – enough to warrant developing a custom solution – we include this feature in our design goals. Our design concept builds on the foundation of Team Y's tool by extending its functionality project-wide, aiming to provide similar value to other hardware development teams. Additionally, we propose generating change reports to provide transparency in the update process and automatic documentation, a long-standing pain point in CAD version control [97].

5.1.5 Tool Concept & Scenario. The design goals and features that we proposed offer general implications for the builders of CAD platforms. However, to illustrate how such a tool could function, we developed a series of initial mockups. We propose a plug-in tool integrated into the CAD interface, allowing designers to easily access relevant dependency information within the context of their current work (see Figure 6). In the plug-in window, a network graph maps the dependency relationships between documents, represented as nodes, with arrows pointing from parent to child. While all linked documents are displayed, the graph highlights the current open document (e.g., *Document 78*) in dark blue, and all its parent and child documents in light blue for clear visibility. A pop-up window can also display a list of parent and child documents, with the ability to view more levels of granularity, such as the specific dependent features or sketches.



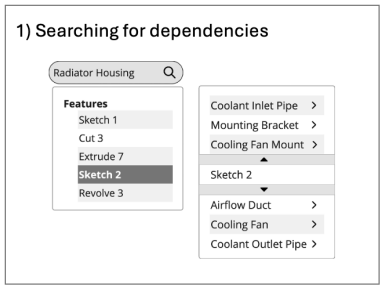
(a) Typical CAD workspace (b) Plug-in window

Fig. 6. A tool can be implemented as a plug-in directly in the CAD platform interface. Note that these document names are dummy placeholders (e.g., "Document 78"). CAD model adapted from [128].

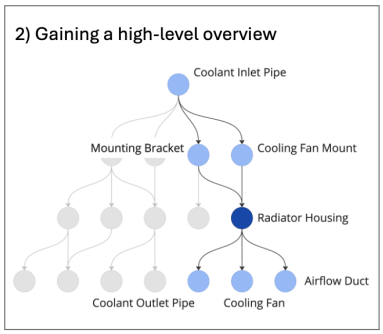
The following user scenario illustrates how such a tool can improve CAD dependency management. Imagine Emily, a junior mechanical design engineer, preparing for a design sprint to improve the airflow of a cooling system. Emily has been assigned to redesign several components of the cooling system and integrate the changes without disrupting the overall product architecture.

Searching for dependencies. Emily's first task is to update the design of the radiator housing. Aware that it depends on other parts of the cooling system, Emily avoids manually searching through numerous interconnected CAD files and folders, and instead uses the newly implemented dependency management tool. She types "Radiator Housing" into the tool's search bar (Figure 7a) and receives a list of upstream and downstream dependencies. The list not only provides direct links to relevant documents but also allows Emily to explore relationships at different levels of granularity, such as sketches and parts. This feature saves her from manually cross-referencing documents, giving her immediate clarity on which artifacts are connected.

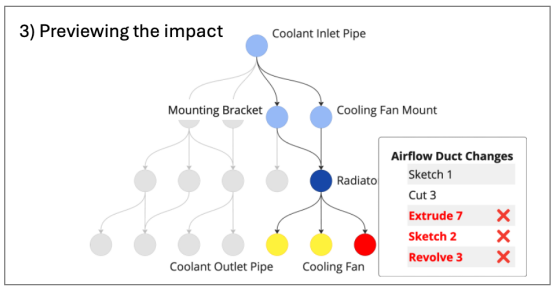
Gaining a high-level overview. Emily needs to understand how changes to the Radiator Housing will affect other documents in the overall product. She uses the high-level overview feature of the tool (Figure 7b), which generates a visual network graph showing all documents and their dependency relationships within the cooling system project. Documents are represented by nodes, with arrows illustrating the dependencies. Using filters, Emily narrows her view to only the documents related to the Radiator Housing, revealing documents like the Airflow Duct and the Cooling Fan. Emily sees how modifications to the Radiator Housing may ripple through the cooling system. She expands nodes to explore specific sketches within the housing that might impact other documents, ensuring that no dependencies are overlooked.



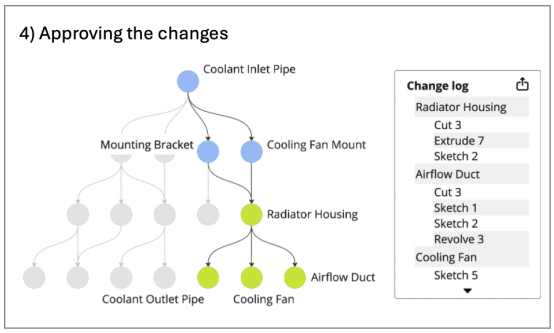
(a) Search for a document's dependencies.



(b) Visualize dependencies in a network.



(c) Preview the impact on dependent documents.



(d) Bulk-approve the change propagation.

Fig. 7. Example user scenario of a CAD dependency management tool.

Previewing the impact. Emily now modifies Radiator Housing. Before finalizing the changes, she activates the impact analysis feature (Figure 7c), to preview the changes. This generates a text-based list and a visual network graph that shows all affected artifacts, with nodes impacted by her changes highlighted in yellow. The Airflow Duct is flagged in red, signalling a potential conflict. The tool provides details on the flagged nodes, alerting her that her modifications interfere with several features (e.g., *Extrude 7*, *Sketch 2*). With this information, Emily can tweak her changes or consult a senior engineer to avoid introducing new problems.

Approving the changes. After reviewing and resolving the design changes, Emily selects the option to bulk-approve the changes (Figure 7d). The tool then propagates the changes, creates new versions of all updated documents (highlighted in green), and generates a summary report, giving Emily a clear record of the changes made to the cooling system's design.

With this tool, Emily no longer needs to painstakingly trace dependencies, manually search through documents to find affected parts, or worry about unknowingly changing another designer's model. Although the design goals and features that we proposed here are by no means exhaustive, they target key challenges of CAD dependency management. Therefore, it is in the interest of CAD platform builders to implement such tools to enhance designers' awareness of dependencies and improve the efficiency of collaborative design.

5.2 Modularity: A Proactive Dependency Management Approach

Our study was motivated by the increasing complexity of hardware products, which comprise numerous components and even more interdependent relationships that are challenging to manage. We also proposed design goals for platform builders to help CAD designers better handle these dependencies. However, this is just one perspective to address the problem. We conjecture that managing dependencies can be approached: *reactively*, by alleviating the challenges we identified, and *proactively*, by adopting best practices that prevent challenges from arising in the first place.

Proactive management involves integrating strategies during the design process to avoid potential dependency issues. CAD researchers recommend minimizing the number of entities that depend on a single component to avoid unwanted changes [116], advising against overloading a single artifact with multiple dependencies. This best practice contradicts the master sketch philosophy, and may require a reevaluation of that architecture. For instance, designers could consider product modularity (the intentional decoupling of components [6]), by decomposing one highly intricate master sketch into several more manageable modules. In our interviews, we observed an emerging strategy that organizes master sketches in a hierarchy. At the top level, a global master sketch defines shared parameters, which are referenced by a set of mid-level master sketches, each of which governs a localized set of artifacts, creating a multi-layered hierarchy. This strategy extends prior CAD literature, which typically assumes a single skeleton structure [27]. While this modular approach can support more proactive dependency management, it also introduces a new challenge: designers must make deliberate decisions about where to draw boundaries between sketches.

To improve modular design in CAD, future systems can draw from well-established software engineering principles. For example, object-oriented design patterns such as *Facade*, *Adapter*, and *Mediator* from Gamma et al.'s catalog [49] offer mechanisms to abstract internal complexity and define standardized points of interaction between components. Similar mechanisms in CAD could help isolate subsystems and make interdependencies more explicit and manageable. Currently, CAD models often depend on implicit geometric references (e.g., a line in one sketch defining the diameter of a part in another document), which are difficult to inspect or refactor. Analogous to software's *interface segregation principle*, which encourages designing smaller, more focused interfaces between software components [98], CAD platforms could enforce clearer contracts at reference points – e.g.,

requiring users to define named interface geometries rather than linking arbitrary sketch elements. While a promising direction, it must be adapted for hardware development, where parts can impact each other even without explicitly defined dependencies; this challenge is especially critical in assemblies with moving parts, where the motion path of one part can interfere with another in a separate module in ways that are difficult to predict or detect.

Additionally, CAD's top-down architecture resembles software's layered architectural styles, yet lacks the tools to manage such separation. Centralized "master sketches" are conceptually similar to central configuration modules in software, but CAD lacks automated tools for detecting violations of architectural boundaries or for restructuring modules when interdependencies grow tangled. Tools inspired by software's dependency injection and module boundaries [82, 133] could assist in refactoring CAD models as the design evolves. Our participants' frustrations with *"chasing their own tail"* when resolving circular references underscore a need for dependency analyzers akin to those used in large software systems.

Importantly, modularity is not just a technical concern but a socio-technical one. In distributed teams, unclear dependencies often become coordination bottlenecks. As in software engineering, where Conway's Law implies that modular code reflects team communication structures [30, 67], CAD platforms could make modular boundaries more visible and enforceable to aid cross-team collaboration. By surfacing shared "interfaces" and ownership of different modules, CAD systems could better align technical architecture with organizational roles, reducing accidental overlap and misalignment across subsystems.

In summary, while modularity has long been a design ideal in software development, realizing it in CAD demands a rethinking of tooling support, grounded in technical patterns and software engineering collaborative practices. CSCW researchers are well-positioned to explore how these concepts translate to the CAD domain, examining how modularity affects collaboration and how teams negotiate, evolve, and contest modular boundaries in practice.

5.3 Implications for Collaborative Work

Our work focused on the management of technical dependencies, which refers to the relationships between CAD artifacts. Designers must be aware of such dependencies in order to successfully create and modify designs. However, understanding technical dependencies is only the first step, and a natural progression is to explore how to manage the dependencies among people and teams [127], i.e., *work dependencies*. Below, we revisit the concepts of coordination and awareness to discuss the role of dependency management in collaborative design.

Levels of collaborative activity. Through the lens of Bardram's levels of collaborative activity [7], we discuss how dependency management shapes coordination, cooperation, and co-construction. Better awareness of technical dependencies supports **coordination** by clarifying which tasks depend on others, and how they should be accomplished (e.g., in series, in parallel) [43]. For example, project managers must determine which documents should be grouped together in a design sprint. Suppose subsystems A, B, and C are interdependent, and A and B are modified during the sprint; it makes sense also to include subsystem C. Redesigning related components concurrently reduces the likelihood of dependency conflicts and backward compatibility issues [138].

Beyond coordination, **cooperation** occurs when individuals focus on a shared design goal and make situated adjustments to their own and others' actions accordingly [7]. When designers know the technical dependencies associated with the CAD artifacts they are working on, it becomes easier to identify with whom they need to communicate [18]. For instance, if a designer knows that a colleague's part references a parameter they plan to change, they may proactively discuss the change, ensuring mutual understanding and reducing the risk of conflict [127].

Finally, dependency management plays a role in **co-construction**, where collaborators jointly shape both the means and object of work [7]. We saw signs of co-constructive activity in our interviews, where participants described an emerging practice of multi-layered master sketches, and a custom tool that maintains dependency freshness within this hierarchy. Co-construction is both driven by, and a response to, increasingly complex technical dependencies. As CAD projects become more modularized, co-construction will involve reconceptualizing how tasks are distributed across teams – shaping not only what is being designed but how the work itself is structured.

Moving towards we-awareness. As introduced in Section 2.1, we-awareness is the socially recursive knowledge that collaborators have of each other [135], which is essential for successful distributed collaboration. Our proposed dependency management tool focuses on surfacing technical dependencies, which can support collaboration, as discussed above, but it does not fully address the need for we-awareness. A promising direction for future work is to incorporate social signals into the dependency graph – for instance, indicating who is actively making changes to which documents, and the status of their changes (e.g., in progress, approved). By providing this kind of contextual information, the tool can enhance spatial awareness [120] and facilitate the reciprocity needed for we-awareness.

Dependency management in other collaborative domains. In large-scale **software development** projects, developers often navigate intricate webs of dependencies. Changes in one module can cause unforeseen ripple effects across the system, harming maintainability and security. Our approach to visualizing dependencies and implementing proactive consistency checks can enhance tools like DepsRAG [2], a multi-agent framework designed to enhance developers' understanding of software dependencies using large language models. By integrating these strategies, software teams can achieve better traceability and mitigate risks associated with dependency changes.

Within **open-source ecosystems**, dependency management poses unique challenges. Many communities struggle with managing dependency risks, often overwhelmed by vulnerability alerts. Researching these alerts is resource-intensive, and attempting to address all of them can be even more costly [8, 76, 110]. Our findings suggest that integrating proactive consistency checks and dependency visualization tools can aid in prioritizing and addressing the most critical vulnerabilities, thereby enhancing the security and stability of open-source projects.

Data scientists face similar dependency awareness challenges during exploratory data analysis (EDA) [86, 140], particularly when trying to track dependencies between code cells and group them into coherent task-specific segments. Our findings on modularizing large, layered master sketches in CAD may offer transferable lessons – such as creating hierarchical structures that could help contain the "messiness" [65] of computational notebooks. Additionally, our study's emphasis on visualizing dependencies and implementing proactive consistency checks can significantly enhance collaborative EDA processes. For instance, tools like MLCask [89] have been developed to manage component evolution in collaborative data analytics pipelines. By incorporating similar visualization and consistency management techniques, teams can improve the robustness and reproducibility of their workflows.

In conclusion, while our study primarily addressed technical dependencies in CAD, the implications of this work can help design teams better plan activities and facilitate collaborative efforts. Although our focus is on CAD, these implications also shed light on awareness and coordination needs in CSCW domains as a whole, where complex dependencies are common in fields like software development [136] or data science [96, 144].

5.4 Limitations & Future Work

This paper focuses on understanding the user requirements for managing technical dependencies between CAD artifacts, but as discussed in Section 5.3, this does not encompass all the dependency needs in cooperative CAD work, such as work dependencies. Nonetheless, we believe that our systematic investigation of dependency management challenges is a necessary first step, and future work will extend this investigation to explore socio-technical dependencies.

Next, like other empirical studies that conducted formative research with participants from a single organization [75, 103], our findings have limited generalizability. However, to mitigate this limitation, we included insights from online forum discussions to broaden the scope of perspectives represented in our dataset.

Our study is also limited by its focus on a single CAD platform, Onshape. We intentionally chose Onshape because of its advanced support for external references, but we cannot claim that the challenges we identified are generalizable to all CAD software. Nevertheless, selecting Onshape was essential, allowing us to investigate persistent challenges even within a state-of-the-art system.

Finally, in this work, we focused on a two-phase formative study of engineers' experiences with CAD dependency management, finding several fruitful user needs. Based on these findings, we developed four design goals for a dependency management tool to address these challenges and proposed initial tool concepts. It must be recognized that these design goals are not exhaustive, and the features we present do not represent a comprehensive checklist for CAD dependency management. For future work, we will develop and implement a functional tool based on the concepts proposed in this paper. We will evaluate this tool using one or more methods, such as usability studies [140, 142], design walkthroughs [63], or longitudinal studies [125], to further refine the design and assess its effectiveness.

6 Conclusion

In this work, we conducted two empirical formative studies, seeking to better understand CAD dependency management challenges. Through a thematic analysis of 100 popular user discussions in online CAD forums and semi-structured interviews with 10 professional designers, we uncovered nine key challenges that hinder workspace awareness and effective coordination within hardware development teams. In an effort to enhance CAD dependency management, we distilled these challenges into design goals and corresponding features that are essential for a CAD dependency management tool. Beyond challenges and design goals, we also contribute initial tool concepts that could implement these features as a plug-in for CAD platform interfaces. Our findings and proposed solutions lay the groundwork for future development and evaluation of tools that can better support designers in managing CAD dependencies and improving collaboration in the hardware development domain.

Acknowledgments

We acknowledge the support of the Government of Canada's New Frontiers in Research Fund (NFRF), [NFRFE-2022-00543]. We also thank our interview participants for generously sharing their time and valuable insights.

References

[1] Mostafa AbediniAla and Banani Roy. 2022. Facilitating Asynchronous Collaboration in Scientific Workflow Composition Using Provenance. *Proceedings of the ACM on Human-Computer Interaction* 6, EICS (June 2022), 1–26. doi:10.1145/3534520

[2] Mohannad Alhanahnah and Yazan Boshmaf. 2024. DepsRAG: Towards Agentic Reasoning and Planning for Software Dependency Management. (2024), 12 pages. doi:10.48550/ARXIV.2405.20455

[3] Jumana Almahmoud and David R. Karger. 2023. Vizdat: A Technology Probe to Understand the Space of Discussion Around Data Visualization on Reddit. *Proceedings of the ACM on Human-Computer Interaction* 7, CSCW1 (April 2023), 110:1–110:16. doi:10.1145/3579544

[4] Aritz Aranburu, Josu Cotillas, Daniel Justel, Manuel Contero, and Jorge D. Camba. 2022. How Does the Modeling Strategy Influence Design Optimization and the Automatic Generation of Parametric Geometry Variations? *Computer-Aided Design* 151 (Oct. 2022), 103364. doi:10.1016/j.cad.2022.103364

[5] Chukwuma M. Asuzu, Kathy Cheng, and Alison Olechowski. 2024. The Personas of Cloud CAD Collaboration: A Case Study of a Team of CAD Professionals. *IEEE Transactions on Engineering Management* 71 (2024), 11225–11237. doi:10.1109/TEM.2024.3409178 Conference Name: IEEE Transactions on Engineering Management.

[6] Carliss Y. Baldwin and Kim B. Clark. 1999. *Design Rules: The Power of Modularity Volume 1.* MIT Press, Cambridge, MA, USA.

[7] Jakob Bardram. 1998. Collaboration, Coordination and Computer Support: An Activity Theoretical Approach to the Design of Computer Supported Cooperative Work. Ph.D. Thesis. *DAIMI Report Series* 27, 533 (May 1998), 264 pages. doi:10.7146/dpb.v27i533.7062 Number: 533.

[8] Daniele Bifolco, Sabato Nocera, Simone Romano, Massimiliano Di Penta, Rita Francese, and Giuseppe Scanniello. 2024. On the Accuracy of GitHub's Dependency Graph. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering (EASE 2024)*. ACM, Salerno, Italy, 242–251. doi:10.1145/3661167.3661175

[9] Christopher Bogart, Christian Kästner, James Herbsleb, and Ferdian Thung. 2016. How to break an API: cost negotiation and community values in three software ecosystems. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'16)*. ACM, Seattle, WA, USA, 109–120. doi:10.1145/2950290.2950325

[10] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* 3, 2 (Jan. 2006), 77–101. doi:10.1191/1478088706qp063oa

[11] Virginia Braun and Victoria Clarke. 2021. *Thematic analysis.* SAGE Publications, London, England.

[12] David Briggs. 2012. Establish digital product development (dpd) low end viewer (lev) and archival standard for 787 project. In *Collaboration & Interoperability Congress (CIC)*. 3D CIC, Bend, OR, USA, 1 pages.

[13] Aline Brito, Marco Tulio Valente, Laerte Xavier, and Andre Hora. 2019. You broke my code: understanding the motivations for breaking changes in APIs. *Empirical Software Engineering* 25, 2 (Nov. 2019), 1458–1492. doi:10.1007/s10664-019-09756-z

[14] Colin Brown, Hamed Alhoori, and David Koop. 2023. Facilitating Dependency Exploration in Computational Notebooks. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics (SIGMOD/PODS '23)*. ACM, Seattle, WA, USA, 1-7 pages. doi:10.1145/3597465.3605222

[15] Jorge D. Camba, Manuel Contero, and Pedro Company. 2016. Parametric CAD modeling: An analysis of strategies for design reusability. *Computer-Aided Design* 74 (May 2016), 18–31. doi:10.1016/j.cad.2016.01.003

[16] Steve Campbell, Melanie Greenwood, Sarah Prior, Toniele Shearer, Kerrie Walkem, Sarah Young, Danielle Bywaters, and Kim Walker. 2020. Purposive sampling: complex or simple? Research case examples. *Journal of Research in Nursing* 25, 8 (June 2020), 652–661. doi:10.1177/1744987120927206

[17] Dan Cascaval, Rastislav Bodik, and Adriana Schulz. 2023. A Lineage-Based Referencing DSL for Computer-Aided Design. *Proc. ACM Program. Lang.* 7, PLDI (June 2023), 109:76–109:99. doi:10.1145/3591223

[18] Marcelo Cataldo, James D. Herbsleb, and Kathleen M. Carley. 2008. Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*. ACM, Kaiserslautern Germany, 2–11. doi:10.1145/1414004.1414008

[19] Marcelo Cataldo, Patrick A. Wagstrom, James D. Herbsleb, and Kathleen M. Carley. 2006. Identification of coordination requirements: implications for the Design of collaboration and awareness tools. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work - CSCW '06*. ACM Press, Banff, Alberta, Canada, 353. doi:10.1145/1180875.1180929

[20] Xiang Chen, Shuming Gao, Youdong Yang, and Shuting Zhang. 2012. Multi-level assembly model for top-down design of mechanical products. *Computer-Aided Design* 44, 10 (Oct. 2012), 1033–1048. doi:10.1016/j.cad.2010.12.008

[21] Kathy Cheng, Phil Cuvin, Alison Olechowski, and Shurui Zhou. 2023. User Perspectives on Branching in Computer-Aided Design. *Proceedings of the ACM on Human-Computer Interaction* 7, CSCW2 (Oct. 2023), 371:1–371:30. doi:10.1145/3610220

[22] Kathy Cheng, Michal K. Davis, Xiyue Zhang, Shurui Zhou, and Alison Olechowski. 2023. In the Age of Collaboration, the Computer-Aided Design Ecosystem is Behind: An Interview Study of Distributed CAD Practice. *Proceedings of the ACM on Human-Computer Interaction* 7, CSCW1 (April 2023), 1–29. doi:10.1145/3579613

[23] Kathy Cheng and Alison Olechowski. 2024. Analysis of Collaborative Assembly in Multi-User Computer-Aided Design. *Journal of Mechanical Design* 146, 3 (March 2024), 12. doi:10.1115/1.4063759

[24] Kathy Cheng, Shurui Zhou, and Alison Olechowski. 2024. "A Lot of Moving Parts": A Case Study of Open-Source Hardware Design Collaboration in the Thingiverse Community. *Proceedings of the ACM on Human-Computer Interaction* 8, CSCW2 (Nov. 2024), 29 pages. doi:10.1145/3687008

[25] Hojin Cho, Kwanwoo Lee, and Kyo C. Kang. 2008. Feature Relation and Dependency Management: An Aspect-Oriented Approach. In *2008 12th International Software Product Line Conference*. IEEE, Limerick, Ireland, 3–11. doi:10.1109/splc.2008.23

[26] Dexin Chu, Xuening Chu, Yupeng Li, Guolin Lyu, and Deyi Xue. 2016. A multi-skeleton modelling approach based on top-down design and modular product design for development of complex product layouts. *Journal of Engineering Design* 27, 10 (Oct. 2016), 725–750. doi:10.1080/09544828.2016.1227428

[27] Claudio Ciaccioli, Anna Eva Morabito, and University of Salento, Department of Engineering for Innovation, Lecce, Italy. 2021. An investigation on skeleton-based top-down modelling approaches of complex industrial product. *Journal of Graphic Engineering and Design* 12, 1 (March 2021), 11–21. doi:10.24867/JGED-2021-1-011

[28] Herbert H. Clark and Susan E. Brennan. 1991. Grounding in Communication. In *Perspectives on Socially Shared Cognition*, Lauren Resnick, Levine B., M. John, Stephanie Teasley, and D. (Eds.). American Psychological Association, Washington, D.C., DC, 13–1991.

[29] Victoria Clarke and Virginia Braun. 2013. *Successful qualitative research.* SAGE Publications, London, England.

[30] Melvin E Conway. 1968. How do committees invent. *Datamation* 14, 4 (1968), 28–31.

[31] Juliet Corbin and Anselm Strauss. 2008. *Basics of Qualitative Research (3rd ed.): Techniques and Procedures for Developing Grounded Theory.* SAGE Publications, Inc., Thousand Oaks, California. doi:10.4135/9781452230153

[32] Joel Cox, Eric Bouwers, Marko van Eekelen, and Joost Visser. 2015. Measuring Dependency Freshness in Software Systems. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. IEEE, Florence, Italy, 109–118. doi:10.1109/icse.2015.140

[33] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work - CSCW '12*. Association for Computing Machinery, Seattle, Washington, USA, 1277. doi:10.1145/2145204.2145396

[34] Maitraye Das, Thomas Barlow McHugh, Anne Marie Piper, and Darren Gergle. 2022. Co11ab: Augmenting Accessibility in Synchronous Collaborative Writing for People with Vision Impairments. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems (CHI '22)*. Association for Computing Machinery, New York, NY, USA, 1–18. doi:10.1145/3491102.3501918

[35] Cleidson de Souza and David Redmiles. 2008. An empirical study of software developers' management of dependencies and changes. In *2008 ACM/IEEE 30th International Conference on Software Engineering*. Association for Computing Machinery, Leipzig, Germany, 241–250. doi:10.1145/1368088.1368122 ISSN: 1558-1225.

[36] C.R.B. de Souza, D. Redmiles, G. Mark, J. Penix, and M. Sierhuis. 2003. Management of interdependencies in collaborative software development. In *2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings.* IEEE Comput. Soc, Rome, Italy, 294–303. doi:10.1109/ISESE.2003.1237990

[37] Cleidson R. de Souza, Stephen Quirk, Erik Trainer, and David F. Redmiles. 2007. Supporting collaborative software development through the visualization of socio-technical dependencies. In *Proceedings of the 2007 international ACM conference on Conference on supporting group work - GROUP '07*. ACM Press, Sanibel Island, Florida, USA, 147. doi:10.1145/1316624.1316646

[38] Cleidson R. B. de Souza and David F. Redmiles. 2011. The Awareness Network, To Whom Should I Display My Actions? And, Whose Actions Should I Monitor? *IEEE Transactions on Software Engineering* 37, 3 (May 2011), 325–340. doi:10.1109/tse.2011.19

[39] Bayram Demirkazık. 2021. Onshape Practices (Youtube Tutorials). https://www.youtube.com/@bayramdemirkazk1618. [Accessed 08-04-2025].

[40] Jens Dietrich, David Pearce, Jacob Stringer, Amjed Tahir, and Kelly Blincoe. 2019. Dependency Versioning in the Wild. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, Montreal, QC, Canada, 349–359. doi:10.1109/msr.2019.00061

[41] Paul Dourish and Victoria Bellotti. 1992. Awareness and coordination in shared workspaces. In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work - CSCW '92*. Association for Computing Machinery, Toronto, Ontario, Canada, 107–114. doi:10.1145/143457.143468

[42] Melanie Duckert, Louise Barkhuus, and Pernille Bjørn. 2023. Collocated Distance: A Fundamental Challenge for the Design of Hybrid Work Technologies. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23)*. Association for Computing Machinery, New York, NY, USA, 1–16. doi:10.1145/3544548.3580899

[43] Steven D. Eppinger, Daniel E. Whitney, Robert P. Smith, and David A. Gebala. 1994. A model-based method for organizing tasks in product development. *Research in Engineering Design* 6, 1 (March 1994), 1–13. doi:10.1007/bf01588087

[44] Halil Erhan, Ahmed M. Abuzuraiq, Maryam Zarei, Osama AlSalman, Robert Woodbury, and John Dill. 2020. What do Design Data say About Your Model? - A Case Study on Reliability and Validity. In *Proceedings of the 25th Conference on Computer Aided Architectural Design Research in Asia (CAADRIA) (CAADRIA 2020, Vol. 1)*. CAADRIA, Bangkok, Thailand, 557–567. doi:10.52842/conf.caadria.2020.1.557

[45] Morten Esbensen and Pernille Bjørn. 2014. Routine and Standardization in Global Software Development. In *Proceedings of the 18th International Conference on Supporting Group Work*. ACM, Sanibel Island Florida USA, 12–23. doi:10.1145/2660398.2660413

[46] Gang Fan, Chengpeng Wang, Rongxin Wu, Xiao Xiao, Qingkai Shi, and Charles Zhang. 2020. Escaping dependency hell: finding build dependency errors with the unified dependency graph. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2020)*. Association for Computing Machinery, New York, NY, USA, 463–474. doi:10.1145/3395363.3397388

[47] James Farre, Helena Kleinschmidt, Jessica Sidman, Audrey St. John, Stephanie Stark, Louis Theran, and Xilin Yu. 2016. Algorithms for detecting dependencies and rigid subsystems for CAD. *Computer Aided Geometric Design* 47 (Oct. 2016), 130–149. doi:10.1016/j.cagd.2016.06.001

[48] Jessie Frazelle. 2021. A new era for mechanical CAD. *Commun. ACM* 64, 10 (Oct. 2021), 36–39. doi:10.1145/3464909

[49] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1994. *Design patterns*. Addison Wesley, Boston, MA.

[50] Rajaram Ganeshan, James Garrett, and Susan Finger. 1994. A framework for representing design intent. *Design Studies* 15, 1 (Jan. 1994), 59–84. doi:10.1016/0142-694X(94)90039-6

[51] Saskia Gilmer, Avinash Bhat, Shuvam Shah, Kevin Cherry, Jinghui Cheng, and Jin L.C. Guo. 2023. SUMMIT: Scaffolding Open Source Software Issue Discussion Through Summarization. *Proceedings of the ACM on Human-Computer Interaction* 7, CSCW2 (Oct. 2023), 297:1–297:27. doi:10.1145/3610088

[52] J. Felipe Gonzalez, Thomas Pietrzak, Audrey Girouard, and Géry Casiez. 2024. Facilitating the Parametric Definition of Geometric Properties in Programming-Based CAD. In *Proceedings of the 37th ACM Symposium on User Interface Software and Technology (UIST '24)*. ACM, Pittsburgh, PA, USA, 14. doi:10.1145/3654777.3676417 arXiv:2408.01815 [cs].

[53] James A. Gopsill, Chris Snider, and Ben J. Hicks. 2019. The emergent structures in digital engineering work: what can we learn from dynamic DSMs of near-identical systems design projects? *Design Science* 5 (2019), 29 pages. doi:10.1017/dsj.2019.20

[54] Julien Gori, Han L. Han, and Michel Beaudouin-Lafon. 2020. FileWeaver: Flexible File Management with Automatic Dependency Tracking. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. ACM, Virtual Event USA, 22–34. doi:10.1145/3379337.3415830

[55] Saul Greenberg and Carl Gutwin. 2016. Implications of We-Awareness to the Design of Distributed Groupware Tools. *Computer Supported Cooperative Work (CSCW)* 25, 4 (Oct. 2016), 279–293. doi:10.1007/s10606-016-9244-y

[56] Rebecca E. Grinter. 2003. Recomposition: Coordinating a Web of Software Dependencies. *Computer Supported Cooperative Work (CSCW)* 12, 3 (Sept. 2003), 297–327. doi:10.1023/A:1025012916465

[57] Tom Gross. 2013. Supporting Effortless Coordination: 25 Years of Awareness Research. *Computer Supported Cooperative Work (CSCW)* 22, 4–6 (June 2013), 425–474. doi:10.1007/s10606-013-9190-x

[58] Carl Gutwin and Saul Greenberg. 1999. The effects of workspace awareness support on the usability of real-time distributed groupware. *ACM Trans. Comput.-Hum. Interact.* 6, 3 (Sept. 1999), 243–281. doi:10.1145/329693.329696

[59] Carl Gutwin and Saul Greenberg. 2002. A Descriptive Framework of Workspace Awareness for Real-Time Groupware. *Computer Supported Cooperative Work (CSCW)* 11, 3 (Sept. 2002), 411–446. doi:10.1023/A:1021271517844

[60] Carl Gutwin and Saul Greenberg. 2004. *The importance of awareness for team cognition in distributed collaboration*. American Psychological Association, Washington, D.C., DC, 177–201. doi:10.1037/10690-009

[61] Carl Gutwin, Reagan Penner, and Kevin Schneider. 2004. Group awareness in distributed software development. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work - CSCW '04*. ACM Press, Chicago, Illinois, USA, 72. doi:10.1145/1031607.1031621

[62] Carl Gutwin, Mark Roseman, and Saul Greenberg. 1996. A usability study of awareness widgets in a shared workspace groupware system. In *Proceedings of the 1996 ACM Conference on Computer Supported Cooperative Work* (Boston, Massachusetts, USA) *(CSCW '96)*. Association for Computing Machinery, New York, NY, USA, 258–267. doi:10.1145/240080.240298

[63] Han L. Han, Junhang Yu, Raphael Bournet, Alexandre Ciorascu, Wendy E. Mackay, and Michel Beaudouin-Lafon. 2022. Passages: Interacting with Text Across Documents. In *CHI Conference on Human Factors in Computing Systems*. ACM, New Orleans LA USA, 1–17. doi:10.1145/3491102.3502052

[64] Runzhi He, Hao He, Yuxia Zhang, and Minghui Zhou. 2023. Automating Dependency Updates in Practice: An Exploratory Study on GitHub Dependabot. *IEEE Transactions on Software Engineering* 49, 8 (Aug. 2023), 4004–4022. doi:10.1109/TSE.2023.3278129 Conference Name: IEEE Transactions on Software Engineering.

[65] Andrew Head, Fred Hohman, Titus Barik, Steven M. Drucker, and Robert DeLine. 2019. Managing Messes in Computational Notebooks. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, Glasgow Scotland Uk, 1–12. doi:10.1145/3290605.3300500

[66] Christian Heath and Paul Luff. 1992. Collaboration and controlCrisis management and multimedia technology in London Underground Line Control Rooms. *Computer Supported Cooperative Work (CSCW)* 1, 1–2 (March 1992), 69–94. doi:10.1007/bf00752451

[67] James D. Herbsleb and Rebecca E. Grinter. 1999. Splitting the organization and integrating the code: Conway's law revisited. In *Proceedings of the 21st international conference on Software engineering*. ACM, Los Angeles California USA, 85–95. doi:10.1145/302405.302455

[68] James Hollan, Edwin Hutchins, and David Kirsh. 2000. Distributed cognition: toward a new foundation for human-computer interaction research. *ACM Transactions on Computer-Human Interaction* 7, 2 (June 2000), 174–196. doi:10.1145/353485.353487

[69] Edwin Hutchins. 1996. *Cognition in the Wild*. Bradford Books, Cambridge, MA.

[70] Felix Hähnlein, Gilbert Bernstein, and Adriana Schulz. 2024. Understanding and Supporting Debugging Workflows in CAD. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology (UIST '24)*. Association for Computing Machinery, New York, NY, USA, 1–14. doi:10.1145/3654777.3676353

[71] Mahmoud Jahanshahi, David Reid, and Audris Mockus. 2025. Beyond Dependencies: The Role of Copy-Based Reuse in Open Source Software Development. *ACM Trans. Softw. Eng. Methodol.* (Jan. 2025), 47 pages. doi:10.1145/3715907

[72] Dhanushka Jayasuriya, Valerio Terragni, Jens Dietrich, Samuel Ou, and Kelly Blincoe. 2023. Understanding Breaking Changes in the Wild. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, Seattle WA USA, 1433–1444. doi:10.1145/3597926.3598147

[73] Joel Johansson, Manuel Contero, Pedro Company, and Fredrik Elgh. 2018. Supporting connectivism in knowledge based engineering with graph theory, filtering techniques and model quality assurance. *Advanced Engineering Informatics* 38 (Oct. 2018), 252–263. doi:10.1016/j.aei.2018.07.005

[74] Dan Kane. 2022. Tech Tip: Utilizing Feature Dependencies in Onshape. https://www.onshape.com/en/resource-center/tech-tips/utilizing-feature-dependencies-in-onshape. [Accessed 25-10-2024].

[75] Pranav Khadpe, Lindy Le, Kate Nowak, Shamsi T Iqbal, and Jina Suh. 2024. DISCERN: Designing Decision Support Interfaces to Investigate the Complexities of Workplace Social Decision-Making With Line Managers. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '24)*. Association for Computing Machinery, New York, NY, USA, 1–18. doi:10.1145/3613904.3642685

[76] Andrey Kharitonov, Amro Abdalla, Abdulrahman Nahhas, Daniel Staegemann, Christian Haertel, Christian Daase, and Klaus Turowski. 2024. A Literature Survey on Pitfalls of Open-Source Dependency Management in Enterprise. In *Proceedings of the 19th International Conference on Software Technologies*. SCITEPRESS - Science and Technology Publications, Dijon, France, 15–22. doi:10.5220/0012710800003753

[77] Kimia Kiani, George Cui, Andrea Bunt, Joanna McGrenere, and Parmit K. Chilana. 2019. Beyond "One-Size-Fits-All": Understanding the Diversity in How Software Newcomers Discover and Make Use of Help Resources. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, Glasgow Scotland Uk, 1–14. doi:10.1145/3290605.3300570

[78] Tae Soo Kim, Nitesh Goyal, Jeongyeon Kim, Juho Kim, and Sungsoo Ray Hong. 2021. Supporting Collaborative Sequencing of Small Groups through Visual Awareness. *Proc. ACM Hum.-Comput. Interact.* 5, CSCW1 (April 2021), 176:1–176:29. doi:10.1145/3449250

[79] Mark Klein, Hiroki Sayama, Peyman Faratin, and Yaneer Bar-Yam. 2002. A complex systems perspective on computer-supported collaborative design technology. *Commun. ACM* 45, 11 (Nov. 2002), 27–31. doi:10.1145/581571.581589

[80] Karine Kozlova, Roham M. Sheikholeslami, Lyn Bartram, and Robert Woodbury. 2011. Graph visualization in computer-aided design: An exploration of alternative representations for GenerativeComponentsTM Symbolic View. In *Proceedings of the 16th Conference on Computer Aided Architectural Design Research in Asia (CAADRIA) (CAADRIA 2011)*. CAADRIA, Newcastle, Australia, 133–142. doi:10.52842/conf.caadria.2011.133

[81] Raula Gaikovina Kula, Daniel M. German, Ali Ouni, Takashi Ishio, and Katsuro Inoue. 2017. Do developers update their library dependencies?: An empirical study on the impact of security advisories on library migration. *Empirical Software Engineering* 23, 1 (May 2017), 384–417. doi:10.1007/s10664-017-9521-5

[82] Rodrigo Laigner, Diogo Mendonça, Alessandro Garcia, and Marcos Kalinowski. 2022. Cataloging dependency injection anti-patterns in software systems. *Journal of Systems and Software* 184 (Feb. 2022), 111125. doi:10.1016/j.jss.2021.111125

[83] Ida Larsen-Ledet, Henrik Korsgaard, and Susanne Bødker. 2020. Collaborative Writing Across Multiple Artifact Ecologies. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. ACM, Honolulu HI USA, 1–14. doi:10.1145/3313831.3376422

[84] Steffen Lehnert. 2011. A taxonomy for software change impact analysis. In *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution (ESEC/FSE'11,*

Vol. 3). ACM, Szeged, Hungary, 41–50. doi:10.1145/2024445.2024454

[85] Piritta Leinonen, Sanna Järvelä, and Päivi Häkkinen. 2005. Conceptualizing the Awareness of Collaboration: A Qualitative Study of a Global Virtual Team. *Computer Supported Cooperative Work (CSCW)* 14, 4 (Aug. 2005), 301–322. doi:10.1007/s10606-005-9002-z

[86] Xingjun Li, Yizhi Zhang, Justin Leung, Chengnian Sun, and Jian Zhao. 2023. EDAssistant: Supporting Exploratory Data Analysis in Computational Notebooks with In Situ Code Search and Recommendation. *ACM Transactions on Interactive Intelligent Systems* 13, 1 (March 2023), 1:1–1:27. doi:10.1145/3545995

[87] Lu Liu, Harm Van Essen, and Berry Eggen. 2024. Let Information Flow Into Awareness: A Design Space for Human-Like Experiences to Promote Informal Communication for Hybrid Work. In *Designing Interactive Systems Conference (DIS '24, Vol. 2)*. ACM, Copenhagen, Denmark, 1666–1680. doi:10.1145/3643834.3661549

[88] Gustavo Lopez and Luis A. Guerrero. 2017. Awareness Supporting Technologies used in Collaborative Systems: A Systematic Literature Review. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*. ACM, Portland Oregon USA, 808–820. doi:10.1145/2998181.2998281

[89] Zhaojing Luo, Sai Ho Yeung, Meihui Zhang, Kaiping Zheng, Lei Zhu, Gang Chen, Feiyi Fan, Qian Lin, Kee Yuan Ngiam, and Beng Chin Ooi. 2021. MLCask: Efficient Management of Component Evolution in Collaborative Data Analytics Pipelines. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, Chania, Greece, 1655–1666. doi:10.1109/icde51399.2021.00146

[90] Kurt E. Madsen. 2009. Collaboration Strategies for Distributed Teams: A Case Study of CAD Systems Integration. In *2009 Fourth International Conference on Systems*. IEEE, Gosier, France, 222–227. doi:10.1109/icons.2009.46

[91] Kathleen Maher, Bill Gordon, and Chris Turner. 2017. *CAD in the Cloud - Market Trends 2017 Report.* Technical Report. Jon Peddie Research and Business Advantage.

[92] Thomas W. Malone and Kevin Crowston. 1990. What is coordination theory and how can it help design cooperative work systems?. In *Proceedings of the 1990 ACM conference on Computer-supported cooperative work - CSCW '90*. ACM Press, Los Angeles, California, United States, 357–370. doi:10.1145/99332.99367

[93] Thomas W. Malone and Kevin Crowston. 1994. The interdisciplinary study of coordination. *Comput. Surveys* 26, 1 (March 1994), 87–119. doi:10.1145/174666.174668

[94] Kirsti Malterud, Volkert Dirk Siersma, and Ann Dorrit Guassora. 2016. Sample Size in Qualitative Interview Studies: Guided by Information Power. *Qualitative Health Research* 26, 13 (July 2016), 1753–1760. doi:10.1177/1049732315617444

[95] Marcio Jose Mantau and Fabiane Barreto Vavassori Benitti. 2022. Awareness Support in Collaborative System: Reviewing Last 10 Years of CSCW Research. In *2022 IEEE 25th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, Vol. 3. IEEE, Hangzhou, China, 564–569. doi:10.1109/cscwd54268.2022.9776091

[96] Yaoli Mao, Dakuo Wang, Michael Muller, Kush R. Varshney, Ioana Baldini, Casey Dugan, and Aleksandra Mojsilović. 2019. How Data Scientists Work Together With Domain Experts in Scientific Collaborations: To Find The Right Answer Or To Ask The Right Question? *Proceedings of the ACM on Human-Computer Interaction* 3, GROUP (Dec. 2019), 237:1–237:23. doi:10.1145/3361118

[97] Maxim Marchenko, Bernd-Arno Behrens, Gregor Wrobel, Robert Scheffler, and Matthias Pleßow. 2011. A New Method of Visualization and Documentation of Parametric Information of 3D CAD Models. *Computer-Aided Design and Applications* 8, 3 (Jan. 2011), 435–448. doi:10.3722/cadaps.2011.435-448

[98] Robert C Martin and Micah Martin. 2006. *Agile Principles, Patterns, and Practices in C#.* Prentice Hall, Philadelphia, PA.

[99] Mahmoud Masmoudi, Patrice Leclaire, Marc Zolghadri, and Mohamed Haddar. 2015. DEPENDENCY IDENTIFICATION FOR ENGINEERING CHANGE MANAGEMENT (ECM): AN EXAMPLE OF COMPUTER-AIDED DESIGN (CAD)-BASED APPROACH. In *the 20th International Conference on Engineering Design (ICED 15) (Organisation and Management, Vol. 3)*. The Design Society, Milan, Italy, 199–208. https://hal.science/hal-01340960

[100] Justin Matejka, Tovi Grossman, and George Fitzmaurice. 2011. IP-QAT: in-product questions, answers, & tips. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. Association for Computing Machinery, Santa Barbara, California USA, 175–184. doi:10.1145/2047196.2047218

[101] Peter Meltzer, Joseph G. Lambourne, and Daniele Grandi. 2024. What's in a Name? Evaluating Assembly-Part Semantic Knowledge in Language Models Through User-Provided Names in Computer Aided Design Files. *Journal of Computing and Information Science in Engineering* 24, 011002 (Jan. 2024), 11. doi:10.1115/1.4062454

[102] Courtney Miller, Mahmoud Jahanshahi, Audris Mockus, Bogdan Vasilescu, and Christian Kästner. 2025. Understanding the Response to Open-Source Dependency Abandonment in the npm Ecosystem. In *Proceedings of the 47th International Conference on Software Engineering*. IEEE, Ottawa, Canada, 13. https://www.cs.cmu.edu/~ckaestne/pdf/icse25_abandonment.pdf

[103] Jasper O'Leary, Holger Winnemöller, Wilmot Li, Mira Dontcheva, and Morgan Dixon. 2018. Charrette: Supporting In-Person Discussions around Iterations in User Interface Design. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, Montreal QC Canada, 1–11. doi:10.1145/3173574.3174109

[104] Abrehet M. Omer and Alexander Schill. 2011. Automatic Management of Cyclic Dependency among Web Services. In *2011 14th IEEE International Conference on Computational Science and Engineering*, Vol. 17. IEEE, Dalian, China, 44–51. doi:10.1109/cse.2011.22

[105] M.Z. Ouertani, S. Baïna, L. Gzara, and G. Morel. 2011. Traceability and management of dispersed product knowledge during design and manufacturing. *Computer-Aided Design* 43, 5 (May 2011), 546–562. doi:10.1016/j.cad.2010.03.006

[106] Sami Paavola and Reijo Miettinen. 2019. Dynamics of Design Collaboration: BIM Models as Intermediary Digital Objects. *Computer Supported Cooperative Work (CSCW)* 28, 1 (April 2019), 1–23. doi:10.1007/s10606-018-9306-4

[107] Gerhard Pahl, Wolfgang Beitz, Jörg Feldhusen, and Karl-Heinrich Grote. 2007. *Engineering Design: A Systematic Approach* (3 ed.). Springer London, London. doi:10.1007/978-1-84628-319-2

[108] Zhiyi Pan, Xin Wang, Rumin Teng, and Xuyang Cao. 2016. Computer-aided design-while-engineering technology in top-down modeling of mechanical product. *Computers in Industry* 75 (Jan. 2016), 151–161. doi:10.1016/j.compind.2015.05.004

[109] A. Podgurski and L. Clarke. 1989. The implications of program dependencies for software testing, debugging, and maintenance. *ACM SIGSOFT Software Engineering Notes* 14, 8 (Dec. 1989), 168–178. doi:10.1145/75309.75328

[110] Gede Artha Azriadi Prana, Abhishek Sharma, Lwin Khin Shar, Darius Foo, Andrew E. Santosa, Asankhaya Sharma, and David Lo. 2021. Out of sight, out of mind? How vulnerable dependencies affect open-source projects. *Empirical Software Engineering* 26, 4 (April 2021), 34 pages. doi:10.1007/s10664-021-09959-3

[111] Ahsan Qamar, Christiaan J. J. Paredis, Jan Wikander, and Carl During. 2012. Dependency Modeling and Model Management in Mechatronic Design. *Journal of Computing and Information Science in Engineering* 12, 041009 (Dec. 2012), 10 pages. doi:10.1115/1.4007986

[112] B.F. Robertson and D.F. Radcliffe. 2009. Impact of CAD tools on creative problem solving in engineering design. *Computer-Aided Design* 41, 3 (March 2009), 136–146. doi:10.1016/j.cad.2008.06.007

[113] Matt Rohr. 2022. Tech Tip: Creating Master Sketches with Onshape's Derived Feature — onshape.com. https://www.onshape.com/en/resource-center/tech-tips/creating-master-sketches-with-onshapes-derived-feature. [Accessed 25-10-2024].

[114] Benjamin Rombaut, Filipe R. Cogo, Bram Adams, and Ahmed E. Hassan. 2023. There's no Such Thing as a Free Lunch: Lessons Learned from Exploring the Overhead Introduced by the Greenkeeper Dependency Bot in Npm. *ACM Transactions on Software Engineering and Methodology* 32, 1 (Jan. 2023), 1–40. doi:10.1145/3522587

[115] Larri Ann Rosser and Zakaria Ouzzif. 2021. Technical Debt in Hardware Systems and Elements. In *2021 IEEE Aerospace Conference (50100)*. IEEE, Big Sky, MT, USA, 1–10. doi:10.1109/AERO50100.2021.9438332 ISSN: 1095-323X.

[116] P. Rosso, J. Gopsill, S. C. Burgess, and B. Hicks. 2022. Does CAD Smell Like Code? A Mapping Between Violation of Object Oriented Programming Design Principles and Computer Aided Design Modelling. *Proceedings of the Design Society* 2 (May 2022), 1737–1746. doi:10.1017/pds.2022.176

[117] Johnny Saldaña. 2012. *The Coding Manual for Qualitative Researchers.* SAGE Publications, London, England.

[118] Anita Sarma, Larry Maccherone, Patrick Wagstrom, and James Herbsleb. 2009. Tesseract: Interactive visual exploration of socio-technical relationships in software development. In *2009 IEEE 31st International Conference on Software Engineering*. IEEE, Vancouver, Canada, 23–33. doi:10.1109/ICSE.2009.5070505 ISSN: 1558-1225.

[119] A. Sarma, D. F. Redmiles, and A. van der Hoek. 2012. Palantir: Early Detection of Development Conflicts Arising from Parallel Code Changes. *IEEE Transactions on Software Engineering* 38, 4 (July 2012), 889–908. doi:10.1109/TSE.2011.64

[120] Beau G. Schelble, Christopher Flathmann, Geoff Musick, Nathan J. McNeese, and Guo Freeman. 2022. I See You: Examining the Role of Spatial Information in Human-Agent Teams. *Proc. ACM Hum.-Comput. Interact.* 6, CSCW2 (Nov. 2022), 374:1–374:27. doi:10.1145/3555099

[121] Kjeld Schmidt. 1994. The organization of cooperative work: beyond the "Leviathan" conception of the organization of cooperative work. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work - CSCW '94*. ACM Press, Chapel Hill, North Carolina, United States, 101–112. doi:10.1145/192844.192883

[122] Kjeld Schmidt. 2002. The Problem with 'Awareness': Introductory Remarks on 'Awareness in CSCW'. *Computer Supported Cooperative Work* 11 (Sept. 2002), 285–298. doi:10.1023/A:1021272909573

[123] Kjeld Schmidt. 2011. *Cooperative Work and Coordinative Practices: Contributions to the Conceptual Foundations of Computer-Supported Cooperative Work (CSCW).* Springer London, London. doi:10.1007/978-1-84800-068-1

[124] Kjeld Schmidt and Liam Bannon. 1996. Taking CSCW Seriously: Supporting Articulation Work. *Computer Supported Cooperative Work (CSCW)* 1 (1996), 7–40. doi:10.1016/B978-0-12-415040-9.50118-4

[125] Michael Sedlmair, Petra Isenberg, Dominikus Baur, Michael Mauerer, Christian Pigorsch, and Andreas Butz. 2011. Cardiogram: visual analytics for automotive engineers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vancouver, BC, Canada) *(CHI '11)*. Association for Computing Machinery, New York, NY, USA, 1727–1736. doi:10.1145/1978942.1979194

[126] Xiaohu Song, Ying Wang, Xiao Cheng, Guangtai Liang, Qianxiang Wang, and Zhiliang Zhu. 2024. Efficiently Trimming the Fat: Streamlining Software Dependencies with Java Reflection and Dependency Analysis. In *Proceedings of the*

IEEE/ACM 46th International Conference on Software Engineering (ICSE '24). Association for Computing Machinery, New York, NY, USA, 1–12. doi:10.1145/3597503.3639123

[127] Manuel E. Sosa, Steven D. Eppinger, and Craig M. Rowles. 2004. The Misalignment of Product Architecture and Organizational Structure in Complex Product Development. *Management Science* 50, 12 (Dec. 2004), 1674–1689. doi:10.1287/mnsc.1040.0289 Publisher: INFORMS.

[128] Christian Stark. 2024. Classic VW Engine and Gearbox. https://onshape.com/documents/a6f221da86aa4e3e8d317737/w/0a1d7a6bf0f94186b94358e0/e/65e2403e8c63dce036723077?renderMode=0&uiState=671c5f54a25f7a08b136e601. [Accessed 25-10-2024].

[129] Rainer Stark. 2022. Major Technology 1: Computer Aided Design—CAD. In *Virtual Product Creation in Industry*. Springer Berlin Heidelberg, Berlin, Heidelberg, 113–138. doi:10.1007/978-3-662-64301-3_7

[130] Viktoria Stray, Nils Brede Moe, and Andreas Aasheim. 2019. Dependency Management in Large-Scale Agile: A Case Study of DevOps Teams. In *Proceedings of the 52nd Hawaii International Conference on System Sciences (HICSS)*. Hawaii International Conference on System Sciences, Grand Wailea, Hawaii, 7007–7016. doi:10.24251/hicss.2019.840

[131] Diane E. Strode. 2015. A dependency taxonomy for agile software development projects. *Information Systems Frontiers* 18, 1 (June 2015), 23–46. doi:10.1007/s10796-015-9574-1

[132] Gengyi Sun, Mehran Meidani, Sarra Habchi, Mathieu Nayrolles, and Shane Mcintosh. 2024. Code Impact Beyond Disciplinary Boundaries: Constructing a Multidisciplinary Dependency Graph and Analyzing Cross-Boundary Impact. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '24)*. ACM, Lisbon, Portugal, 122–133. doi:10.1145/3639477.3639726

[133] Ruben Taelman, Joachim Van Herwegen, Miel Vander Sande, and Ruben Verborgh. 2022. Components.js: Semantic dependency injection. *Semantic Web* 14, 1 (Nov. 2022), 135–153. doi:10.3233/sw-222945

[134] James Tam and Saul Greenberg. 2006. A framework for asynchronous change awareness in collaborative documents and workspaces. *International Journal of Human-Computer Studies* 64, 7 (July 2006), 583–598. doi:10.1016/j.ijhcs.2006.02.004

[135] Josh Tenenberg, Wolff-Michael Roth, and David Socha. 2016. From I-Awareness to We-Awareness in CSCW. *Computer Supported Cooperative Work (CSCW)* 25, 4 (Oct. 2016), 235–278. doi:10.1007/s10606-014-9215-0

[136] Christoph Treude and Margaret-Anne Storey. 2010. Awareness 2.0: staying aware of projects, developers and tasks using dashboards and feeds. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE '10*, Vol. 1. ACM Press, Cape Town, South Africa, 365. doi:10.1145/1806799.1806854

[137] Tuomo Tuikka. 2002. Remote concept design from an activity theory perspective. In *Proceedings of the 2002 ACM conference on Computer supported cooperative work*. ACM, New Orleans Louisiana USA, 186–195. doi:10.1145/587078.587105

[138] Daniel Venturini, Filipe Roseiro Cogo, Ivanilton Polato, Marco A. Gerosa, and Igor Scaliante Wiese. 2023. I Depended on You and You Broke Me: An Empirical Study of Manifesting Breaking Changes in Client Packages. *ACM Transactions on Software Engineering and Methodology* 32, 4 (May 2023), 1–26. doi:10.1145/3576037

[139] Ruotong Wang, Lin Qiu, Justin Cranshaw, and Amy X. Zhang. 2024. Meeting Bridges: Designing Information Artifacts that Bridge from Synchronous Meetings to Asynchronous Collaboration. *Proceedings of the ACM on Human-Computer Interaction* 8, CSCW1 (April 2024), 35:1–35:29. doi:10.1145/3637312

[140] Nathaniel Weinman, Steven M. Drucker, Titus Barik, and Robert DeLine. 2021. Fork It: Supporting Stateful Alternatives in Computational Notebooks. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. ACM, Yokohama Japan, 1–12. doi:10.1145/3411764.3445527

[141] Chauncey E. Wilson. 2006. Triangulation: the explicit use of multiple methods, measures, and approaches for determining core issues in product development. *Interactions* 13, 6 (Nov. 2006), 46. doi:10.1145/1167948.1167980

[142] Moritz Wittenhagen, Christian Cherek, and Jan Borchers. 2016. Chronicler: Interactive Exploration of Source Code History. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, San Jose California USA, 3522–3532. doi:10.1145/2858036.2858442

[143] Stephanie Yang, Amreen Amin Poonawala, Tian-Shun Allan Jiang, and Bertrand Schneider. 2023. Can Synchronous Code Editing and Awareness Tools Support Remote Tutoring? Effects on Learning and Teaching. *Proceedings of the ACM on Human-Computer Interaction* 7, CSCW2 (Oct. 2023), 328:1–328:30. doi:10.1145/3610177

[144] Amy X. Zhang, Michael Muller, and Dakuo Wang. 2020. How do Data Science Workers Collaborate? Roles, Workflows, and Tools. *Proceedings of the ACM on Human-Computer Interaction* 4, CSCW1 (May 2020), 1–23. doi:10.1145/3392826

Received October 2024; revised April 2025; accepted August 2025