Designing The Organizational Reuse Environment

Enabling Citizen Developers to Reuse Process Automation Artifacts

Peter A. François[1][0000-0003-1537-1026] and Ralf Plattfaut[2][0000-0002-1442-4758]

1 South Westphalia University of Applied Sciences, Lübecker Ring 2, 59494 Soest, francois.peter@fh-swf.de
2 University of Duisburg-Essen, Chair of Information Systems and Transformation Management, Universitätsstraße 2, 45141 Essen, ralf.plattfaut@icb.uni-due.de

Abstract. With the increasing proliferation of Robotic Process Automation and other low-code approaches in automating business processes come new challenges for reusing automation artifacts. Business logic is implemented in multiple technologies and developed decentrally across the organization. Especially citizen developers, who do not have extensive programming knowledge, face issues regarding reuse. Based on a thorough literature analysis and informed by Transaction Cost Theory, this research in progress identifies five challenges for reuse. Following design science research, we develop five principles that help organizations set up reuse environments that encompass both low-code and traditional development regarding automation artifacts. In addition, we describe how we plan to validate the principles in the next stage of our research.

Keywords: Reuse, Low-Code, Process Automation, Citizen Developer, Technological Rule, Design Science

1 Introduction

Organizations use various technologies to automate their business processes [1], including Robotic Process Automation (RPA) and other low- or no-code approaches. Some of these automation efforts are carried out by Citizen Developers. Citizen developers are regular employees aiding in process automation. They can apply their in-depth process knowledge during optimization and automation, however, they have little or no formal knowledge regarding professional techniques – such as reuse [2, 3]. This "lack" of knowledge impairs awareness regarding the benefits and pitfalls of reuse.

The benefits of reuse range from lower implementation cost over faster implementation time to consistent (high) artifact quality and overall reduced maintenance needs [4–6]. Simultaneously, reuse brings a set of issues or even dangers. A recent example is the reuse of Log4j, a logging utility used for logging in Java programs. The vulnerability allowed malicious entities to inject and execute code in multiple thousand software systems that (re)used the framework (see, e.g., [7]). With the advent of citizen developers, reuse – and therefore both its benefits and issues – are made available to

various business users [8]. They then face similar issues as traditional, trained developers in achieving the benefits and mitigating potential issues and dangers of reuse.

Since reuse holds such benefits and can be so hard to conduct correctly, organizations would benefit from design principles for creating reuse environments that, on the one hand, unlock the potential of reuse for all employees conducting Business Process Optimization (regular and citizen developers) and, on the other hand, shield the organization (and individuals) from potential damage from improper reuse or improper artifacts. These principles must be easily understandable, not only by regular IT developers but also by citizen developers. At the same time, the design principles must be applicable to all process automation artifacts at all stages of development, both in low-code and traditional development to harness the full potential of all reusable artifacts.

The participation of citizen developers and the use of varying technologies (traditional programming and RPA or low- and no-code technologies) bring significant challenges to business process automation. The decentral nature of citizen development and the citizen developers' lack of development knowledge can hinder reuse. Simultaneously, important and valuable business logic now exists within traditional and also low-code artifacts. This leads to the following research questions:

- What are the challenges associated with the reuse of process automation artifacts, especially considering a citizen developer involvement?
- How should organizations design organizational environments from a managerial standpoint to facilitate the reuse of distributed process automation artifacts in different technologies?

This paper is structured in 7 sections. We first describe the design science method used to answer these questions. We chose to place the background section thereafter since the concepts described there directly inform our design principles. Next, we set out to develop design principles for the organizational reuse environment. To this end, we rely on the Transaction Cost Theory (TCT). As a first rigor-cycle, we employ challenges regarding reuse based on a structured literature review. Finally, we derive design principles for the organizational reuse environment. We end with a concluding discussion outlining our future research approach to evaluate the design principles.

2 Method

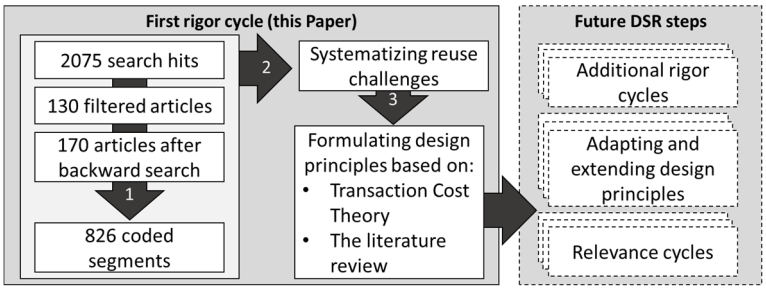

Fig. 1. Research Process

This paper follows a design science research approach [9, 10]. Figure 1 gives an overview of the research process. We first (1) conducted a thorough systematic literature review to identify existing research on automation reuse using "automation reuse" without quotation marks as the search term in the AIS eLibrary, Senior Scholar's Basket of Eight, and the proceedings of the BPM conference. The literature review consists of 2075 original hits. We chose these outlets to both get a broad overview of the field of reuse in process automation [11] and also a good idea of how the phenomena of low-code development and citizen developership impact it. We then filtered the articles, first by title, then by abstract. The filtered search resulted in 136 articles. After conducting a selective backward search, we included 170 articles in total. We then applied grounded theory [12] to the 170 selected papers, resulting in 826 coded segments. Based on these results, we applied axial coding, resulting in 89 aggregated codes. The literature review process is described in further detail in [13].

As a second step, we systematized the challenges regarding reuse based on the literature. The results of this step are described in section 5.

Based on the findings of the entire literature review, as a third step, we applied a view of TCT and developed a set of organizational design principles following van Aken [14]. According to van Aken [14], such principles are to be viewed as an "exemplar" that aids in achieving a specific state in a specific situation, meaning that adjustment is required when aims or situations diverge. For this paper, we assume that every design principle aims to maximize the potential of reuse by decreasing transaction cost while mitigating potential pitfalls. Following the work of Gregor et al. [10], who take the work of van Aken [14] into consideration, we detail the design principles with the relevant justificatory knowledge (e.g., TCT as our kernel theory).

The effectiveness of the proposed principles will be evaluated in a later stage in our research. To this extent, we plan to validate our findings in multiple organizations (so far being in contact with 7) and refine them utilizing the practitioner's feedback.

3 Background

3.1 3.1 Low-Code development and citizen developers

Low-code platforms allow developers to create software with pre-defined components that include specific functionality while using little to or even no code in the traditional sense [3]. The use of the pre-defined automation components allows the (re)use of basic functionality provided by the platform vendor, which can then be configured to fit the process and specific requirements [15]. The use of components and the quick reconfigurability can lead to high development efficiency [15]. However, low-code systems can reach a complexity similar to regular systems [16]. The quick artifact development and current knowledge of process execution improvement potentials allow quick process innovation, which is one driver for businesses to utilize such systems [17].

RPA is one technology in this category that allows the automation of digital tasks that were previously manually conducted on the graphical user interface [32].

Low-code development is usually performed in a visual editor and is said to require little to no formal training, which also allows "regular" (non-IT) employees to create

automation solutions utilizing them [15]. These so-called Citizen Developers have little to no formal programming training [3], but being close to the process execution allows them to bring a deep understanding of business processes, allowing for process and IT alignment [3, 18].

3.2 Reuse for citizen-developers

The idea of partial or full reuse of artifacts such as libraries or standard software has been a part of information systems (IS) development for a long time (see, e.g., [19, 20]). Reuse is the act of utilizing an existing artifact in new ways or utilizing (parts of) existing artifacts in new development [19]. Reuse may range from copying and pasting parts of artifacts (see, e.g., [19]) over modular or model-driven systems (see, e.g., [21, 22]) up to the reuse of partial or entire architectures [23]. Reuse is applied not only to purely technical artifacts but may also encompass specifications such as business process models (see, e.g., [24]) or knowledge (see, e.g., [25]). Due to its benefits, it is routinely applied in software engineering practice (see, e.g., [26, 27]) with some modern developing methodologies even having reuse as one of the core concepts of development (e.g., service-oriented architectures [19, 26]).

Allowing non-IT personnel to develop low-code or no-code applications has enabled companies to develop IT artifacts more quickly and to bring IS development closer to the business knowledge manifested in organizational routines [2, 8]. Traditional programming knowledge – however, is not necessary to become a citizen developer [2]. The citizen developers creating these low-code applications now face similar questions regarding reuse that professional IT workers face without having undergone the same comprehensive training. Therefore, methods and technologies to help citizen developers with reuse without having to undergo vast training are needed.

The literature on the reuse of low-code and citizen developer software is sparse. Table 1 offers a selection of the seminal literature.

Table 1. Excerpts from the seminal literature on low-code and citizen developer reuse

Source	Stance on reuse
[28]	The authors propose reuse as one measure to allow quick scaling and cost-saving in low-code projects. They describe that the existence of a large number of components reduces assembly and delivery cost in new development. However, they do not describe how to set up a reuse environment.
[16]	Reuse of low-code applications is severely limited by a lack of professional development features that aid it, such as modules (depending on the technology). This can lead to opportunistic reuse and, thus, inefficient reuse and propagating errors.
[29]	After scaling RPA operations, the case company used reusable components to enable the automation of more processes. This is supported by a central component repository allowing citizens to search for components and track their automation ideas/wishes. The authors describe that the repository eases maintenance.

Since artifacts developed by citizen developers are naturally created decentrally (by employees across the organization), finding suitable artifacts for reuse across a vast number of private artifact repositories may increase the effort for each instance of reuse.

At an extreme, this may lead to constant redevelopment due to this high effort or even biases like the "not-invented-here syndrome" (e.g., [19, 30]). A high resulting number of artifacts further complicates finding "the right" artifact [31]. Thus, artifact development in a citizen developer setting can lead to further inefficiencies in software reuse.

The described additional inefficiencies in the reuse process can make reuse unviable, especially when compared to the reduced initial or re-developing cost citizen developer systems bring (compare, e.g., [32] and [19]). In contrast, however, Lacity et al. [28] identify reusing low-code solutions as a major driver for scaling low-code solutions. Noppen et al. [33] additionally describe the need for reuse in low-code applications due to high maintenance efforts caused by a large number of similar components.

3.3 Transaction cost theory

Coarse [34, 35] argued that there are costs associated with performing transactions in a market. According to Coarse [34], a resource is traded until either A) the value of a good is maximized within the market or B) the transaction cost of performing further trades outweighs the trade's added value. The formation of organizational structures can be used to minimize transaction cost within the organization. A large availability of goods on the market, therefore, allows to satisfy more customers and leads to lower purchasing cost. However, Coarse [34] also describes a negotiation cost associated with organizations agreeing on the terms of trading a specific good.

Regarding software artifacts, the value of an artifact is not necessarily decreased when multiple (non-malicious) actors have access to the artifact. Especially in the same organization, the value of an artifact can be harnessed several times, gaining additional value from each (re)use [19]. In theory, an artifact can bring infinite value if there is a reuse opportunity that brings higher benefits than transaction costs. In practice, there is a higher cost associated with creating a reusable artifact than a traditional artifact in the first place or adapting a reusable artifact to be reusable in a new context [19].

4 Taking a transaction cost perspective on reuse

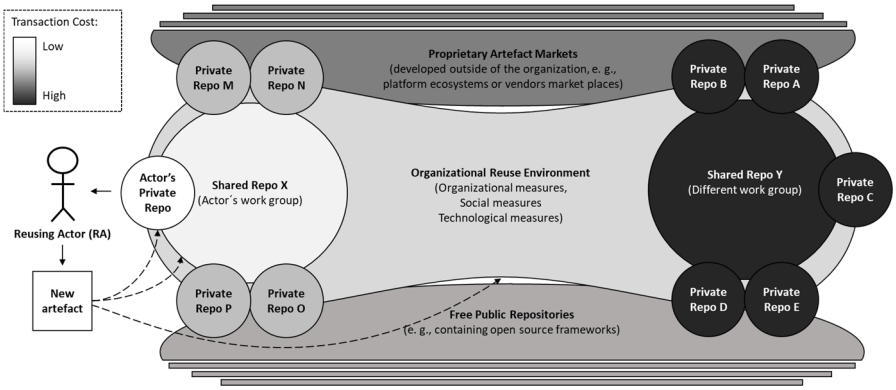

Fig. 2. Reuse artifact markets inside an organizational reuse environment

We argue that organizational reuse environments can include several 'markets' for reusable components and thus have several boundaries that can add transaction cost. Especially in citizen development (usually conducted decentrally), "silos" can form.
Figure 2 shows exemplary artifact markets existing in an exemplary organizational reuse environment and the relative associated transaction cost. Each time reuse (i.e. a transaction) is considered, the reusing actor (RA) has the option of accessing markets 'offerings', choosing and reusing one of the artifacts. These artifacts may be developed by citizen developers or "professionals" in low-code or traditional programming.
First, the RA may have access to a private repository that holds their working files. The RA should be roughly aware of the artifacts within, leading to a low transaction cost regarding reuse. In the case of a shared workgroup repository the RA has access to ("Repo X" in Figure 2), an additional element of evaluation cost applies since the repository holds artifacts created by other actors which must be examined and understood before they can be reused adding to the transaction cost.

Further personal or shared repositories (e.g., group folders of other groups) may exist within the company. The RA may not have access to these ('infinite' transaction cost). Only if the RA has some prior knowledge about which sort of artifacts may be within the repositories can the RA request access to artifacts from these repositories. Alternatively, an actor with access to these repositories could voluntarily offer the RA artifacts stored within them. For this to happen, the participants must create some form of connection. The transaction cost would still be fairly high for both artifacts, requiring direct coordination, possibly manual component transfer, and trust between both parties. Especially when the RA is a citizen developer, additional hurdles lay in understanding and evaluating the artifacts and making them compatible with, for example, RPA.

Additionally, there may be an organizational environment designed to aid reuse, including a repository of components for reuse across the organization, where the organizational measures and social constructs within the organization influence reuse.

Another possible source for reusable artifacts outside the organization is proprietary artifact markets or free public repositories. Proprietary Artifact markets allow actors to buy artifacts from vendors [18]. In the case of a trusted vendor, this transaction can be seen similarly to a transaction within the organization, plus the buying price for the artifact. Free public repositories allow access to artifacts without purchase [35]. In the case of an unknown artifact provider, an additional evaluation cost may occur. Both artifact markets and free public repositories may, however, include a large number of competing vendors offering similar components. This high number of components can then further increase the cost of discovery and evaluation.

5 The challenges of reuse

From the literature, we identify five major challenges that must be overcome to conduct reuse successfully. We will describe them in the following.

C1 Designing Reusable Artifacts: Challenges in developing reusable artifacts arise in the development of the basic functionality of the artifact, its granularity, making it reusable in several contexts, and finding all relevant contexts. First, in creating reusable

artifacts, there is the initial effort and cost of developing the reuse artifact. To calculate the cost of a specific reuse instance, this initial development cost can be divided by the number of instances the artifact is reused [19,38]. Moreover, artifacts must have the "correct granularity" (size) to be reusable [19, 37]. Smaller artifacts are easy to reuse since they have simple and easy-to-understand functionality (e.g., an RPA bot to log into a system). Larger artifacts are more challenging to reuse since they include context or process-specific requirements, yet are more valuable when reused (e.g., an RPA bot that executes several process steps) [19].

There can also be an additional development cost to make an artifact reusable (e.g., additional requirements like interfaces or effort for generalizing; see [38, 39]). The field of Software Engineering has developed various procedures to reduce reusability cost by designing artifacts for reuse, such as standardized interfaces like REST or design principles for reusable artifacts (see, e.g., [22, 40, 41]). Citizen developers, however, may not be aware of these concepts, especially as RPA enables automation via the user interface. These concepts may also not be easily applicable to low-code technologies.

Artifact creators must consider all possible contexts in which an artifact may be used during its lifetime to be able to cover requirements introduced by the use in that context [42]. Imagining all potential contexts is almost impossible at the time of artifact creation. Creating artifacts that are reusable in different contexts can be costly in itself. Adaptation (see C3 below) can be necessary when reuse in a specific context has not been considered during initial development.

C2 Accessing and Evaluating Reusable Artifacts: When deciding whether to reuse, one must weigh reuse against redevelopment [19, 38]. Thus, an actor has to find the relevant reusable artifacts (e.g., in one of many repositories), gain access to them (following formal or informal access processes), and evaluate the available artifacts regarding their fit in the desired context (e.g., by examining documentation).

Consequently, similar to the concept of coordination cost in TCT, there is the cost of finding and gaining access to reusable artifacts. Especially with small artifacts, the cost of finding them may offset the benefit of reuse [31]. Artifacts must be searched for amongst additional artifacts, making the discovery process tedious. This is especially true if artifacts are not kept in a central repository but must be retrieved from various undocumented repositories throughout the organization. Access may be complicated or unachievable when artifacts are stored in multiple repositories and exist in several versions. The decentral nature of citizen development can further this issue.

In addition, there is the problem of evaluating the artifact. Each of the artifacts found must be understood (to some extent) and evaluated regarding its value potential for this specific reuse instance. The actor has to evaluate if the artifact fits the new context. The artifact's fit for the new context (and thus the amount of adaptation required) plays an essential part in the cost of reuse and, thus, the decision between reuse and remake [38].

For citizen developers, a lack of formal training in software engineering and understanding of code may further complicate assessing components.

C3 Adapting and integrating Reusable Artifacts: When an artifact is found to be applicable for use, adaptation and integration can be necessary to enable the artifact to fit into the new environment and achieve its maximum potential. This adaptation can, for example, include making artifacts compatible or adapting an artifact to fit a different

context [19, 41]. Increasing the effort spent in Evaluating Reusable Artifacts (C2) can influence the effort in Adapting it (C3); it can aid in identifying more suitable artifacts that require less adaptation and show which adaptation will be necessary. If high adaptation rates are necessary, redevelopment can be more cost or time-efficient [19, 43]. Regardless of whether the artifacts have been adapted, it must be integrated into the new environment. The integration of artifacts can be laborious and error-inducing [4]. Therefore, the adaptation and integration must be tested, leading to further effort [19].

C4 Ensuring Security and Maintainability: The reuse of artifacts can necessitate additional maintenance. If adaptations and maintenance operations are not propagated upwards through the reuse pedigree, several versions of the same artifact will further increase the need for (now separately conducted) maintenance and allow for additional security and reliability issues (see, e.g., [7, 44]). Security and maintenance for reuse must therefore be considered in two directions: It is hard to track where a specific artifact has been reused (e.g., 'Who uses our purchase bot? We want to change it.') and vice versa, which artifacts have been reused in a specific artifact (e.g., 'Which artifacts (low-code and traditional) does our purchase bot rely on?'). The additional effort in security and maintenance leads to cost that may be small for generalized and often reused parts but larger when specialized artifacts with extensive adaptation are reused.

As citizen developers rarely have in-depth IT knowledge, performing such security assessments and maintenance can be especially difficult for them. Reusing components without sufficiently understanding them can introduce additional issues [5].

C5 Managing Reuse: Suitable governance mechanisms are critical for successful reuse [45]. Too little guidance will hinder security and maintainability (see C4) and will not allow the organization to harness the full potential of reuse [46]. It can also make reuse inefficient: too little knowledge, too few reuse mechanisms, and too few best practices will be available to the reuse actors. Too strict guidance will increase the transaction cost as reuse processes become more complex (tracking reuse, asking for permission, asking for requirements of other departments, coordinating update schedules). Introducing citizen developers can further complicate governance efforts [47].

6 Designing the reuse environment to reduce transaction cost

The challenges mentioned contribute to the cost of each reuse transaction, making reuse less viable. We argue that transaction cost is a roadblock for reuse for low-code automation. Even in traditional programming, transaction costs should be reduced as much as possible to reap the full benefits of reuse. However, since artifacts created using low-code technologies are cheaper and quicker to produce, there appears to be a reduced benefit from reusing such an artifact compared to reusing more expensive traditional artifacts. Reuse has been identified as critical to getting low-code projects up to scale [28]. In the following, we develop five design principles (DPs) for organizational reuse environments based on the abovementioned challenges, using transaction cost as the kernel theory. We deem these DP viable both for low-code and traditional development.

DP1 Enable Actors to build Reusable Artifacts. For reuse to be possible, suitable reusable artifacts must be available in the reuse market. As citizen developers are often

not formally trained in developing artifacts with reuse in mind, this is a considerable challenge due to the decentral nature of artifact development. Therefore, to enable reuse actors to build a meaningful number of reusable artifacts…:

Give all artifact creators (including citizen developers) clear guidance on how to perform reuse. Knowledge on reuse is a critical factor in being able to perform reuse effectively and efficiently [19, 43]. Since artifacts must be reusable in other departments, developers must know the concept of reuse, how it is done in the organization (standards and conventions), and how to create reusable artifacts [43, 45, 46].

Give artifact creators the knowledge required to decide which artifacts should be designed with reuse in mind and made available. Since making artifacts reusable in different contexts is associated with an additional cost [19, 42] and large repositories can lead to high search, retrieval, and evaluation cost [19, 31], RAs must decide which artifacts they should prepare for reuse and make available. To be able to make this decision, artifact creators need to be enabled to estimate the reuse potential of components. As citizen developers usually are close to the point of process knowledge creation (see, e.g. [17, 48]), this value potential will include knowledge on how processes are (or should be) conducted (see, e.g., [3]).

Offer reusable artifacts in different granularity (where possible). Finding the right granularity is a balancing act between large components that bring the most benefits and small components that can fit a variety of contexts without adaptation [37]. The benefit of reusing artifacts that are too small may be countered by the transaction cost [49]. Therefore, artifacts that are available in different granularities or are logically dividable (e.g., assembled from several different artifacts) should be offered as such. If low-code applications are used as a temporary fix (see e.g. [17]) this approach can also help transition parts of the automation to "regular" backend systems [50].

DP2 Empower Artifact Access and Evaluation. To ensure that all (potential) reuse actors are able to quickly evaluate potential reuse artifacts and deal with the number of available artifacts…:

Ensure that every (potential) reuse actor can easily find potential reuse artifacts across technologies and departments. According to Coarse [34], every market transaction requires finding potential market partners, building contact with them, negotiating the terms of the trade, conducting the trade, and evaluating the goods. Coarse states: "These operations are often extremely costly, sufficiently costly at any rate to prevent many transactions that would be carried out in a world in which the pricing system worked without cost" ([34]). In the context of reusing artifacts Rothenberger and Kulkarni [51] phase this as: "The ideal retrieval method would neither require any prior knowledge of the repository nor any informal communication among developers to find the components needed." [51] We, therefore, argue that any potential developer should have A) easy access to reusable components, with B) a standardized way of accessing them, with C) a mostly automated "contract negotiation" (i.e., access rights), preferably in D) one central artifact repository that allows searching and filtering as artifact repositories have been identified as essential for reuse (see, e.g., [19, 51, 52]).

G*ive relevant metadata on the available artifacts.* Evaluating artifacts for their reusability and need for adaptation requires a major effort. This effort increases with the number of artifacts available [31, 41]. Relevant metadata aids evaluation [53].

Name trusted and untrusted external sources of reuse components. As described above, the reuse of artifacts created outside of the organization is another viable option. In this case, trust and security become more important in artifact evaluation [41]. When artifacts are trusted, and the characteristics of the artifacts are sufficiently evident, developers can utilize the artifacts in a black box approach (not examining the inside working of the artifacts), leading to additional time savings [20]. Trusted artifacts should be marked as such. The "Not invented here syndrome" may additionally hinder reuse from outside repositories and vendors [19, 30]. We therefore suggest naming trusted artifact suppliers.

DP3 Enforce Traceability and Lifecycle Control. To allow all reuse actors to trace bugs and find security issues in their artifacts…:

Ensure that reuse artifacts can be traced throughout the entire reuse pedigree. Changes to software artifacts can impact other artifacts [53]. Therefore, changes must be tracked and (if necessary) applied to other contexts in which the artifact is used. Otherwise, security flaws and bugs remain in these contexts. Otherwise, there will be multiple versions of the same artifact, leading to increased maintenance and reuse (evaluation) cost. If security issues arise and are inherited to the new context a faulty system is reused in, these issues must be trackable throughout the entire reuse pedigree.

Observe the lifecycle of reuse artifacts. Make its status and changes known to the RAs. Each software should conform to a software lifecycle to achieve conformance, quality, and security [54]. As low code governance can be centrally or decentrally organized [47], mechanisms should be in place that allow either central or decentral lifecycle tracking in order to be able to find obsolete artifacts in the reuse pedigree and update all (possibly adapted) artifact instances. Research must still be done on how the lifecycle can be effectively tracked in such environments.

DP4 Manage Knowledge and Reuse Through Communities. To allow all reuse actors to resolve issues, communicate about reuse, build, and share knowledge…:

Connect RAs through communities of practice and allow them to motivate each other. The attitude of individual reuse actors and trust between RAs can impact the success of reuse activities [55]. If there is no bond and collegiality between RAs, reluctance to share code with or use code by others can arise [43, 56]. Communication and connection is necessary for the success of reuse [25]. Therefore, organizations should create reuse communities, forming bonds and trust between reusing developers.

Allow for the building of cross-sectional knowledge by implementing suitable knowledge management. Knowledge (on reuse or in general) can also be reused. However: "Many organisational and professional cultures reward – sometimes unconsciously – knowledge creation over knowledge reuse" [57]. Organizations should, therefore, enhance the reuse of knowledge through appropriate measures of knowledge management, like central knowledge repositories. The resulting community can also help citizen developers who are new to reuse since new reuse actors face a steep learning curve [43]. Hallikainen et al. [29] found knowledge sharing as crucial to scale low-code efforts.

Advance the reuse on knowledge by utilizing appropriate training and providing teaching material. The amount of knowledge required necessitates adequate training in

reuse [43]. Therefore, organizations should provide general training regarding reuse and their specific reuse processes, considering the needs of different developer roles.

DP5 Build Strategy and Governance. To enable all reuse actors to align their reuse procedures with the company's goals and to further reuse…:

Clarify the organization's strategy and goals regarding reuse. This includes whether reuse should be seen as a goal in itself (e.g., to reduce the number of artifacts to maintain) or if reuse should only be conducted when financially advantageous. Middle managers, in particular, can oppose efforts to set up reuse programs if a strategic view is missing. Reuse efforts are often lengthy and do not show immediate benefits in their personal performance indicators. On the contrary, efficiency may go down at first [5, 19]. A clearly defined strategy and goals will also aid developers in making reuse decisions (e.g., reuse vs. remake, which artifact to choose).

Reward actors whose artifacts are reused. Kim and Stohr [19] describe that lower-level management, when judged on their individual (and often short-term performance), may not see the benefits of reuse. Similarly, in the spirit of TCT, Coarse [34] states that when the value-adding processes of one actor (e.g., the reusing developer) diminishes value or introduces cost (e.g., maintenance cost) at a second actor, the second actor should be compensated by the first actor for the additional cost. Since a large portion of the cost for reusable artifacts arises at the department of the creating actor (development cost, continued maintenance, issue resolution), some form of compensatory mechanism is recommendable that lessens the burden of providing reusable artifacts.

Build suitable governance mechanisms to guide reuse processes yet do not over-restrict reuse. Governance mechanisms must not be too restrictive since "For the software reusability technique to be effective, it should be easier to reuse the devices than to create them from scratch." [58]. Yet, mechanisms must ensure security, compliance, and compatibility between artifacts. While there are first insights on low-code governance (e.g. [47, 59]), future research should determine suitable mechanisms for reuse.

7 Concluding discussion

In this paper, we developed five design principles for the organizational reuse environment to enable citizen developers to reuse process automation artifacts. We based these principles on common challenges in reuse and applied a transaction cost perspective.

Research has called for mechanisms that aid organization-wide reuse for both low-code and BPM [6]. The revised BPM capability framework calls for "Multi-purpose Process Design" [60]. Research on RPA has found "design[ing] bots for scalability and later migration" as a critical success factor for RPA [61]. Similarly, Syed et al. describe scaling RPA as a challenge and call for methods to aid it [62]. The proposed design principles can help operationalize these calls. With our paper, we aid in furthering reuse through a managerial approach. Researchers can build upon the principles and translate the design-oriented insights into contributions to more general theories. Furthermore, we extend the literature on process automation reuse [19]. Potentially, our insights could inform existing theories in the domains of BPM, socio-technical change or new development approaches for low-code development. For example, François et al. [63]

propose a method for anchoring reuse in the BPM lifecycle. Van Looy and Rotthier [64] describe creating a set of reusable "building blocks" to aid process automation. Průcha and Madzík [65] describe a technical approach to finding similar low-code components. Such endeavors could benefit from our proposed approach. In addition, we contribute to the Low-Code and Citizen Developer literature. For example, Bock et al. call for "methods for guiding lay developers in the use of LCPs and classical software development facilities" [15]. Our design principles can inform such methods and enable organizations to implement managerial methods to allow process automation reuse.

Practitioners can use the challenges and design principles to implement or improve process automation reuse initiatives, especially when relying on low-code or no-code development. By doing so, they can circumvent the challenges of reuse systematized above. By reducing the effort required for each act of reuse, we aid in maximizing the overall value of reuse and making reuse accessible for citizen developers.

We plan a thorough in-vivo evaluation within a recently started four-year research project with multiple private sector organizations. Here, we plan to implement the design principles in these organizations, conduct interviews with the responsible personnel, and implement an overarching reuse environment between the six participating organizations to aid cross-organizational reuse. We have conducted the first rigor cycle [9] in this paper. The challenges of traditional software reuse have been stringently researched from a theoretical and practical perspective [19]. Together with the challenges faced in scaling low-code development like RPA [62], this underlines the practical relevance of our proposed organizational design principles. Our planned evaluation will also lead to iteratively refining them based on the empirical insights [66].

In addition, more research should be carried out on citizen developer reuse. So far, it is unclear if the motivation to provide reuse components or to reuse components by others is different for Citizen Developers who may not have extensive programming knowledge and thus may be less able to assess artifacts regarding their reuse benefit.

We are aware that our research also comes with limitations. We acknowledge that the "products" (components) in the market can, in theory, be used indefinitely without losing value, which is not in the original spirit of TCT. However, the theory seems to hold for other digital goods. In addition, while we conducted a broad literature review and have implemented our design principles stringently based on this, there may be additional relevant concepts in the literature or practice.

In summary, reuse has a rich history in Software Engineering. However, citizen developers face additional challenges in reuse, such as low formal training and decentral development. Organizations should create an environment that allows for both the reuse of traditional and low-code artifacts in order to maximize value. In this paper, we developed design principles for creating organizational reuse environments. To this aim, we systematized common issues in reuse and applied a transaction cost perspective, minding to the needs and limitations of citizen developers.

Acknowledgements
This work has been developed for the project KEBAP at South Westphalia University of Applied Sciences. The project (reference number: 13FH034KX0) is partly funded by the German Federal Ministry of Education and Research (BMBF).

8 References

1. Dumas M, La Rosa M, Mendling J et al. (2018) Fundamentals of Business Process Management, 2nd ed. 2018. Springer, Berlin Heidelberg
2. Woo M (2020) The Rise of No/Low Code Software Development—No Experience Needed? Engineering 6
3. Novales A, Mancha R (2023) Fueling Digital Transformation with Citizen Developers and Low-Code Development. MISQ Executive 22
4. Apte U, Sankar CS, Thakur M et al. (1990) Reusability-Based Strategy for Development of Information Systems: Implementation Experience of a Bank. MISQ 14
5. Allen G, Parsons J (2010) Is Query Reuse Potentially Harmful? Anchoring and Adjustment in Adapting Existing Database Queries. Inf Syst Res 21:56–77
6. Beerepoot I, Di Ciccio C, Reijers HA et al. (2023) The biggest business process management problems to solve before we die. Computers in Industry 146:103837
7. Xie W, Iyer L, Simpson SJ (2022) Agile Software Development Vulnerabilities and Challenges: An Empirical Study. AMCIS 2022 Proceedings 15
8. Li Y, Huang R (2022) Participating in Citizen Development: Theory of Planned Behavior. AMCIS 2022 Proceedings 2
9. Hevner AR, March ST, Park,Jinsoo Ram,Sudha (2004) Design Science in Information Systems Research. MISQ 28
10. Gregor S, Chandra Kruse L, Seidel S (2020) Research Perspectives: The Anatomy of a Design Principle. JAIS 21
11. Gogan JL, McLaughlin M-D, Thomas D (2014) Critical Incident Technique in the Basket. ICIS 2014 Proceedings
12. Wolfswinkel JF, Furtmueller E, Wilderom CPM (2013) Using grounded theory as a method for rigorously reviewing literature. EJIS 22:45–55
13. François PA, Plattfaut R (2023) The Reuse of Business Process Automation Artefacts. LNI 337
14. van Aken JE (2004) Management Research Based on the Paradigm of the Design Sciences: The Quest for Field-Tested and Grounded Technological Rules. J. Manag. Stud. 41
15. Bock AC, Frank U (2021) Low-Code Platform. BISE 63:733–740
16. Lethbridge TC (2021) Low-Code Is Often High-Code, So We Must Design Low-Code Platforms to Enable Proper Software Engineering. In: Margaria T, Steffen B (eds) Leveraging Applications of Formal Methods, Verification and Validation, vol 13036. Springer, Cham
17. François PA, Borghoff V, Plattfaut R et al. (2022) Why Companies Use RPA: A Critical Reflection of Goals. Business Process Management. BPM 2022, LNCS 13420
18. Naqvi SAA, Zimmer MP, Syed R et al. (2023) Understanding The Socio-Technical Aspects Of Low-Code Adoption For Software Development. ECIS 2023 Research Papers 357
19. Kim Y, Stohr EA (1998) Software Reuse: Survey and Research Directions. J Manag Inf Syst 14
20. McIlroy MD (1968) Mass Produced Software Components. NATO Software Engineering Conference
21. Delgado A, Ruiz F, García-Rodríguez de Guzmán I (2018) A reference model driven Architecture linking Business Processes and Services. HICSS 2018 Proceedings:4651–4660
22. Huang JC, Henfridsson O, Liu MJ (2022) Extending digital ventures through templating. Inf Syst Res 33
23. Li S, Zhang H, Jia Z et al. (2021) Understanding and Addressing Quality Attributes of Microservices Architecture: A Systematic Literature Review. Inf Softw Technol 131

24. Becker J, Delfmann P, Knackstedt R (2007) Adaptive Reference Modeling: Integrating Configurative and Generic Adaptation Techniques for Information Models. In: Reference Modeling: Efficient Information Systems Design Through Reuse of Information Models. Physica
25. Asatiani A, Penttinen E, Rinta-Kahila T et al. (2019) Organizational Implementation of Intelligent Automation as Distributed Cognition: Six Recommendations for Managers. ICIS 2019 Proceedings
26. Baskerville R, Cavallari M, Hjort-Madsen K et al. (2005) Extensible Architectures: The Strategic Value of Service Oriented Architecture in Banking. ECIS 2005 Proceedings
27. Bjørnstad S (1994) A research programme for object-orientation. EJIS 3
28. Lacity M, Willcocks LP, Craig A (2015) Robotic Process Automation: Mature Capabilities in the Energy Sector. The Outsourcing Unit Working Research Paper Series
29. Hallikainen P, Bekkhus R, Pan S (2018) How OpusCapita Used Internal RPA Capabilities to Offer Services How OpusCapita Used Internal RPA Capabilities to Offer Services to Clients to Clients. MISQ Executive 17
30. Katz R, Allen TJ (1982) Investigating the Not-Invented-Here (NIH) syndrome: A look at the performance, tenure, and communication patterns of 50 R&D project groups. R&D Management 12
31. Nazareth DL, Rothenberger M (2006) Does the 'Golidlocks Conjecture' Apply to Software Reuse? JITTA 8
32. Lacity M, Willcocks LP (2018) Innovating in Service: The Role and Management of Automation. In: Willcocks LP, Oshri I, Kotlarsky J (eds) Dynamic innovation in outsourcing: Theories, cases and practices. Palgrave Macmillan
33. Noppen P, Beerepoot I, van de Weerd I et al. (2020) How to Keep RPA Maintainable? Business Process Management. BPM 2020, Lecture Notes in Computer Science 12168
34. Coarse R (1960) The Problem of Social Cost. JLE, 3
35. Coarse R (1937) The Nature of the Firm. Economica 4
36. Stallman RM (2002) Free software, free society: Selected essays of Richard M. Stallman. GNU Press, Boston
37. Subramanyam R, Ramasubbu N, Krishnan MS (2012) In Search of Efficient Flexibility: Effects of Software Component Granularity on Development Effort, Defects, and Customization Effort. ISR 23
38. Nazareth DL, Rothenberger MA (2004) Assessing the cost-effectiveness of software reuse: A model for planned reuse. Journal of Systems and Software 73
39. Heinrich B, Huber A, and Zimmermann S (2011) Make-And-Sell Or Buy Of Web Services A Real Option Approach. ECIS 2011 Proceedings
40. Anguswamy R, Frakes WB (2013) Reuse Design Principles. DReMeR '13 - International Workshop on Designing Reusable Components and Measuring Reusability Picture held in conjunction with the 13th International Conference on Software Reuse
41. Ren M, Lyytinen KJ (2008) Building Enterprise Architecture Agility and Sustenance with SOA. CAIS 22
42. Bērziša S, Bravos G, Gonzalez TC et al. (2015) Capability Driven Development: An Approach to Designing Digital Enterprises. BISE 57
43. Sherif K, Menon NM (2004) Managing Technology and Administration Innovations: Four Case Studies on Software Reuse. JAIS 5
44. Poulin JS, Caruso JM, Hancock DR (1993) The business case for software reuse. IBM Syst J 32
45. Joachim N, Beimborn D, Weitzel T (2013) The influence of SOA governance mechanisms on IT flexibility and service reuse. JSIS 22

46. Becker A, Widjaja T, Buxmann P (2011) Value Potentials and Challenges of Service-Oriented Architectures. BISE 3
47. Borghoff V, Plattfaut R (2022) Steering the Robots: An Investigation of IT Governance Models for Lightweight IT and Robotic Process Automation. In: Business Process Management: LNBIP, 459
48. Mirispelakotuwa I, Syed R, Wynn MT (2023) Is RPA Causing Process Knowledge Loss? Insights from RPA Experts. In: Köpke J, López-Pintado O, Plattfaut R et al. (eds) Business Process Management: Blockchain, Robotic Process Automation and Educators Forum, vol 491. Springer Nature
49. Rothenberger MA, Jain H, Sugumaran V (2017) A Platform-based Design Approach for Flexible Software Components. JITTA 18
50. Strothmann A, Schulte M (2023) Migrating from RPA to Backend Automation: An Exploratory Study. In: Business Process Management: Blockchain, Robotic Process Automation and Educators Forum. BPM 2023 Proceedings
51. Rothenberger MA, Kulkarni UR (2000) Software Reuse in Information Systems Development. AMCIS 2000 Proceedings
52. Witman PD (2006) Tracing Patterns of Large-Scale Software Reuse. AMCIS 2006 Proceedings
53. Juhrisch M, Thies G (2008) Service Management Beyond UDDI - A Design Science Approach. PACIS 2008 Proceedings
54. (2017) ISO/IEC/IEEE International Standard - Systems and software engineering - Software life cycle processes (ISO/IEC/ IEEE 12207)
55. Ryan TJ, Walter C, Alarcon G et al. (2019) The Influence of Personality on Code Reuse. HICSS 2019 Proceedings
56. Schreieck M, Wiesche M, Krcmar H (2022) Governing innovation platforms in multibusiness organisations. EJIS
57. Davenport TH (2015) Process Management for Knowledge Work. In: vom Brocke J, Rosemann M (eds) Handbook on Business Process Management 1: Introduction, Methods, and Information Systems, 2nd ed. Springer
58. Montecinos F (1996) An experience in the establishment of a software reuse culture in a real environment. AMCIS 1996 Proceedings.
59. Kedziora D, Penttinen E (2021) Governance models for robotic process automation: The case of Nordea Bank. JITTC 11
60. Kerpedzhiev GD, König MU, Röglinger M et al. (2021) An Exploration into Future Business Process Management Capabilities in View of Digitalization. BISE 63
61. Plattfaut R, Borghoff V, Godefroid ME et al. (2022) The Critical Success Factors for Robotic Process Automation. Computers in Industry 138:103646
62. Syed R, Suriadi S, Adams M et al. (2020) Robotic Process Automation: Contemporary themes and challenges. Computers in Industry 115:103162
63. François PA, Kampmann M, Plattfaut R et al. (2023) Systematically embedding automation reuse in business process management projects. LNI 340
64. van Looy A, Rotthier S (2018) Kiss the Documents! How the City of Ghent Digitizes Its Service Processes. In: vom Brocke J, Mendling J (eds) Business Process Management Cases. Springer International Publishing
65. Průcha P, Madzík P (2023) SiDiTeR: Similarity Discovering Techniques for Robotic Process Automation. In: LNBIP, vol 491. Springer Nature Switzerland, Cham,
66. Hevner AR (2007) A three cycle view of design science research. SJIS 19