Software development organizations live or die based on how effectively they generate, assimilate, reuse, and leverage their knowledge. While there is an abundance of knowledge artifacts like code modules, program charts, system designs, and requirement specifications in such organizations, this overabundance is precisely what keeps reuse an elusive silver bullet. Two decades after the widespread commotion around reuse, its results in improving software quality, efficiency, and development agility have been dismal with no panacea in sight. Innovations such as object-oriented platforms—library-like repositories of standalone code fragments in reusable objects were touted as the solution to the reuse problem. The idea was that future developers would extract code from these “knowledge repositories” to efficiently retrieve existing solutions that match design problems. Coding a complex piece of software, in theory, would become much like putting together existing Lego blocks.

In this article, we offer insights from field research in 25 software development organizations to highlight the two missing parts of the reuse knowledge management (KM) puzzle—recoding and redesign [6] (see the sidebar “How the Study was Conducted”). All organizations in our study had some sort of systematic reuse program in place. We must acknowledge that the sophistication of the reuse program differed across organizations. Some organizations were more advanced and mature in their reuse programs and systems than others. The findings we present here are those that were common across these organizations regardless of how sophisticated their reuse program was. Our findings provide interesting insights into what happens even when there is an organizational reuse mandate, regardless of the maturity of the reuse programs. The picture might be even grimmer if we consider the case of ad-hoc software reuse. Overall, the findings show that rookies are
more likely to reuse software artifacts while experts refrain from doing so. More reuse occurs within than across software development groups. We also found that transient project teams are more likely to engage in reuse than permanent development teams.

**REUSE, REDesign, AND RECODE DYNAMICS**

Figure 1 illustrates the knowledge consumption life cycle that involves three salient dynamics: reuse, redesign, and recode. Reuse is the application of existing software artifacts as is; redesign is the act of altering existing software artifacts; and recoding is the discovery of new software artifacts through construction of software code or system designs. Here, we consider each.

A commonly overlooked notion in the analysis of software development firms is the presence of multiple knowledge spaces [6]. A knowledge space refers to a logical or physical location that houses knowledge objects. Examples include a file folder, a database, a repository, and the oft-overlooked minds and rolodexes of employees. While software organizations have focused primarily on explicit knowledge spaces, ones inhabited by tacit knowledge are often overlooked. We deduced the prominence of three distinct knowledge spaces: private, quasi-public, and public. Each software engineer has a private knowledge space; it can be explicit, that is, stored in a file folder or on one’s personal computer, or tacit residing in one’s mind. Software engineers collaborate with peers and work in logical work units that can take the form of teams, groups, or departments. Private knowledge spaces emerge into quasi-public spaces over time through continued and repeated interactions with peers. These are akin to shared mental models; it is important to remember that simple additive rules do not apply here—the quasi-public space is greater than the sum of private knowledge spaces it represents. Organizations have dedicated “global” knowledge management systems that are accessible by all engineers and these represent the public knowledge space. Figure 2 illustrates these three types of knowledge spaces for Harry, in the context of his colleagues and collaborators.

Next we direct our attention to our key findings about public knowledge spaces on which software reuse traditionally focuses. Who reuses code from public knowledge spaces? In sum:

- Novices more than experts.
- Within projects more than across projects.
- Transient project teams more than permanent ones.

**Finding 1.** Experts and veterans shun reuse from public knowledge spaces. Contrary to popular opinion, we found that experts are less likely to reuse knowledge that was in the public domain. Why? Experts and veterans have rich private knowledge spaces; these are constructed over their tenure in the organization and as their skill sets have matured. As such, when faced with a new task, they first search their private knowledge space. This might include searching through logs of their past projects, parsing through past experiences, and instinctively drawing on their intuition. If the desired knowledge object is not found, they will redesign it; if a close substitute is found, they will recode it for the new task. However, if the knowledge object is not Found 2. Rookies and novices reuse from public knowledge spaces. Rookies and novices, in stark contrast to experts, are more likely to reuse knowledge in public spaces. When faced with a problem they are more likely to search through the public knowledge space. A key reason cited for this behavior is risk aversion. Novices and rookies in organizations are not risk takers—a finding backed by years of prior research on the psychology of newcomers. They find it safer to reuse existing knowledge artifacts found in the organization and redesign them if needed to complete a task. Coding from scratch is the least desirable alternative, because by doing so, they open themselves to criticism and questioning from peers. Moreover, they have more to justify to their peers regarding the need for a brand new code module or design. Novices and rookies also lack a broad level of systemic knowledge [6]. They possess knowledge in one or two niche domains, for example Java development or Oracle, this being the reason for their hiring by the organization. However, they do not possess adequate systemic knowledge, which calls for an understanding of the organization-specific sys-
tems, the nature of coupling between systems, and the nature of maintenance interdependencies. As such it is easier and more efficient for them to use existing software artifacts that have calls to other modules, take care of implementation details, and are already organization system friendly. Therefore, pervasive software modularity enhances reuse while integrality discourages it. Hence, unlike experts, the cost of searching in the global space for artifacts is less than the cost of writing novel code. Experts possess such knowledge, and hence can recode artifacts with relative ease.

Finding 3. More reuse from public knowledge spaces occurs within than across software development groups. Software engineers working in groups have a higher propensity to reuse knowledge shared within their work unit than possessed outside the work unit. Two reasons for this are salient. One is the close working relationships with peers in the group [5]. This familiarity leads them to appreciate each other’s coding style and conventions and results in greater comprehension of software artifacts created by members of the group. In comparison, software artifacts created by members outside the group are less comprehensible, and thus less reusable. It is more difficult for members in different groups to understand the intricacies and the subtle context of each other’s knowledge artifacts. Moreover, due to less frequent personal interactions, this problem is self-reinforcing. That is, less frequent interaction means more difficulty understanding another’s knowledge product, and less frequent use of another’s knowledge product reduces the potential for personal interactions. It is more efficient to reuse knowledge of a colleague who is a cubicle away rather than an office or country away, as the tacit know-how regarding the explicit knowledge object is readily available. A facet of such dynamics is the high degree of localized trust; members trust artifacts built by their team rather than those from the outside, which also reinforces the within group reuse. Somewhat surprisingly, we found that when knowledge created from members outside a group was reused, this was mainly facilitated through informal communications and networks—someone knew someone else in another group who had created a knowledge object and brought it to the group’s attention. This finding has important implications for open source development projects as well: the likelihood of reuse across open source public spaces and from such spaces in internal projects is lower.

Finding 4. Transient project teams are much more likely to exploit public knowledge spaces than are permanent teams. Logical work units such as teams and departments are common in organizations, software firms being no exception. In permanent groups, individuals work in cohesiveness for long durations to achieve common goals; ad-hoc or dynamic groups are

HOW THE STUDY WAS CONDUCTED

Over the course of two years we have interviewed over 100 software engineers in over 25 companies. We chose to conduct semi-structured interviews with software engineers; each interview lasted approximately 45 minutes. We analyzed interview transcripts for knowledge discovery and also re-interviewed engineers on matters that were high in equivocality. Hence, our findings were generated through an iterative, grounded knowledge discovery process [4]. With five companies, we even conducted onsite observations of how software engineers worked with code modules. Three companies allowed their software engineers to participate in a survey that gauged antecedents to reuse of knowledge objects. The survey was administered over the Internet, and respondents were assured confidentiality. Of the 25 companies, 12 were ranked in the Fortune 100, and the rest were medium-size IT firms. All organizations had a program in place to promote systematic software reuse. Due to confidentiality reasons, we cannot disclose the names of all 25 companies. (For further details on the study’s methodology contact Kevin DeSouza.)
shorter lived and disband after accomplishment of a single project or goal. Permanent groups are characterized by repeated interactions between a defined set of individuals. As such, over time we see convergence in individual private knowledge spaces, the key reason being reinforcement by members of the group. Consider an example: If two programmers are searching for a piece of code, at time $t$, there is a high probability they will choose different search patterns and strategies. However, at $t+n$, we will notice similarities in their search strategies, because of discussions between the two engineers as to which is a better search strategy.

Over time, a permanent group’s search strategy for knowledge converges. Convergence does not need to be perfect, and in most cases is far from perfect; however convergence is significant enough that over time, members of a group will all search similar portions of the public knowledge space for knowledge artifacts. Hence, this leads them to focus on a few areas and reinforce them in the group, lowering the likelihood of reuse over time as new knowledge found in the other sectors of the public space is ignored. In comparison, transient and dynamic work unit members are more likely to explore wider spans of the public knowledge space. Due to their interactions with a diverse set of individuals, they are more likely to discover new locations of knowledge artifacts and are hence more likely to find solutions to their problem, leading to greater reuse. This finding offers important insights into open source development projects: transient project teams from outside an open source network are likely to try reusing code and modules from such public spaces.

**Conclusion**

Reuse has long been an idea ahead of its time. As knowledge management increasingly becomes crucial to the survival of software organizations, it is time to rediscover the power of the idea. To give the idea of reuse some teeth, it is imperative that software development organizations appreciate that knowledge exploitation is a dynamic, three-pronged process that spans public, quasi-public, and private knowledge spaces. Our findings provide new insights into the reuse black-box and show the types of software project teams and environments where traditional reuse from public knowledge spaces—repositories, code libraries, traceability systems, and CASE tools—is likely to be used and where it is not.

Two recent trends in computing have brought reuse to the forefront: open source development projects such as Linux, and component-based software development. The success of open source development rests on individuals contributing code fragments, scripts, and ideas to the public knowledge space associated with the project. Our findings provide unique insights into the challenges of such projects. The most likely users of open source knowledge and code are likely to be rookies rather than experts, and temporary project teams rather than the permanent open source community directly associated with the project, although knowledge and code spillovers to other projects are less likely with such users. However, the success of open source projects hinges on experts and veteran programmers contributing to the public knowledge space—not novices reusing from it. The longer-term success of the open source model therefore requires new and different types of incentive systems (such as reputational and financial systems) to sustain high-quality knowledge contributions in the medium to long run.

Consider next the insights into component-based software that can be drawn from our findings. The success of component-based software requires potential customers reusing from a semi-public knowledge space. Our findings suggest that novices and temporary project teams are the most likely adopters of such software. It is such users and projects to which component-based software ought to be targeted.

The key takeaways: Traditional reuse works better among novice than expert developers, and in transient rather than permanent teams. As surprising as this seems, it explains why our wishful silver bullet has never had its shine.

**References**


**Kevin C. Desouza** (desouza@engagedenterprise.com) is senior advisor to The Engaged Enterprise, a strategy consulting firm, Chicago, IL, and an assistant professor at the University of Washington, Seattle, WA. **Yukika Awazu** (awazu@engagedenterprise.com) is president of The Engaged Enterprise, Chicago, IL. **Amrit Tiwana** (tiwana@iastate.edu) is an assistant professor at Iowa State University, Ames, IA.