Agile Knowledge Sharing

I. Burak Ersoy *burak.ersoy@tamucc.edu*
School of Engineering and Computing Sciences
Texas A&M University-Corpus Christi
Corpus Christi, TX 78412, USA

Ahmed M. Mahdy *ahmed.mahdy@tamucc.edu*
School of Engineering and Computing Sciences
Texas A&M University-Corpus Christi
Corpus Christi, TX 78412, USA

Abstract

In today's economy, enterprises require knowledge more than ever before. Employees are being classified through their skill set and experience, where the tacit knowledge of individuals is the key factor. The effect of knowledge hunger can be easily seen in agile software development teams. To sustain the quality permanence of software development, it is essential to transform individuals' tacit knowledge to core organizational knowledge. To achieve this goal, every software development process utilizes different knowledge sharing and creation approaches. In this paper, knowledge sharing issues are surveyed and categorized into: 1) sociological issues, 2) documentation issues, and 3) implementation issues with/without pair programming. Finally, a proposed technique, Knowledge Temple, is introduced as feasible improvement to well-known knowledge sharing problems for small agile software development teams.

Keywords: Agile Software Development, Knowledge Sharing, Knowledge Creation, Knowledge Loss, Knowledge Hoarding.

1. INTRODUCTION

Creating successful projects is the ultimate goal of software engineering. Thus, software development methodologies are introduced to overcome software development issues, such as late projects, budget issues, and faults [12]. Traditional software development methodologies, team software process (TSP) and personal software process (PSP) from the Software Engineering Institute (SEI) [5], and Agile methodologies [14] are leading software life-cycle models. Every life-cycle model offers different participation or learning activities, such as cognitive apprenticeship and knowledge repository creation routines. All those methodologies evolve around knowledge management; in fact, knowledge sharing is the major component of each.

Tacit knowledge is the experience of development, training, and/or education, which materializes in a person [2][16][31][51][60][64]. Software development is based on the tacit knowledge of the individuals. To sustain the quality permanence of software development, it is essential to transform individuals' tacit knowledge to core organizational knowledge. To achieve this goal, every software development process utilizes different knowledge sharing and creation approaches.

Traditional software development methodologies make use of extensive documentation to accomplish knowledge sharing [13][19][26][59][61]. The documentation contains the project management plan, configuration management plan, quality assurance plan, validation and verification plans, requirements specification, design description, application testing, and user documentation for the project. However, creation of those comprehensive documents is time consuming because the documents are excessively project-specific, thus, the reusability of the documents is nominal.

TSP and PSP offer self and team training through in-house and/or external educational programs [5][56]. Moreover, TSP and PSP models are also performed in academia to create industry-level software engineering for university students [25][62]. Although this training is influential, the cost of the training is high especially for small software development teams. In addition, training lets the software development team get sidetracked by gaining the knowledge because they are not able to continue project development. Thus, the productivity of the development team becomes almost zero.

The cognitive apprenticeship model presents an active participation technique between master and the apprentice. This approach is applied in class and in virtual settings in academia [24][32][43]. Its collaborative learning experience creates an authentic setting for knowledge sharing. On the other hand, the success of cognitive apprenticeship depends on the lead quality of the master. In addition, cognitive apprenticeship requires time for profitable knowledge sharing [34].

Knowledge repository creation is an active learning and developing approach [6][53][58][71]. Creating process assets increases the tacit knowledge transformation among developers. Moreover, this technique increases the reusability of externalized tacit knowledge. Yet, version management of created assets and evolving assets, which have high functionality and specificity, are the downside of knowledge repository creation.

Agile methodologies introduce two knowledge sharing approaches, which create strong enthusiasm in software engineering [18][19][33][59][61][67]. Pair programming not only allows successful knowledge sharing between pairs but also enhances the development quality. Pair rotation builds a sincere software development environment by breaking the ice between software development team members. Those two approaches are also carried out in academia as a classroom technique to facilitate peer knowledge sharing and to increase intercommunication among students [17][38][40][66]. However, those two methods lead to unequal participation and pair incompatibility.

Agile development offers a productive, flexible, and adaptive environment, where knowledge sharing limitations may arise [1][28][44][55]. The key concept of agile methodologies is creating working software via customer satisfaction and development pace [9][14][22][44]. Therefore software development teams focus more on applying the knowledge than sharing.
In this work, the main goal is to survey knowledge sharing techniques for agile software engineering.

1.1 Why Knowledge Sharing?

The problems of software development teams are:

1. knowledge loss via retirement or high turnover rates and
2. knowledge hoarding for interpersonal reasons or organizational climate.

If the organization suffers from knowledge loss and knowledge hoarding, it may mean the organization is staff-dependent. For organizational success and continuity, organizations have to be staff-independent. Being staff-independent means both knowledge loss and knowledge hoarding protected. In order to be staff-independent, organizations should share the knowledge among the development team.

2. KNOWLEDGE SHARING CHELLENGES

Knowledge is considered a principal component in developing software because software development is a people-based activity where developers' knowledge impacts the process. To achieve software complexity and quality demands, organizations need successful programmers

[8][34][42][50]. However, finding good programmers is a challenge for many small-level organizations and research institutes. The reason can be either the cost of a good programmer or the lack of desire of the programmer to become a part of a small development team [27].

Knowledge loss is a serious issue for every level of the software development team [11][29][49][68]. However, the effects on small development teams are more catastrophic than mid-level or large development teams. Most small development teams have a strong dependency on their productive developers. Therefore, losing the knowledge of productive developers means losing the development quality. Knowledge loss can be caused through retirement or high turnover rates. In particular, external turnover of skilled developers is a rising dilemma for small development teams [11][29].

Knowledge hoarding is another serious issue for software development teams [10][11][30][41][63]. Individuals want to keep their knowledge hidden for interpersonal reasons. Another reason for knowledge hoarding is the lack of organizational culture. Building a knowledge sharing climate in the workplace is a demanding business activity.

The pace of technology change is another challenge for knowledge sharing [12][47]. Developers may not find time to update their knowledge while trying to meet deadlines. They may not even get around to sharing knowledge with colleagues [4][7][15][48][70]. It also brings the knowledge creation standards to action because fast-paced technology compels explicit knowledge creation. Still, applying version control to avoid garbage knowledge creation is required [69][72].

Agile development speeds up the software development process and has high response to customer requirements and changes [3][20][36][37][52]. It provides an iterative and incremental development fashion among self-organizing and cross-functional teams. Nonetheless, agile approaches have a unique development through the means of the Agile Manifesto [21]. Accordingly, adapting to agile development is an arduous process for both developers and organizations [54].

Although pair programming can be seen as a knowledge sharing technic for agile methods, expecting programmers in a small development team to have the same level of knowledge is unrealistic [35]. There is always different levels of developers in software organizations along with their different expertise. Therefore, handling different types of developers is another challenge of knowledge sharing [13][36][65].

3. KNOWLEDGE SHARING REVIEW

In today's economy, enterprises require knowledge more than ever before. Employees are being classified through their skill set and experience, where the tacit knowledge of individuals is the key factor [8]. The effect of knowledge hunger can be easily seen in agile software development teams. Biawo-wen [10] claims that we are in the "knowledge economy era" and states the knowledge necessity for agile software development teams in three steps:

1. knowledge is the only meaningful resource,
2. companies products and services are based on the transformation of the knowledge, and
3. software employees require more knowledge management than any other business sectors.

However, implementing knowledge sharing is not an easy task for agile development teams compared to its increasing demand. We classified knowledge sharing implementation issues under three perspectives: sociological, documentation, and implementation.

3.1 Sociological Issues

Sociological perspective covers a hidden factor of knowledge sharing. In comparison with the technical side, the human side of agile development teams has been ignored for a long time. It is important to reveal the value of social structure in an agile development team in order to comprehend the development process.

Occupational stress is one of the most important problems of knowledge sharing implementation [7]. The connection between software development and agile developers relies on tacit knowledge and human creativity. Occupational stress keeps tacit knowledge and human creativity isolated in the body of an individual. Thus, the productivity and the desire of sharing knowledge decrease very dramatically. In addition, Amin, Basri, Hassan, and Rehman [7] provide the key factors of occupational stress as fear of obsolescence, individual team interactions, client interactions, work family interface, role overload, work culture, technical constraints, family support towards career, workload, and technical risk propensity.

Chau, Maurer, and Melnik [13] explore the theoretical link between agile team members as "Trust and Care." Developing the organizational and individual trust in the teams and between the teams is indispensable. Trusting increases knowledge generation, and sharing between the colleagues where caring for teammates is also created. Agile methods, such as collective code ownership, stand-up meetings, onsite customer, and pair programming, build the mutual trust and care among collaborators. Moreover, Crawford, Castro, and Monfroy [16] discuss the importance of not only trust but also freedom in order to accomplish knowledge sharing. Interactions among the members of a team can become a fact voluntarily not by an order from executives [13][16]. On the other hand, Mathew, Joseph, and Renganathan [41] suggest that financial incentive fosters the team members to share knowledge. Even more importantly, they claim all type of personalities can be influenced financially. However, the research indicates negative results.

Chua, Eze, and Goh [15] have recently developed a conceptual framework for knowledge sharing. This framework contains six hypotheses: kiasuism, subjective norm, affiliation, worker empowerment, knowledge technology, and intention to share knowledge. Kiasuism is defined as "getting the most out of every transaction and a desire to be ahead of others." Subjective norm is described as a social pressure for high performance, and affiliation is explained as the fellowship among the team members. Thus, conceptual framework recommends high level of subjective norm, affiliation, worker empowerment, use of knowledge sharing technologies, supportive attitude towards knowledge sharing, and low level of kiasuism for positive influence on knowledge sharing.

The study by Jabar, Cheah, and Sidi [30] is noteworthy in that it combines organizational factors, which are distributive, procedural and interactional justice, and individual factors, which are perceived goal and perceived reward interdependence, to build the knowledge sharing attitude in software development teams. They also argue that positive knowledge sharing attitude and subjective norm evolve the knowledge sharing behavior in organizations.

In addition to using agile methods, such as pair programming and stand-up meetings [13] and giving autonomy to software development teams [15], Law and Charron [36] introduces "Co-location" and organizing social activities with associates. Two different Co-location techniques are applied through the study, open environment and common cubicle zone. While open environment offers face-to-face interaction, common cubicle zone boosts express communication along with personal space and privacy. Birthday luncheons, game and tea parties in afternoon, and toys that break the ice between team members are available as social activities. They encourage the affinity and alliance among the team members with great synergy.

For managing one of the most popular social agile development issues, developer turnover, Rong et al. [49] present a model based on information entropy to measure the turnover risk on a software project. Information entropy theory helps to assess the uncertainty and uniformity of the turnover risk. This argument quantitatively states the catastrophe of losing the key contributor of the software team. It foresees the future turnover risk for managers to perform a precautionary knowledge sharing approach. Furthermore, Whitworth and Biddle [67] define a qualitative grounded theory based model to determine socio-psychological experiences in agile development teams. They define the agile teams as "complex adaptive socio-technical systems," which contain strong social forces. Their approach stresses the importance of agile methods to activate the knowledge sharing process.

In addition, Izquierdo-Cortazar, Robles, Ortega, and Gonzalez-Barahona [29] demonstrate a methodology to measure the quantitative impact of knowledge loss due to developer turnover. In order to quantify the knowledge loss, this study introduces "orphaned" lines of code. When a member of the agile development team leaves the software development team, his/her code becomes orphaned. Thus, the knowledge sharing process becomes insecure by the amount of orphaned lines and the project requires greater focus on software archaeology. The results of the study indicate that the use of orphaned lines evaluates the "health" of the software project and clues in managers before it is too late.

Yang and Wu [70] propose an agent-based modeling (ABM) concept to explore the knowledge sharing motivation in the agile development team. ABM is a simulation system where researcher can create, observe, and analyze the experimental personal behavior and motivation of sharing knowledge within the development team. It uncovers the team members with high knowledge and sharing behavior along with the organizational knowledge sharing climate and culture.

3.2 Documentation Issues

Another substantial perspective of knowledge sharing is the knowledge storing process. It is clearly stated in the Agile Manifesto that agile developers should value working software over comprehensive documentation [21]. However, knowledge transfer without documentation is a challenging practice. Consequently, agile teams feel documentation is necessary through varied approaches.

Analyzing the documentation approaches for different methodologies is essential for this reason. Chau, Maurer, and Melnik [13] discuss the varied documenting techniques for both Tayloristic and agile methods. Tayloristic methods require a large number of documents, which comprise all possible requirements, design, development, and management issues. On the other hand, agile methods argue "lean, mean, and just enough" documentation techniques. Additionally, agile methods introduce collective ownership that any team member can participate and alter the knowledge repository to keep it up-to-date [13]. Law and Charron [36] stress that keeping the documentation updated and define the issue as "one webmaster syndrome." Using social software development tools is a best practice to accomplish collaborative revising responsibility. As a result, agile development teams utilize "work-in-progress" documentation fashion, which requires collaborative authority.

Abbattista, Calefato, Gendarmi, and Lanubile [1] survey the literature on social software development tools. They group the tools into seven categories based on their main functionality; software configuration management, bug and issue tracking, build and release management, product and process modeling, knowledge center, communication tools, and collaborative development environments. Moreover, they argue that adequate technology support is fundamental for active knowledge sharing. Finally, their results show that collaboration is a side effect of social software development teams.

Among the social software development tools, Wiki is the most revised tool in terms of usage and features. It is a practical tool for not only small development teams but also large enterprises [22]. Sousa, Aparicio, and Costa [58] remark using Wikis is essential as an organizational knowledge sharing tool. Organizational Wiki facilitates sharing through knowledge map of the individuals, conveys tacit knowledge between team members, transforms tacit to explicit knowledge, and commutes explicit to tacit knowledge in order to sustain the sharing process.

Another documentation model assumes that utilizing an appropriate ontology to index Wikis improves the knowledge sharing process. Tang, de Boer, and van Vliet [63] introduce Semantic Wiki with a lightweight and adaptable ontology, which classifies concepts and supports knowledge retrieval. A semantic Wiki allows custom-defined indexing and acknowledges agile team members through an event-based notification system for their asynchronous knowledge request.

The study by Amescua, Bermon, Garcia, and Sanchez-Segura [6] is noteworthy in that it combines creating process asset libraries (PALs) and Wikis. Their study provides a set of guidelines to create a PAL-Wiki. The PAL-Wiki captures, codifies, and disseminates the knowledge about software agile processes and facilitates an active learning environment. The results of the study report that the PAL-Wiki is easy to learn, use, and operate in order to provide a knowledge sharing mechanism. This approach also motivates the agile software development team to explore concepts independently. Furthermore, Law and Charron [36] present using mockups as a Wiki documentation technique. They believe "a picture is worth a thousand words" and in keeping the Wiki web site as visual as possible.

An alternative visual technique proposes Unified Modeling Language (UML) usage to minimize documentation for agile development teams. Stettina, Heijstek, and Faegri [60] divide the documentation process into two perspectives: documentation as a product and documentation as a medium. The first perspective, documentation as a product, requires more textual and formal documents through the iterative development process. At the end of the development, agile development teams possess the documentation as a valued product, but team members identify the progress as "a task that needs to be done." Although it increases the quality of the product, it decreases the motivation to participate. On the contrary, the documentation as a medium perspective requires UML-based documentation artifact creation during the agile development progress. It derives team motivation, easy updates, and generalist team roles; however, it drops the sustainability of the knowledge sharing documentation in the long run.

Prause and Durdik [47] inquire about the results of a reputation mechanism to answer the documentation argument of agile development teams. According to their research, reputation is considered the driving force to make selfish individuals cooperate and participate. Moreover, reputation systems, which compute reputation scores of participants, encourage rating the available documentation. Survey results of the study show 85% of experts believe the reputation system is promising and will have a positive effect on agile documentation via "pro-social" behavior of the agile development team members.

3.3 Implementation Issues

Implementing knowledge sharing for agile development teams is more troublesome due to the nature of the agile process. Pair programming is one of the most respected agile development techniques, with influential knowledge sharing as a side effect. We surveyed knowledge sharing implementation methods with and without pair programming perspectives.

3.3.1 Implementation without Pair Programming

Chatti, Schroeder, and Jarke [12] examine the relation between knowledge management and technology-enhanced learning to propose the Learning as a Network (LaaN) theory. The knowledge vision of the system is a personal network and the learning concept is a knowledge ecological approach. LaaN allows learning through the continuous creation of a personal knowledge network.

Huang and Sun [26] introduce a mobile agent system to accomplish establishing, operating, and disassembling management of the virtual enterprise. The virtual alliance of the knowledge management system relies on six agents. The agents are named as communication control, lifecycle management, knowledge processing, establishing management, operation management, and disassembling management. They analyze, process, store, and share the knowledge among the whole enterprise in an automated fashion.

Zhang, Tang, Liu, and You [72] declare another multi agent knowledge sharing architecture based on the Internet and varied knowledge inventories. Domain knowledge, organization knowledge, process knowledge, distributed case base, ontology, user interface, workflow, and toolset agents are utilized to build a cooperative design. Share Knowledge Space and Communication Control Center operate the knowledge exchange and interaction during the whole development time. Moreover, agents have a knowledge sharing mechanism through application, mind, message, and communication layers.

Jiang, Liu, and Cui [31] consider a five layered knowledge sharing framework in the interest of organizational knowledge management. The system combines knowledge management strategy, organizational learning, and business process reengineering theories. Basic construction, system management, content management, knowledge management, and theory layers originate the framework of the knowledge management system.

Tang, de Boer, and van Vliet [63] present a knowledge sharing perspective with roadmapping process in order to succeed in timely knowledge traffic. Their research indicates that the inadequacy of knowledge sharing is not the knowledge creation but the effective knowledge transferring between the team members. A collaborative knowledge inventory, Semantic Wiki, facilitates the communication capability. This roadmapping process, with an indexed pattern, provides a direct knowledge search ability and notification system for formerly-demanded knowledge.

Some discussions of the role of applying agile methodologies can be found in Landaeta, Viscardi, and Tolk [35]. Through the strategic management of agile projects, software development teams can share knowledge across the projects and create an organizational learning culture among the agile team members. The extended agile methodology offers mentoring, coaching, and staffing project teams with members of other projects and participation in both multi-project reviews and retrospectives. Therefore, team members can share knowledge in crossed fashion via parallel projects and active team members.

Kavitha and Ahmed [33] propose another knowledge sharing framework through a collaborative environment connected by internet and intranet. The approach facilitates an incremental organizational learning using knowledge enablers. Communities of Practice (CoPs), questionnaire responses, email archives, work notes, informal knowledge sharing sessions, voluntary contributions, project learnings, and discussion forums are the knowledge enablers for the informal knowledge sharing framework. The experience recorder, idea map, and forums capture the tacit knowledge from knowledge enablers and structure the knowledge repository using frequently asked questions and lessons learned retrieval mechanisms.

3.3.2 Implementation with Pair Programming

Pair programming is an agile software development technique that allows two programmers to collaboratively design, code, and test side-by-side [16][37][65]. Each pair has a particular role, which is either driver or navigator. The driver is the one that produces the code or design and performs test cases. The navigator actively determines the tactical and strategic weaknesses and continuously helps the pair to improve development. Through the pairs' determination, the driver and navigator switch roles and carry forward the development routine [45]. Moreover, changing partners between other pairs, pair rotation, is highly recommended to achieve an efficient knowledge sharing [33].

Sanders [52] provides pair programming adaptation experiences and pairing issues as an Agile Coach. There are two essential concerns to switch the organizational development technique from solo to pair programming. First, some agile leads believe pair programming doubles the person hours to complete a task. Second, agile development team members are not interested in pairing with others. According to Sanders [52] small changes, such as buying big monitors for pairs and arranging designated area for pair programming, can effect the motivation of agile team members. However, the privileges for pair programming teams should be influential, efficient, and easy to implement. Increased programming motivation and code coverage in every sprint are the results of pair programming migration.

Chau, Maurer, and Melnik [13] describe pair programming as an informal training. Compared to Tayloristic methods with formal training, agile methodologies have informal approaches, such as pair programming and pair rotation. System knowledge, coding convention, design practices, and tool usage tricks are tacit knowledge instances that participants can easily share through pair programming. More often then not, the tacit knowledge instances are neither documented nor a part of the formal training. Chau, Maurer, and Melnik [13] also present the pair programming drawbacks, such as pair incompatibility and increased training cost through particular circumstances compared to formal training options.

Ganis, Maximilien, and Rivera [22] report an "Agile@IBM" survey from 2008 and 2009 across all of IBM. Agile@IBM covers the key agile practices, such as sustainable pace, whole team planning, continuous integration, daily scrums, and pair programming. The results of the survey indicate significant improvements in credibility of blooming agile practices, which are sustainable pace (55.1%), whole team planning (44.8%), continuous integration (34.5%), and daily scrums (26.2%). Nevertheless, there is a huge amount of credibility decrease (37.9%) in usage of pair programming.

Law and Charron [36] demonstrate a knowledge sharing approach, which unites pair programming, co-location, daily status meetings, and minimal documentation. To solve the pair scheduling issue of pair programming, team members do code inspection in addition to pair programming. Examining the source code for code alterations and error discovery are the core part of the code inspection. Pair programming and code inspection mixture make both knowledge sharing and cross knowledge training possible for agile team members. Yet, the experiment results denote time-sharing penalties, motivation loss for novice team members, and a shift in focus from pair programming to deadline-driven task development.

Srikanth, Williams, Wiebe, Miller, and Balik [59] examine the advantages and disadvantages of pair programming and pair rotation on undergraduate level students. Their results are vital for software development teams, which have junior level team members. Enhanced quality, teamwork, communication, retention, confidence, comprehension, and learning are the pair programming advantages for agile development pairs. However, pair programming presents schedule issues, pair incompatibility and unequal participation. The bottom-line concern of pair programming implementation is the skill level of pairs. A higher skill level gap produces a lower level job satisfaction and productivity both for knowledge sharing and development processes. Furthermore, researchers report the pair rotation advantages as gained knowledge of team members and elevated desire to pair with new team members. On the other hand, pair rotation kindles partner compatibility, motivation decrease due to good partner loss, and programming fashion re-adjustment in consequence of new partner.

Poff [46] observes the organizational learning effect of pair programming on newly-hired team members in an industrial setting. The study pairs the junior level team members, requires voluntary mentoring from experienced team members, and aims to facilitate the technical and environmental training of the newcomers. The experiment shows the new-hired pairs require more man-hour and more mentoring than new-hired solo programmers. However, the novice-novice collaboration increases overall productivity, allows more accurate project planning, partially hastens technical and environmental knowledge sharing, and decreases programming defects compared to newly-hired solo programmers.

Giri and Dewangan [23] introduce an improved version of IBM's Programming Aptitude Tests (PATs) for pair programers. Through PAT scores, organizations can determine programming abilities and potential of newly hired programmers. Researchers take advantage of the PAT scores for team building and pairing agile development team members. Using total effort/time measurement with PAT scores, Giri and Dewangan [23] calculate the "Relative Effort Afforded by Pairs (REAP)" value as well. REAP values indicate one of the five different conditions: total development time of pairs is less than individual, pairs and individuals have the same total development time, pairs require more total man-hours but develop faster than individual, elapsed development time for pairs and individuals is almost the same, or elapsed development time for pairs is longer than individuals.

Lui and Chan [39] present a Software Process Fusion (SPF), which combines both solo and pair programming. The approach divides the software processes as "Recipient" and "Donor." Agile team members pair for Recipient Processes and work individually for Donor Processes. Thus, pairing motivation never decrease via repeating the same task again. Team members decide the transfer conditions for pairing or splitting. Researchers use the transfer conditions value to calculate a Software Fusion Ratio (SFR). SFR shows the efficiency and productivity of SPF.

Another study considers the effects of pair programming at the development team level based on productivity, defects, design quality, knowledge transfer, and enjoyment of work. Vanhanen and Lassenius [65] report a productivity difference between pair and solo programming. The productivity decreases while pairs are under the learning curve. However, the productivity level is almost the same for both pair and solo programming practices after the learning period. Although pair programmers code with less defects, their final product contains more issues because of the system testing oversight. Pairs excessively depend on the peer review process of pair programming, which causes over-reliance in the testing phase. Pair programming enables knowledge transfer between peers and team; however, the pair programming abates development teams' working enthusiasm. Moreover, Vanhanen and Lassenius [65] emphasize that the task complexity does not affect the effort differences between solo and pair programming.

Sillitti, Succi, and Vlasenko [57] examine the impact of pair programming via the developer's focus. This study tracks the usage of nine popular applications: Microsoft Visual Studio, Browser, Microsoft Outlook, Microsoft Office Word, Microsoft Office Excel, Microsoft Management Console, Microsoft Windows Explorer, Microsoft Messenger, and Remote Desktop. Solo programmers constantly utilize the Internet for information retrieval. Browser usage decreases from 9% to 6% with pair programming. Microsoft Outlook, Microsoft Messenger and Remote Desktop usage also decreases because pairs create a robust communication between each other. In addition, programming motivation increase via pairing pressure. Microsoft Visual Studio utilization increases from 34% to 64% with pair programming [39].

4. A PROPOSED TECHNIQUE

Small software development teams suffer from knowledge lost due to miscellaneous reasons. Therefore, surveying knowledge sharing issues through sociological, documentation, and implementation perspectives is essential to reveal the real motive. Agile practices offer state-of-art solutions for knowledge building and sharing; however, they have their own drawbacks.

A proposed knowledge sharing technique, Knowledge Temple, is a feasible improvement to bridge the gap between the well-known pair programming issues. It is a hybrid technique, incorporating knowledge sharing and building models, such as cognitive apprenticeship, on-the-job-training, solo programming, pair programming, parallel peer programming, pair rotation, and knowledge repository creation. This hierarchical approach provides an iterative and incremental solution to share and create knowledge in a collaborative and cooperative fashion.

In Knowledge Temple, individuals work as a small team, a Temple, which has three members with different levels of experience (Figure 1). Every Temple has its own master and two apprentices. In order to achieve an active learning and development environment, every Temple has its own rules and procedures to share the knowledge and increase productivity. This flexible environment creates a collaborative team culture along with cooperative and self-responsible individuals.
The Temple Master leads development and utilizes two apprentices to enhance productiveness. S/he is in charge of communication, revision control, and documentation tools, tracks the collaborative development and progress of apprentices, and ensures the knowledge sharing

process. Moreover, the Temple Master reports to the project manager the progress of work on a weekly basis and discusses potential problems.

Temple Apprentices are free with their internal affairs; however, they are master-dependent on foreign affairs. In other words, the apprentices are responsible for accomplishing determined duties from their Temple master. These duties can be documentation, programming, testing, learning required information, or attending on-the-job training sessions. Yet, they decide their individual duties and the manner of operation.

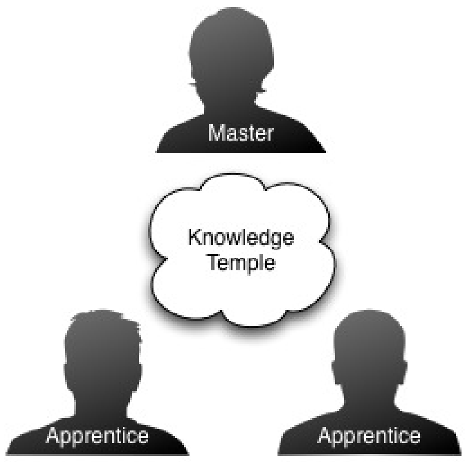

FIGURE 1: Knowledge Temple Paradigm.

Knowledge Temple offers:

- novice-novice inspiration to solve motivation issues,
- development flexibility for expert developers to increase the individual and collaborative productivity,
- schedule flexibility for all the team members to answer the development progress needs,
- hands-on knowledge sharing for agile learners both master and apprentice supported, and
- good use of new knowledge sharing technologies to allow cooperative knowledge transformation and development.

Consequently, the Temple assures high productivity from the Temple Master and collaborative knowledge sharing among the Temple Apprentices.

5. CONCLUSION

In this paper, we review the agile knowledge sharing field and discuss major knowledge sharing issues. Software development is a process, which highly depends on developers' implementation and design experiences. In other words, the tacit knowledge of the developer determines the software development quality. Different software development methodologies introduce different solutions to the knowledge sharing problem within the development team. The effects of knowledge loss and knowledge hoarding are huge for any level software development teams; however, it may create damage that cannot be put back in place for small agile development teams. Knowledge sharing issues are surveyed and categorized into: 1) sociological issues, 2) documentation issues, and 3) implementation issues with/without pair programming. Application environments require various solutions and bring diverse opportunities for software development teams. Finally, a proposed technique, Knowledge Temple, is introduced as a feasible enhancement to well-known knowledge sharing problems for small agile software development teams.

6. REFERENCES

[1] Abbattista, F., Calefato, F., Gendarmi, D., and Lanubile, F. In- corporating social software into distributed agile development environments. In Automated Software Engineering - Workshops, 2008. ASE Workshops 2008. 23rd IEEE/ACM International Conference on (2008), pp. 46–51.

[2] Abdullah, R., and Talib, A. Knowledge management system model in enhancing knowledge facilitation of software process improvement for software house organization. In Information Retrieval Knowledge Management (CAMP), 2012 International Conference on (2012), pp. 60–63.

[3] Akbar, R., and Hassan, M. A collaborative-interaction model of software project development: An extension to agile based methodologies. In Information Technology (ITSim), 2010 International Symposium in (2010), vol. 1, pp. 1–6.

[4] Allison, I. Organizational factors shaping software process improvement in small-medium sized software teams: A multi-case analysis. In Quality of In- formation and Communications Technology (QUATIC), 2010 Seventh Interna- tional Conference on the (2010), pp. 418–423.

[5] Amaral, L., and Faria, J. A gap analysis methodology for the team software process. In Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the (2010), pp. 424–429.

[6] Amescua, A., Bermon, L., Garcia, J., and Sanchez-Segura, M.-I. Knowledge repository to improve agile development processes learning. Soft- ware, IET 4, 6 (2010), 434–444.

[7] Amin, A., Basri, S., Hassan, M., and Rehman, M. Software engineering occupational stress and knowledge sharing in the context of global software development. In National Postgraduate Conference (NPC), 2011 (2011), pp. 1– 4.

[8] Bergersen, G. R., and Sjoberg, D. I. K. Evaluating methods and tech- nologies in software engineering with respect to developers' skill level. In Eval- uation Assessment in Software Engineering (EASE 2012), 16th International Conference on (2012), pp. 101–110.

[9] Bessam, A., Kimour, M.-T., and Melit, A. Separating users' views in a development process for agile methods. In Dependability of Computer Systems, 2009. DepCos-RELCOMEX '09. Fourth International Conference on (2009), pp. 61–68.

[10] Biao-wen, L. The analysis of obstacles and solutions for software enterprises to implement knowledge management. In Information Management and En- gineering (ICIME), 2010 The 2nd IEEE International Conference on (2010), pp. 211–214.

[11] Boehm, B., and Turner, R. People factors in software management: lessons from comparing agile and plan-driven methods. Crosstalk-The Journal of De- fense Software Engineering,(Dec (2003).

[12] Chatti, M., Schroeder, U., and Jarke, M. Laan: Convergence of knowl- edge management and technology-enhanced learning. Learning Technologies, IEEE Transactions on 5, 2 (2012), 177–189.

[13] Chau, T., Maurer, F., and Melnik, G. Knowledge sharing: agile methods vs. tayloristic methods. In Enabling Technologies: Infrastructure for Collabo- rative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE Interna- tional Workshops on (2003), pp. 302–307.

[14] Chowdhury, A., and Huda, M. Comparison between adaptive software development and feature driven development. In Computer Science and Net- work Technology (ICCSNT), 2011 International Conference on (2011), vol. 1, pp. 363–367.

[15] Chua, J. L. Y., Eze, U., and Goh, G. G. G. Knowledge sharing and total quality management: A conceptual framework. In Industrial Engineering and Engineering Management (IEEM), 2010 IEEE International Conference on (2010), pp. 1107–1111.

[16] Crawford, B., Castro, C., and Monfroy, E. Knowledge management in different software development approaches. In Advances in Information Sys- tems, T. Yakhno and E. Neuhold, Eds., vol. 4243 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 304–313.

[17] Devedzic, V., and Milenkovic, S. Teaching agile software development: A case study. Education, IEEE Transactions on 54, 2 (2011), 273–278.

[18] Dorairaj, S., Noble, J., and Malik, P. Knowledge management in distributed agile software development. In Agile Conference (AGILE), 2012 (2012), pp. 64–73.

[19] Duka, D. Agile experiences in software development. In MIPRO, 2012 Pro- ceedings of the 35th International Convention (2012), pp. 692–697.

[20] Dyba, T., and Dingsoyr, T. What do we know about agile software devel- opment? Software, IEEE 26, 5 (2009), 6–9.

[21] Fowler, M., and Highsmith, J. The Agile Manifesto. Software Develop- ment Magazine 9(8) (Aug. 2001). http://agilemanifesto.org.

[22] Ganis, M., Maximilien, E., and Rivera, T. A brief report on working smarter with agile software development. IBM Journal of Research and Devel- opment 54, 4 (2010), 1–10.

[23] Giri, M., and Dewangan, M. A study of pair programming in the context of facilitating the team building. In Advanced Computing Communication Tech- nologies (ACCT), 2012 Second International Conference on (2012), pp. 20–23.

[24] Hazeyama, A., Ogaxne, Y., and Miura, M. Cognitive apprenticeship- based object-oriented software engineering education support environment. In Advanced Learning Technologies, 2005. ICALT 2005. Fifth IEEE International Conference on (2005), pp. 243–244.

[25] Honig, W. Teaching successful ”real-world” software engineering to the ”net” generation: Process and quality win! In Software Engineering Education and Training, 2008. CSEET ’08. IEEE 21st Conference on (2008), pp. 25–32.

[26] Huang, M., and Sun, B. Research on modeling and implementating of knowl- edge management system in virture enterprise. In Machine Learning and Cy- bernetics, 2009 International Conference on (2009), vol. 3, pp. 1424–1428.

[27] Hui, A., and Jing, Z. Evaluation on the cost and performance of knowledge management. In Intelligent Computation Technology and Automation, 2009. ICICTA ’09. Second International Conference on (2009), vol. 4, pp. 201–205.

[28] ISO/IEC/IEEE. Systems and software engineering – developing user docu- mentation in an agile environment. ISO/IEC/IEEE 26515 First edition 2011- 12-01; Corrected version 2012-03-15 (2012), 1–36.

[29] Izquierdo-Cortazar, D., Robles, G., Ortega, F., and Gonzalez- Barahona, J. Using software archaeology to measure knowledge loss in soft- ware projects due to developer turnover. In

System Sciences, 2009. HICSS ’09. 42nd Hawaii International Conference on (2009), pp. 1–10.

[30] Jabar, M., Cheah, C.-Y., and Sidi, F. The effect of organizational justice and social interdependence on knowledge sharing. In Information Retrieval Knowledge Management (CAMP), 2012 International Conference on (2012), pp. 64–68.

[31] Jiang, H., Liu, C., and Cui, Z. Research on knowledge management system in enterprise. In Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference on (2009), pp. 1–4.

[32] Jones, K., Kristof, D., Jenkins, L., Ramsey, J., Patrick, D., Burn- ham, S., and Turner, I. Collaborative technologies: Cognitive apprentice- ship, training, and education. In Collaborative Technologies and Systems, 2008. CTS 2008. International Symposium on (2008), pp. 452–459.

[33] Kavitha, R. K., and Irfan Ahmed, M. A knowledge management frame- work for agile software development teams. In Process Automation, Control and Computing (PACC), 2011 International Conference on (2011), pp. 1–5.

[34] Kopczynska, S., Nawrocki, J., and Ochodek, M. Software development studio - bringing industrial environment to a classroom. In Software Engineering Education based on Real-World Experiences (EduRex), 2012 First International Workshop on (2012), pp. 13–16.

[35] Landaeta, R., Viscardi, S., and Tolk, A. Strategic management of scrum projects: An organizational learning perspective. In Technology Management Conference (ITMC), 2011 IEEE International (2011), pp. 651–656.

[36] Law, A., and Charron, R. Effects of agile practices on social factors. In Proceedings of the 2005 workshop on Human and social factors of software en- gineering (New York, NY, USA, 2005), HSSE ’05, ACM, pp. 1–5.

[37] Levy, M., and Hazzan, O. Knowledge management in practice: The case of agile software development. In Cooperative and Human Aspects on Software Engineering, 2009. CHASE ’09. ICSE Workshop on (2009), pp. 60–65.

[38] Lingard, R., and Barkataki, S. Teaching teamwork in engineering and computer science. In Frontiers in Education Conference (FIE), 2011 (2011), pp. F1C–1–F1C–5.

[39] Lui, K., and Chan, K. Software process fusion by combining pair and solo programming. Software, IET 2, 4 (2008), 379–390.

[40] Marrington, A., Hogan, J., and Thomas, R. Quality assurance in a student-based agile software engineering process. In Software Engineering Con- ference, 2005. Proceedings. 2005 Australian (2005), pp. 324–331.

[41] Mathew, C., Joseph, K., and Renganathan, R. Accelerating organisa- tional learning in the backdrop of knowledge hoarding: A case study with refer- ence to eco-tourism destinations. In Management Issues in Emerging Economies (ICMIEE), Conference Proceedings of 2012 Intenrational Conference on (2012), pp. 63–68.

[42] Ming, C. Research on knowledge management of software enterprises —with lenovo for case study. In Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on (2009), pp. 291–294.

[43] Murray, S., Ryan, J., and Pahl, C. A tool-mediated cognitive apprentice- ship approach for a computer engineering course. In Advanced Learning Tech- nologies, 2003. Proceedings. The 3rd IEEE International Conference on (2003), pp. 2–6.

[44] Neves, F., Correia, A., Rosa, V., and de Castro Neto, M. Knowledge creation and sharing in software development teams using agile methodologies: Key insights affecting their adoption. In Information Systems and Technologies (CISTI), 2011 6th Iberian Conference on (2011), pp. 1–6.

[45] Palmieri, D. W. Knowledge Management Through Pair Programming. PhD thesis, North Carolina State University, 2200 Hillsborough, Raleigh, NC 27695, 2002.

[46] Poff, M. Pair Programming to Facilitate the Training of Newly-hired Pro- grammers. Florida Institute of Technology, 2003.

[47] Prause, C., and Durdik, Z. Architectural design and documentation: Waste in agile development? In Software and System Process (ICSSP), 2012 Interna- tional Conference on (2012), pp. 130–134.

[48] Read, A., and Briggs, R. The many lives of an agile story: Design pro- cesses, design products, and understandings in a large-scale agile development project. In System Science (HICSS), 2012 45th Hawaii International Confer- ence on (2012), pp. 5319–5328.

[49] Rong, J., Hongzhi, L., Jiankun, Y., Tao, F., Chenggui, Z., and Jun- lin, L. A model based on information entropy to measure developer turnover risk on software project. In Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on (2009), pp. 419– 422.

[50] Rong-guang, Q., and Shi-jie, L. Research on comprehensive evaluation of enterprises knowledge management capabilities. In Management Science and Engineering (ICMSE), 2010 International Conference on (2010), pp. 1031–1036.

[51] Salleh, K. Tacit knowledge and accountants: Knowledge sharing model. In Computer Engineering and Applications (ICCEA), 2010 Second International Conference on (2010), vol. 2, pp. 393–397.

[52] Sanders, A. Ten tales of positive change. In Agile Conference (AGILE), 2011 (2011), pp. 181–186.

[53] Sauer, T. Using design rationales for agile documentation. In Enabling Tech- nologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on (2003), pp. 326–331.

[54] Savolainen, J., Kuusela, J., and Vilavaara, A. Transition to agile de- velopment - rediscovery of important requirements engineering practices. In Re- quirements Engineering Conference (RE), 2010 18th IEEE International (2010), pp. 289–294.

[55] Selic, B. Agile documentation, anyone? Software, IEEE 26, 6 (2009), 11–12.

[56] Serrano, M., Montes de Oca, C., and Cedillo, K. An experience on using the team software process for implementing the capability maturity model for software in a small organization. In Quality Software, 2003. Proceedings. Third International Conference on (2003), pp. 327–334.

[57] Sillitti, A., Succi, G., and Vlasenko, J. Understanding the impact of pair programming on developers attention: A case study on a large industrial experimentation. In Software Engineering (ICSE), 2012 34th International Conference on (2012), pp. 1094–1101.

[58] Sousa, F., Aparicio, M., and Costa, C. J. Organizational wiki as a knowledge management tool. In Proceedings of the 28th ACM International Conference on Design of Communication (New York, NY, USA, 2010), SIGDOC ’10, ACM, pp. 33–39.

[59] Srikanth, H., Williams, L., Wiebe, E., Miller, C., and Balik, S. On pair rotation in the computer science course. In Software Engineering Education and Training, 2004. Proceedings. 17th Conference on (2004), pp. 144–149.

[60] Stettina, C., Heijstek, W., and Faegri, T. Documentation work in agile teams: The role of documentation formalism in achieving a sustainable practice. In Agile Conference (AGILE), 2012 (2012), pp. 31–40.

[61] Suganya, G., and Mary, S. Progression towards agility: A comprehensive survey. In Computing Communication and Networking Technologies (ICCCNT), 2010 International Conference on (2010), pp. 1–5.

[62] Sussy, B., Calvo-Manzano, J., Gonzalo, C., and Tomas, S. Teaching team software process in graduate courses to increase productivity and improve software quality. In Computer Software and Applications, 2008. COMPSAC ’08. 32nd Annual IEEE International (2008), pp. 440–446.

[63] Tang, A., de Boer, T., and van Vliet, H. Building roadmaps: a knowl- edge sharing perspective. In Proceedings of the 6th International Workshop on Sharing and Reusing Architectural Knowledge (New York, NY, USA, 2011), SHARK ’11, ACM, pp. 13–20.

[64] Tao, Y., Wang, J., Wang, X., He, D., and Yang, S. Knowledge-based flexible business process management. In TENCON 2006. 2006 IEEE Region 10 Conference (2006), pp. 1–3.

[65] Vanhanen, J., and Lassenius, C. Effects of pair programming at the devel- opment team level: an experiment. In Empirical Software Engineering, 2005. 2005 International Symposium on (2005), pp. 10 pp.–.

[66] Venkatagiri, S. Teach project management, pack an agile punch. In Software Engineering Education and Training (CSEE T), 2011 24th IEEE-CS Conference on (2011), pp. 351–360.

[67] Whitworth, E., and Biddle, R. The social nature of agile teams. In Agile Conference (AGILE), 2007 (2007), pp. 26–36.

[68] Williamson, J. Knowledge needed by an agile enterprise. In Engineering Management Conference, 2003. IEMC ’03. Managing Technologically Driven Organizations: The Human Side of Innovation and Change (2003), pp. 393– 395.

[69] Xie, X., Zhang, W., and Xu, L. A description model to support knowledge management. In Computer and Computational Sciences, 2006. IMSCCS ’06. First International Multi-Symposiums on (2006), vol. 2, pp. 433–436.

[70] Yang, H.-L., and Wu, T. Knowledge sharing in an organization - share or not? In Computing Informatics, 2006. ICOCI ’06. International Conference on (2006), pp. 1–7.

[71] Zanoni, J., Ramos, M., Tacla, C., Sato, G., and Paraiso, E. A semi- automatic source code documentation method for small software development teams. In Computer Supported Cooperative Work in Design (CSCWD), 2011 15th International Conference on (2011), pp. 113–119.

[72] Zhang, C., Tang, D., Liu, Y., and You, J. A multi-agent architecture for knowledge management system. In Fuzzy Systems and Knowledge Discovery, 2008. FSKD ’08. Fifth International Conference on (2008), vol. 5, pp. 433–437.