KNOWLEDGE TEMPLE: A COLLABORATIVE KNOWLEDGE SHARING TECHNIQUE FOR AGILE SOFTWARE DEVELOPMENT

A Thesis

by

ILHAN BURAK ERSOY

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Texas A&M University - Corpus Christi

Corpus Christi, Texas

August 2013

Major Subject: Computer Science

KNOWLEDGE TEMPLE: A COLLABORATIVE KNOWLEDGE SHARING TECHNIQUE FOR AGILE SOFTWARE DEVELOPMENT

A Thesis

by

ILHAN BURAK ERSOY

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Computer Science Graduate Program
School of Engineering & Computing Sciences
Texas A&M University - Corpus Christi

Approved by:

Ahmed M. Mahdy, Committee Chair

John D. Fernandez, Committee Member

Scott A. King, Committee Member

August 2013

ABSTRACT

Despite the productive, flexible, and adaptive nature of agile development, it may suffer from knowledge sharing limitations. This includes knowledge loss due to retirement or high turnover rates of skilled professionals and knowledge hoarding due to interpersonal or organizational climate. The objective of this work is to build a knowledge sharing culture and collaboration norm in the workplace for small agile development teams with high turnover rates, where organizational success is not only maintained but also enhanced. The Knowledge Temple, the proposed approach, is hybrid, 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 is designed as an iterative and incremental solution to share and create knowledge in a collaborative and cooperative fashion. A single-blind experiment was performed with the Innovation in Computing Research (iCORE) at Texas A&M University-Corpus Christi, where the Knowledge Temple technique was administered in three different projects with ten varied Temples. To evaluate this empirical experiment, Temple members' development contributions, a Knowledge Temple questionnaire, and observational outcomes were utilized. Consequently, the results of the Knowledge Temple experiment showed great potential for an impactful approach. The results indicate novice-novice inspiration to solve motivation issues. They also show development flexibility for expert developers that may increase the individual and collaborative productivity. Moreover, this new technique offers schedule flexibility for all the team members, 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.

This thesis is lovingly dedicated to my mother, Aysel Ersoy, and my father, Salih Murat Ersoy. Their support, encouragement, and constant love have sustained me throughout my life.

ACKNOWLEDGMENTS

This research was done at the Innovation in Computing Research (iCORE) at the Texas A&M University-Corpus Christi during the SOAR SI, CCISD, and Museum projects.

I would like to thank professor Ahmed M. Mahdy and all the other members of the iCORE team for their sincere support during this work.

I also wish to express my sincere thanks to professor John D. Fernandez and professor Scott A. King for their unceasing guidance and encouragement.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 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 bugs [14]. Traditional software development methodologies, team software process (TSP) and personal software process (PSP) from the Software Engineering Institute (SEI) [5], and Agile methodologies [15] are very well known models of software engineering. 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, 17, 32, 55, 65, 69]. 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 [14, 20, 27, 64, 66]. 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 educational

partners [5, 60]. Moreover, TSP and PSP models are also performed in academia to create industry-level software engineering for university students [26, 67]. 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 studies in academia [25, 33, 46]. 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 [37].

Knowledge repository creation is an active learning and developing approach [6, 57, 63, 77]. Creating process assets increase the tacit knowledge transformation among developers. Moreover, this technique increases the reusability of externalized tacit knowledge. Yet, version management of created assets and generating 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 [19, 20, 35, 64, 66, 73]. 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

[18, 41, 43, 71]. 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, 29, 47, 59]. The key concept of agile methodologies is creating working software via customer satisfaction and development pace [9, 15, 23, 47]. Therefore software development teams focus more on applying the knowledge than sharing.

In this work, the main goal is to design a collaborative knowledge sharing approach for agile software engineering.

1.1 Problem Description

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 means 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. Pair programming is one of the promising and noticed knowledge sharing techniques for agile development pairs. However, the nature of agile development does not allow a successful knowledge sharing environment for team members. Law and Charron [39] reported:

- *"Pair programming can blend expertise of programmers. However, passive programmers tend to lose their motivation in the pairing activities, as the dominant programmer tends to dictate the path for the development. "*

- *"Time-sharing penalties arose, while attempting to perform a number of tasks concurrently. "*

- *"The programmers need to respond and resolve the issue before the deadline. If the solution did not come on a timely fashion, the team performance evaluation would be impacted according to the service level agreement. As a result, the paired programmers tended to focus of separate tasks, which addressed their own individual deadlines. Since both parties were not dedicating themselves to the common goal related to the original pair programming project, the productivity and schedule would suffer, as a result. "*

In addition to motivational, scheduling, and separating task issues, Chau, Maurer, and Melnik [14] noted:

- *"Assigning two people to work cooperatively as a pair is also an extremely tricky task. One may argue that pair programming constantly reduces the productivity of the experts as they need to train novice all the time and formal training is therefore less expensive. "*

Similar pair programming issues, while performing knowledge sharing, have also been observed [42, 64, 70].

1.2 Proposed Technique

The 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.

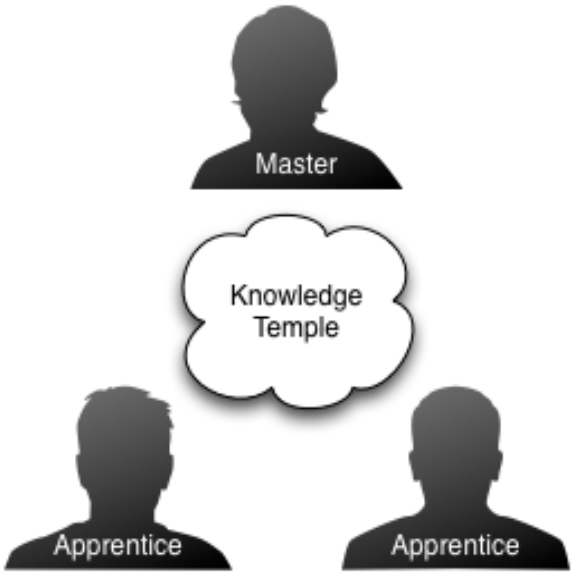


Figure 1. Knowledge Temple Paradigm.

The Temple Master leads development and utilizes two apprentices to enhance productiveness. S/he is in charge of communication, revision control, and documen-

tation 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.

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.

1.3 Challenges

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, 37, 45, 54]. 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 [28].

Knowledge loss is a serious issue for every level of the software development team [11, 30, 53, 74]. 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, 30].

Knowledge hoarding is another serious issue for software development teams [10, 11, 31, 44, 68]. 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 [13, 50]. 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, 16, 51, 76]. 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 [75, 78].

Agile development speeds up the software development process and has high response to customer requirements and changes [3, 21, 39, 40, 56]. 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 [22]. Accordingly, adapting to agile development is an arduous process for both developers and organizations [58].

Although pair programming is a knowledge sharing fashion for agile methodologies, 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 [14, 39, 70].

1.4 Objective and Contributions

The objective of the thesis is to design a knowledge sharing method, where individuals' tacit knowledge transforms into core organizational knowledge and the development productivity concurrently increases through a collaborative and cooperative development pattern. The contributions are:

- designing a new knowledge sharing technique for small agile development teams based on well-known pair programming issues,
- building a knowledge sharing culture encourages the professional development and success of organizations, and
- creating a collaboration norm through knowledge sharing lower the instances

of knowledge hoarding.

1.5 Thesis Organization

The remaining chapters of this thesis will be organized as follows:

- Chapter 2. LITERATURE REVIEW provides an overview of the sociological, documentation, and implementation issues of knowledge sharing.
- Chapter 3. EXPERIMENT DESCRIPTION explains the empirical study environment of the Knowledge Temple experiment.
- Chapter 4. PROPOSED TECHNIQUE: KNOWLEDGE TEMPLE OVERVIEW discusses the general characteristics and operations of the proposed technique.
- Chapter 5. KNOWLEDGE TEMPLE EXPERIMENT presents the development details of the Knowledge Temple experiment.
- Chapter 6. EXPERIMENT RESULTS summarizes the team results through development contributions, the data produced by the Knowledge Temple questionnaire, and observational results.
- Chapter 7. CONCLUSION AND FUTURE RESEARCH provides a summary of findings and presents recommendations for future work.

CHAPTER 2

LITERATURE 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 seen much more easily 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.

2.1 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 [14] 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 [17] 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 [14, 17]. On the other hand, Mathew, Joseph, and Renganathan [44] 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 [16] 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 [31] 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 [14] and giving autonomy to software development teams [16], Law and Charron [39] 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.* [53] 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 [73] 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 contains 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 [30] 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 [76] 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 [76].

2.2 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 [22]. 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 [14] 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 [14]. Law and Charron [39] stress 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. Therefore, agile development teams enable "work-in-progress" documentation fashion along with collaborative authority.

Abbattista, Calefato, Gendarmi, and Lanubile [1] survey the literature on social software development tools. They group the tools into seven categories via their main functionality, which are 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 a fundamental for active knowledge sharing. Finally, their results show that collaboration is a side effect of social software

development teams [1].

Among the social software development tools, Wiki is the most revised tool in consequence of usage and features. It is a practical tool for not only small development teams but also large enterprises [23]. Sousa, Aparicio, and Costa [63] 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 [68] 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 [39] present using mockups as a Wiki documentation technique. They believe "a picture is worth a thousand words" and keep the Wiki web site as visual as possible.

An alternative visual technique proposes Unified Modeling Language (UML)

usage to create minimal the documentation for agile development teams [65]. Stettina, Heijstek, and Faegri [65] divide 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 [50] 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 [50].

2.3 Implementation Issues

Implementing knowledge sharing for agile development teams is more troublesome due to the nature of agile development. 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.

2.3.1 Implementation without Pair Programming

Chatti, Schroeder, and Jarke [13] 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 [13].

Huang and Sun [27] 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 communication control, lifecycle management, knowledge processing, establishing management, operation management, and disassembling management agents. The agents analyze, process, store, and share the knowledge among the whole enterprise with an automated fashion [27].

Zhang, Tang, Liu, and You [78] 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 [78].

Jiang, Liu, and Cui [32] consider a five layered knowledge sharing framework in the interest of organizational knowledge management. The system combines knowl-

edge 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 [32].

Tang, de Boer, and van Vliet [68] 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, unties the communication capability. This roadmapping process, with an indexed pattern, provides a direct knowledge search ability and notification system for formerly-demanded knowledge [68].

Some discussions of the role of applying agile methodologies can be found in Landaeta, Viscardi, and Tolk [38]. 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 [38].

Kavitha and Ahmed [35] 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 [35].

2.3.2 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 [17, 40, 70]. 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 [48]. Moreover, changing partners between other pairs, pair rotation, is highly recommended to achieve an efficient knowledge sharing [35].

Sanders [56] 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 [56] 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 [56].

Chau, Maurer, and Melnik [14] describe pair programming as an informal train-

ing. 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 [14] 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 [23] 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 [23].

Law and Charron [39] 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 [39].

Srikanth, Williams, Wiebe, Miller, and Balik [64] 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 [64].

Poff [49] 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 [49].

Giri and Dewangan [24] 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 [24] 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 [42] 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 [42].

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 [42] 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 [42] emphasize that the task complexity does not affect the effort differences between solo and pair programming.

Sillitti, Succi, and Vlasenko [62] 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 [42].

CHAPTER 3

EXPERIMENT DESCRIPTION

Software engineering is a developing practice compared to other engineering fields or science disciplines. Even if software engineering is still an immature regimen, it has progressed very far in a short amount of time along with new software engineering branches. Agile software engineering is one of the most challenging and promising areas for empirical software engineering research. The nature of agile methodologies require informal, observational, and on-the-job research. Therefore, empirical studies offer an essential way to evaluate new agile approaches. However, researchers argue about the contributions of empirical software engineering research [72] and offer ground rules to improve the results of empirical studies [36, 61].

In order to improve the research and reporting processes, the Empirical Research in Software Engineering Guideline designed by Kitchenham *e*t *a*l. [36] was followed. The researched characterization framework introduced by Shaw [61] and the empirical software engineering research best practices from Weyuker [72] were also considered and utilized. In addition, the Knowledge Survey, which was developed by Palmieri [48], was put into practice as a research and evaluation method.

3.1 Experiment Context

Pair programming is a successful knowledge sharing technique if its requirements are all fulfilled. For a small agile development team, however, applying pair programming causes utter confusion between productivity and knowledge sharing for the pairs. The tight schedule of application development does not allow lead contributors to share their tacit knowledge with newcomers. Moreover, knowledge hoarding issues increase

if the small development team has a high turnover rate. As a result, the small agile development teams may not create harmony for a collaborative and cooperative working environment through pair programming.

In this work, the possibility of a new knowledge sharing technique is discussed, considering the well-known pair programming issues. To enable a collaborative production, the experience gap between the pairs was focused on. Augmenting the knowledge transfer potential was sought, while development productivity was ensured. The issue of scheduling was delved into through development deadline and knowledge sharing burden. Finally, the team determined to create a knowledge sharing culture, which constantly increases team motivation in an agile environment.

3.2 Experiment Population

The Knowledge Temple was applied in the Innovation in Computing Research (iCORE) at Texas A&M University-Corpus Christi. iCORE is a research, development, and commercialization group, which comprises undergraduate and graduate level students. The agile development team of iCORE was formed from sophomore, junior, and senior level undergraduate and Master's level graduate students. The team did not include freshman level undergraduate students due to their insufficient programming abilities. All the students were part-time workers, who contributed ten or twenty weekly work hours as a part of the agile development team.

The varied levels of computer science students created an environment that could be considered as a real world atmosphere:

- sophomore and junior level undergraduate students as newly-hired developers or interns,

- senior level undergraduate students as junior developers, and
- master's level graduate students as senior developers.

Therefore, iCORE offered a unique empirical research environment for an observational experiment. Moreover, it is essential to have a diverse group of team members to effectively evaluate knowledge sharing results. It is assumed that the expert developers have more experience on project development requirements than apprentice developers in Knowledge Temple experiment.This unique environment exposed a mandatory employee turnover rate through the graduation of team members.

The cultural diversity of iCORE also offered an outstanding research environment. The experiment population contained team members from the United States, Vietnam, India, and Turkey. It allowed for the creation of a melting pot of different cultures and work ethics. In addition, applying the proposed technique in a university environment was promising because today's students will be tomorrow's professionals; thus, it was important to get results from future generations.

The agile development team had fifteen members. Each team member named as TMb# (Team Member #), where "#" stands for both the sequence of recruitment and ID number. For instance, TMb1 joined the development team first and TMb18 was last. The numbering system is important to comprehend the evolution of the team members. Nonetheless, it does not show the experience difference between team members because there is an opportunity that a Master's level graduate student can join the agile team after a sophomore level undergraduate student or vice versa.

3.3 Experiment Projects

The experiment environment had different levels of developers and different types and levels of projects. The proposed knowledge sharing technique was applied to three different projects. One of the projects was examined through version 0, version 1 and version 2 standings. Moreover, the proposed approach was applied not only to programming but to every aspect of the project development process, such as client collaboration, application publishing, and project presentation. SOAR SI, CCISD, and Museum were the names of the projects.

The SOAR SI project was an informative mobile application for science, technology, engineering, and mathematics (STEM) undergraduate students. It offers schedule, location, and orientation about supplemental instruction (SI) sessions offered by the Title V-STEM Outreach, Access, and Retention (SOAR) Program at Texas A&M University-Corpus Christi. The application contains six touch user interfaces (TUI) and nine development modules. The development team utilized the Appcelerator Platform and the Titanium SDK as the mobile application development platform. The SOAR SI project was published for both iOS (iPhone and iPad) and Android (smartphones and tablets) devices.

The CCISD project is a full educational guidance application for Corpus Christi Independent School District (CCISD). It presents a school directory, CCISD school calendar, CCISD lunch menu, CCISD news, CCISD athletics, reporting a bully functionality, and more. The application contains twelve touch user interfaces (TUI) and fourteen development modules. The development team utilized the Appcelerator Platform and the Titanium SDK as the mobile application development platform. The CCISD project was developed for both iOS (iPhone and iPad) and Android (smartphones and tablets) devices.

The Museum project is a full body interactive wall with custom design exhibits for the Corpus Christi Museum of Science and History. It introduces a dynamic projected content on the museum wall for children through interactive science and history education. Adobe Flash Professional and GroundFX Flash SDK from GestureTek were selected as the development platforms. The Museum project was under prototyping process, which was designed for a special interactive wall projection system.

3.4 Experiment Technologies

The use of technology is a driving force for software engineering methodologies. Especially for agile development, there is a skyrocketing market for different methods, conditions, and settings. The Knowledge Temple presents a knowledge sharing technique; however, building knowledge sharing culture within the organization and beneficial technology solutions for the agile development team are the beginnings of success. Therefore, any technological tool that works for the Temple was the point of interest. It was also important to build a balance for the flexible operating manner for the Temples.

In iCORE, it is a rule to use Bitbucket as a version control system. Bitbucket, a web-based hosting service for projects, allows public and private project repositories, team management, code reviews, and source code insight. Therefore, the Temple development and sharing progress was tracked by the developer submissions, assigned issues, Wiki, and comments through source code reviews. However, some Temples also took advantage of Trello for their project management purposes.

For mobile development, the Appcelerator Platform was used. The Titanium SDK employs only JavaScript language for creating native applications across dif-

ferent mobile devices. Using one development language for both iOS and Android development accelerated the development iterations. Moreover, the modular development design of Titanium allowed the team to build an on-the-job knowledge sharing culture through code modules. Another script-based platform, Adobe Flash Professional is also used in this experiment.

For documentation purposes, the development team suggested using the JSDoc documentation tool. It is an inline API documentation tool for Javascript. Therefore, the Temple members added documentation comments to source code to create Wikis for knowledge sharing fashion.

To enhance communication and collaboration, the development team facilitated different video conferencing tools. They made use of Skype and Google Hangouts occasionally. However, TeamViewer was the widely used tool to establish a flexible time-sharing. The screen sharing and browser-based presentation features were indispensable and formed a robust learning environment.

In addition to software products, the team employed Alienware 23-Inch Desktops, 21-Inch iMacs, a projector, and a multi-touch smart board. The desktop computers were put in practice for development, testing, and knowledge sharing. The projector was used when Temples met in the brainstorming area at iCORE. Nonetheless, the most engaging learning tool was the smart board, in the iCORE conference room, because team members performed knowledge sharing, testing, informative Temple meetings, and customer collaboration with the help of the smart board. It increased the team motivation and empowered application interaction with its multi-touch feature.

3.5 Experiment Questionnaire

In addition to observational research, a survey was utilized as one of the research methods because of the high developer turnover in iCORE. Most of the participants had graduated and started to work in different parts of the United States while the progress of the experiment was being observed.

Palmieri [48] developed the Knowledge Survey to assess the experiment of using pair programming as a knowledge management technique. It was essential to use a survey that had already proved reliable and valid. All the questions were closed-ended to offer the same mental set while answering the questionnaire. The questions were kept as similar as possible to perform a similar questionnaire concept to the proposed solution. The survey was divided into 3 sections:

- Section 1: Knowledge Sources
- Section 2: Knowledge Acquisition, Dissemination, and Maintenance
- Section 3: Demographical Background Information

In Section 1, the questions were designed to investigate the sources utilized instead of emphasizing knowledge sharing terminology. In addition, Section 1 questioned the tools and knowledge sharing procedures that the proposed technique should evaluate. Section 2 investigated the organizational strategy on knowledge sharing. Additionally, Section 2 inquired about the effect of the proposed technique on knowledge hoarding and employee turnover. Finally, Section 3 had questions to capture the demographical background information of the team members.

Required sections were modified in order to fulfill the experiment context. In Section 1 and Section 2, some questions were deleted due to the Knowledge Temple

technique progress and experiment resources. In Section 3, the question that directly related with pair programming was deleted and two questions were added instead:

- How satisfied are you working in a small team with 2 peers?
- How satisfied are you working in a small team with a peer?

The newly added questions investigated the receptiveness of team members, who worked in a small group of three people, to each other and the method. Participants were asked to reply through their satisfaction level. Their feedback formed the fundamental results of our research and the foundational theory for future studies.

CHAPTER 4

KNOWLEDGE TEMPLE OVERVIEW

The paradigm shift from knowledge 'management' to knowledge 'sharing' has allowed software development teams to focus on the team members and their culture as much as their productivity. Maintaining productivity requires sustaining team member motivation, especially, for agile development teams. In addition, a good organizational culture transforms team development motivation to a successful knowledge sharing environment.

4.1 Evolution Knowledge Temple Practice

In order to create a knowledge sharing culture, pair programming was implemented with a small agile development team at iCORE. Three different types of progress was observed in every iteration cycle:

- Beginning of the iteration: low productivity and high knowledge sharing
- Middle of the iteration: medium productivity and low knowledge sharing
- Near the end of the iteration: high productivity and very low knowledge sharing

There was an inverse relationship between production level and knowledge sharing level. In every sprint, because of tight project deadlines and high turnover rate, high productivity and at least medium knowledge sharing was required. This requirement increased the responsibility burden of the expert developers. Both application development and knowledge exchange were fulfilled by the agency of expert developers, and it was the cause of their responsibility burden. To accomplish high levels of

knowledge sharing, expert developers were paired with novice developers. It was the only way of growing the agile team because of the iCORE's developer resources.

The outcome of applying pair programming was not successful. It was either inadequate productivity and good knowledge exchange or good productivity and inadequate knowledge exchange. The novice programmers made lots of complaints about expert developers' availability. On the contrary, expert developers reported novice developers' motivation level as 'ground-level intentness.'

Even if pair programming was not the optimum choice, a natural apprenticeship instance between expert and novice developers occurred. The cognitive apprenticeship theory proceeded through expert mentoring rather than pair programming. However, it was not enough for carrying out the novice developers' contribution and sharing.

A middle-man was utilized between the expert and novice developers, creating the Knowledge Temple. The middle-man should:

- free the expert developer to increase the productivity,
- support the novice developer to stimulate learning curve,
- contribute toward the development progress, and
- hold up the development and knowledge sharing structure.

Consequently, small teams of three were formed and named as 'Temple.' Every Temple had a mandatory expert developer and two apprentices, entitled Temple Master and Temple Apprentices.

4.2 Knowledge Temple Technique

Having two apprentices under the influence of a lead created a core team culture. Cognitive apprenticeship theory is the dominant characteristic of the Knowledge Temple, as it is in human nature. The leadership of the Temple Master is as important as the will and autonomy of Temple Apprentices. However, the Temple Master has a high responsibility to sustain the Knowledge Temple mechanism. As shown in Figure 2, the Knowledge Temple contains three different zones addressing development and knowledge sharing.

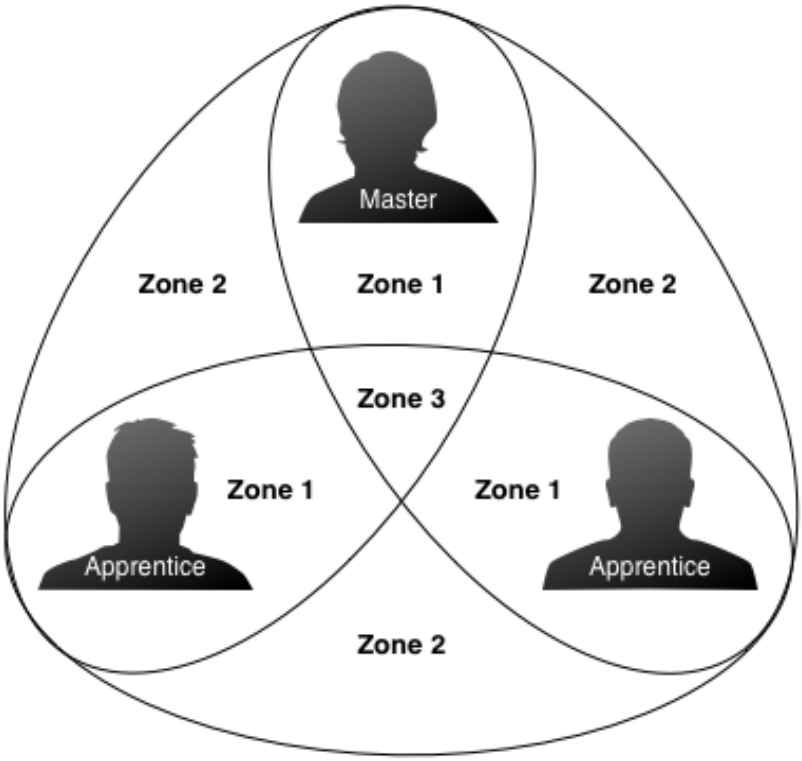


Figure 2. Knowledge Temple Technique.

Zone 1 is the Temple phase where the Temple Master and Apprentices perform solo programming. Zone 1 is extremely important for productivity when there is tight deadlines. The Temple Master should reserve development time, particularly when the development contribution of the Temple Apprentices is low. Moreover, project management meetings can be performed between the Temple Master and project manager as a zone 1 activity. While the Temple Master is in zone 1, the Temple Apprentices can stay in the phase of zone 1 or they can call for zone 2 between them. Being in the phase of zone 1 for the Temple Apprentices is essential both for development and knowledge building. The Temple Master has the privilege to assign duties, which can be a contribution for productivity, hands-on learning tasks, or a knowledge repository creation.

Zone 2 is the phase where the Temple works as pairs. There are two ways of pairing: Master - Apprentice or Apprentice - Apprentice. The Master - Apprentice pairing allows higher productivity than knowledge sharing. On the other hand, the Apprentice - Apprentice pairing enables knowledge sharing more than productivity. In zone 2, pairs can perform on-the-job-training, pair programming, parallel peer programming, and knowledge repository creation. In addition, the nature of the Knowledge Temple technique facilitates pair rotation. Pair rotation can be performed in the Temple and among the Temples because an apprentice for Temple 1 can be an apprentice for Temple 2. For this reason, knowledge spreads in the agile development team like a social network.

In zone 3, both Temple Master and Apprentices come together and carry out activities as a team. Zone 3 is the core of the Knowledge Temple technique. It is the phase that lowers Temple member production, but highly increases the knowledge sharing and team building activities. The Temple Master creates the meeting agenda

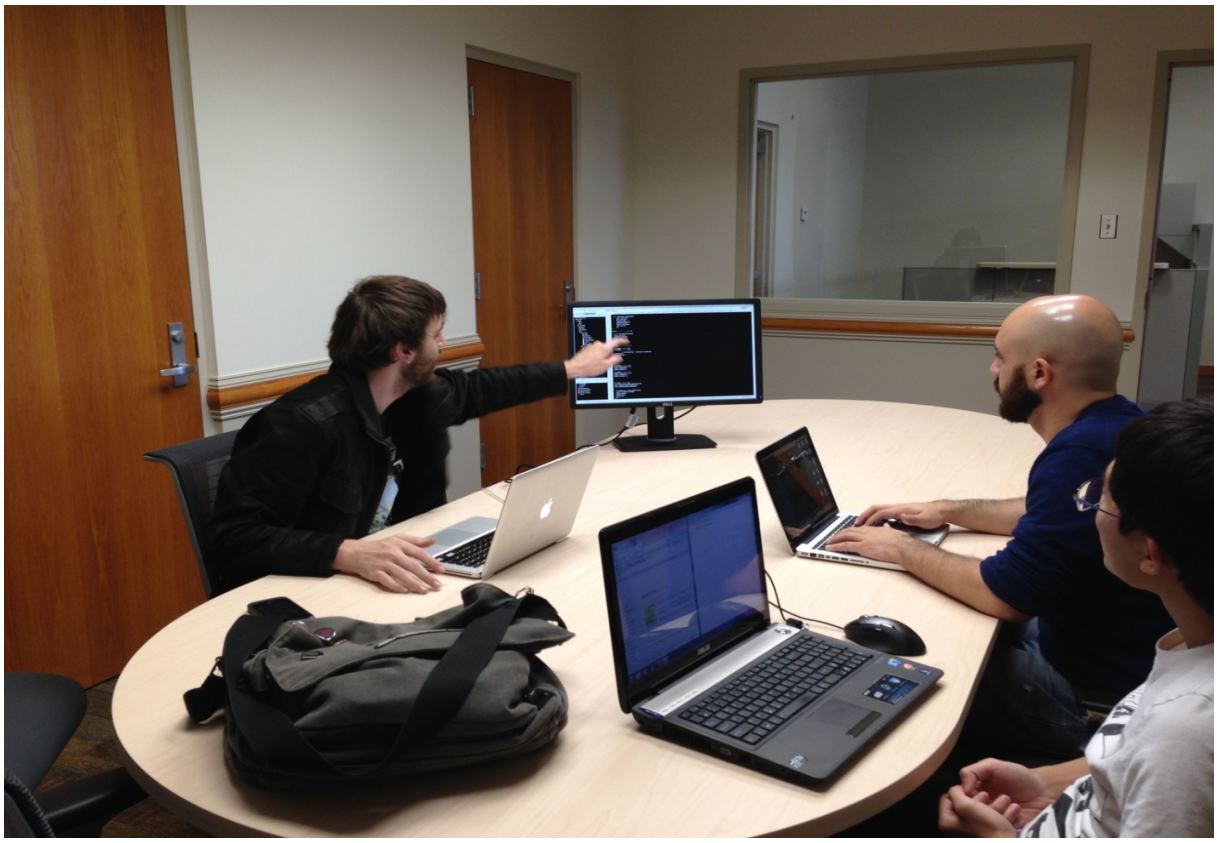


Figure 3. A Zone 3 Meeting.

for the zone 3 phase through the progress of development. Temple members may engage in brainstorming, on-the-job-training, formal training, code revision, code inspection, Q&A sessions, or enhancing the communication between Temple members. Figure 3 is an example of a Q&A session for a zone 3 meeting. The Temple forms a team structure in zone 3 to overcome the sociological issues of knowledge sharing. Furthermore, the project managers can be a part of the zone 3 meetings in order to monitor the Temple efficiency.

4.3 Building the Temple

The Temple initiation is an essential period in the life of the Knowledge Temple. Assigning the Temple Master among the agile development team is a simple but not easy task. It is simple because the selection process is related to the project and required development talents. Therefore, the number of available Temple Masters decreases through their required development experience. It is not easy because the Temple Master should have leadership and tutoring abilities to enhance knowledge sharing and team management. However, the team environment of Temples help the Temple Master for both managing the team and maintain the development quality. After deciding the Temple Master, it is time to select the apprentices. The apprentice selection depends on the project requirements, which may demand:

- High productivity,
- A balance between productivity and knowledge sharing, or
- High knowledge sharing.

However, it is essential to keep the knowledge sharing level no less than medium because the high turnover rate is a concerning issue for all small agile development teams. Furthermore, a master may serve as an apprentice depending on the project requirements and expert skills.

The Temple, containing three expert team members, empowers high productivity. In this setting, the Temple Apprentices take more responsibility for application development. At the same time, they obtain more information about the project, the status of the project, and the development method of the project. They adapt faster for both the development and knowledge sharing phases. In addition, using three expert team members is a good way of growing new Temple Masters.

To create a balance between productivity and knowledge sharing, the Temple should contain one expert, one average, and one novice level developers. This is the best setting for the Knowledge Temple technique because it completely fulfills the middle-man approach. The Temple apprentice, who has an average level of experience, supports the other Temple apprentice for both development and knowledge sharing needs. Moreover, an average apprentice contributes to application development much more than a novice apprentice, which makes The Temple Master's job easier. At the same time, he learns from the Temple Master quicker than a novice apprentice, and shares the knowledge with the novice apprentice efficiently.

An expert and two novice developers form the Temple for high knowledge sharing. Two Temple Apprentices, who have almost the same level of experience, enables a strong learning environment. Even if their contribution to development is minor compared to other Temple alternatives, the Temple Apprentices have a strong motivation to exchange knowledge. This ambitious impulse provides a big potential for future projects. In addition, two novice Temple Apprentices may affect the Temple Master's productivity. However, the Temple develops more innovative approaches due to an increased number of Temple meetings.

4.4 Knowledge Temple vs. Jedi Temple

The fun factor of agile development is also indispensable. It encourages team building and team unity. Star Wars™ was selected as the theme of the Knowledge Temple technique (Figure 4).

There are two reasons behind the Star Wars™ theme. First, Star Wars™ is a very popular movie among people in science and technology [12, 34, 52]. Therefore, introducing a new technique with a well-known and beloved theme allows an

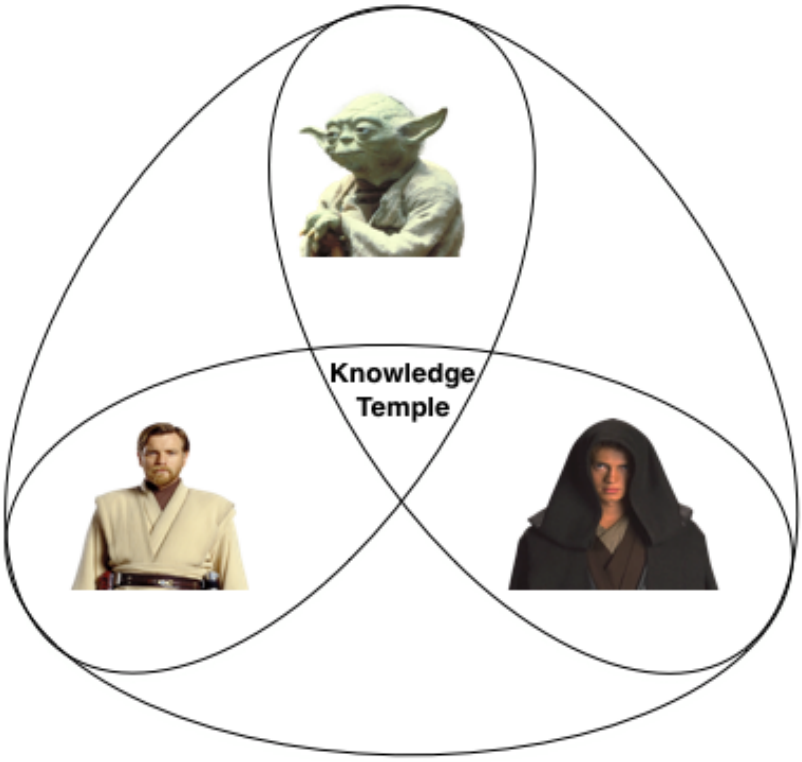


Figure 4. Knowledge Temple vs. Jedi Temple.

immediate adaption. Second, the nature of the selected Star Wars™ characters are self-descriptive for both the Temple roles and the Knowledge Temple mechanism.

The Yoda character in the Star Wars™ universe is selected as the Temple Master in the Knowledge Temple technique, due to his leadership, mentorship, and high talents. The Obi-Wan and the Anakin characters are the Temple Apprentices in the proposed technique because of their cooperative and collaborative efforts in the Star Wars™ universe. Through the selected characters, the agile development team conceptualizes the roles and the role hierarchy in the Knowledge Temple technique. Moreover, the interaction between the selected Star Wars™ characters describes the

the Knowledge Temple mechanism. Yoda, Obi-Wan, and Anakin may accomplish quests as a tightly-coupled team, loosely-coupled teams, or solo heroes. They have their individual and team responsibilities and report to each other through their hierarchy. They are always determined in their quests, eager to learn the power of the Force, and respectful to each other. As a result, the ambiance of the Jedi Temple in the Star Wars™ universe is the ideal scene for the Knowledge Temple technique.

CHAPTER 5

KNOWLEDGE TEMPLE EXPERIMENT

Observing the effects of the proposed technique in a small agile development team was crucial. The Knowledge Temple experiment was performed on iCORE agile development team members. However, the team members were not informed about the experiment. The proposed technique was introduced as iCORE's development and knowledge sharing culture. Therefore, iCORE members did not feel the pressure for experiment results. This single-blind process allowed us to collect genuine and autonomous behavior. In addition, the Star Wars^{TM} theme was put into practice for both describing the technique and augmenting the eagerness of the team members.

5.1 The Environment

iCORE was an enthusiastic experiment environment through its:

- team members' cultural and experience diversity,
- projects' difficulty level and variety, and
- availability of technological tools.

However, iCORE offered a high employee turnover rate and spontaneous knowledge hoarding between the team members. Although the total number of participants was fifteen, the active development members decreased or increased gradually due to students's graduation. iCORE lost nine team members during the experiment, which occurred in the Fall 2012, Spring 2013, and Summer 2013 semesters. This situation created a great opportunity for evaluating the Knowledge Temple technique through the knowledge sharing progress. However, it was also a great challenge for both

experiment management and development continuity. Moreover, iCORE behaved as a real workplace culture with students functioning in a competitive job environment. As a result, a more preservative behavior with knowledge sharing appeared through the experiment.

5.2 Projects and The Temples

The Knowledge Temple technique applied to the SOAR SI version 0, SOAR SI version 1, SOAR SI version 2, CCISD, and Museum projects. Every project's progress and development results contributed to the evolution of the proposed technique. The

Temple#	Temple Master	Temple Apprentice	Temple Apprentice
1	TMb2	TMb1	TMb11
2	TMb13	TMb3	TMb6
3	TMb12	TMb9	TMb10
4	TMb6	TMb13	TMb14
5	TMb12	TMb10	TMb8
6	TMb2	TMb13	TMb3
7	TMb9	TMb13	TMb6
8	TMb9	TMb13	TMb3
9	TMb12	TMb14	TMb15
10	TMb2	TMb3	TMb15

Table I. Temples of Knowledge Temple Experiment.

Knowledge Temple life cycle was divided into three stages: alpha, beta, and release. The alpha stage covered the growth, the beta stage included the maturing, and the

release stage contained the diversification phases of the Knowledge Temple technique. As shown in Table I, every Temple has three TMbs appointed as Temple Master and two Temple Apprentices. Ten Temples was formed during the Knowledge Temple experiment.

In addition, the development team performed pair programming with three different pairs in the early phases of iCORE's knowledge sharing culture. However, the need for more efficient alternatives oriented the research towards creating a new hybrid knowledge sharing technique.

5.3 Alpha Stage

At the beginning of the alpha stage of the Knowledge Temple experiment, the team experienced the pair programming dilemma due to sociologic and implemental manners. For SOAR SI version 0, the team applied pair programming with two different pairs. The development process was divided into two parts: method and database development. Pair 1 was in charge of the method development part and Pair 2 was in charge of database development. As shown in Figure 5, TMb4 and TMb7 were the members of Pair 1, and TMb5 and TMb6 formed Pair 2. Due to agile development team resources, an expert developer was paired with a novice one.

At the beginning of the development cycles, pairs did not face any difficulty in developing and sharing knowledge. The team recognized that the development pace was not sufficient; however, they were regardful about the knowledge sharing progress. Meanwhile, the client was excited about iCORE's client collaboration process; thus, they were expecting an incremental outcome from the development team. Although the development cycles were not changed, the productivity decreased through method development, which required more experience. In addition, TMb7

Figure 5. The Pairs of SOAR SI Version 0.

graduated, and consequently, Pair 1 was formed as TMb4 and TMb8. After the necessary change, Pair 1 suffered through the knowledge sharing process. Along with the tight deadlines, expert peers started to focus on development more than sharing. They stopped applying pair programming because the level difference did not allow them to continue developing as pairs. On the other hand, novice peers complained that expert peers were not available and did not associate with the novice peers.

At the end of Fall 2012, one of the lead developers of the iCORE agile development team, TMb4, graduated. The assumption was that the other member of Pair 1 would step up and continue working on SOAR SI version 0. However, TMb8 was not ready, and the team failed development sustainability. The experiment benefit

Figure 6. The SOAR SI Version 1 Knowledge Temple Meeting with All Temples.

extracted from SOAR SI version 0 was the cognitive apprentice behavior of our agile developers. Even if pairs had tried to apply pair programming, expert developers treated novice developers as apprentices because of the experience difference.

At the beginning of Spring 2013, the proposed technique, the Knowledge Temple, was instigated in iCORE. As the result of the previous semester's failure and insufficient developer talents, the application environment was changed to Appcelerator for SOAR SI version 1. The Knowledge Temple technique and appropriate tools to iCORE's agile development team was introduced through a meeting (Figure 6). All the Temples utilized a version control system, Bitbucket, to enhance the collaborative and cooperative work.

At the end of the meeting, the assigned three Temples performed project-wise and Temple-wise meetings. The Temple Masters of Temple 1, Temple 2 and Temple 3 determined the schedule for both development and knowledge sharing (Figure 7).

Figure 7. The SOAR SI Version 1 Temples.

Every Temple created its own internal schedule and development style separately; however, all Temples obeyed the project deadlines. The progress of the Temples was followed through consecutive meetings. The Temples met the development demands, but the knowledge sharing progress also needed to be examined. Therefore, Temple 4 and Temple 5 were altered from Temple 2 and Temple 3. The new developers, who joined Temple 4 and Temple 5, did not slow down the productivity of the project. However, more Zone 3 and Zone 2 meetings were reported by Tem-

ple Apprentices through the production process. Moreover, the Temple Masters of Temple 4 and Temple 5 denoted more Zone 1 study for the production while the Temple Apprentices performed Zone 2 meetings. Consequently, Temple 1, Temple 4, and Temple 5 finished the application development by the deadlines.

5.4 Beta Stage

The development sustainability was tested through the beta stage of the Knowledge Temple experiment. After the project development of SOAR SI version 1, Temple 6 was formed for the application distribution and Temple 7 was formed for SOAR SI version 2. All the Temple members of Temple 6 and Temple 7 were former Temple

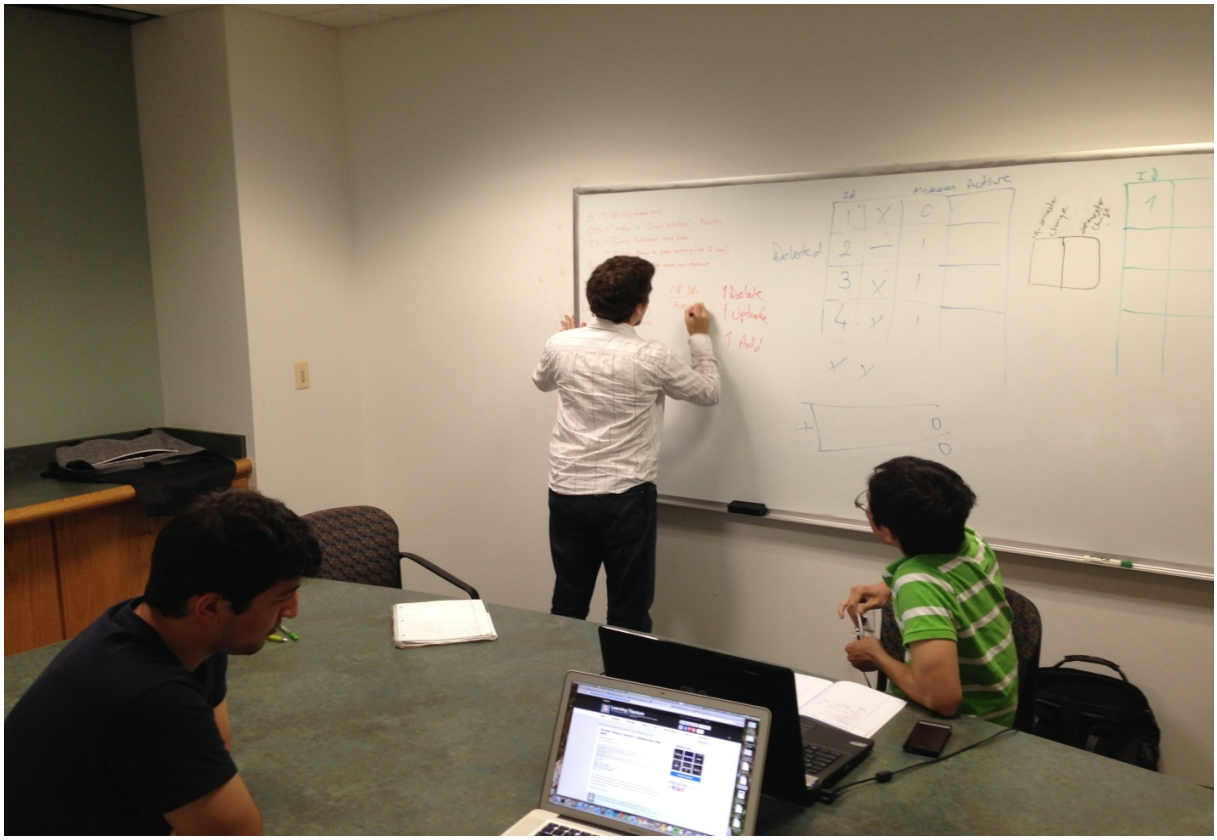


Figure 8. A SOAR SI Version 2 Temple Meeting.

members of SOAR SI version 1. Thus, there was a chance to observe not only productivity rates but also knowledge transfer between the Temple members.

Temple 6 was responsible for both iOS and Android distribution. They created the required accounts and followed the distribution progress. SOAR SI version 1 was published successfully by Temple 6 in the Summer semester of 2013. Moreover, Temple 6 was tasked to fix the reported issues after application distribution. They also worked coordinately with Temple 7 for SOAR SI version 2. As shown in Figure 8, Temple 7 started SOAR SI version 2 development simultaneously. They designed the new features for the SOAR SI application. One of the Temple members also graduated; thus, Temple 8 was assembled. Again, the new member of Temple 8

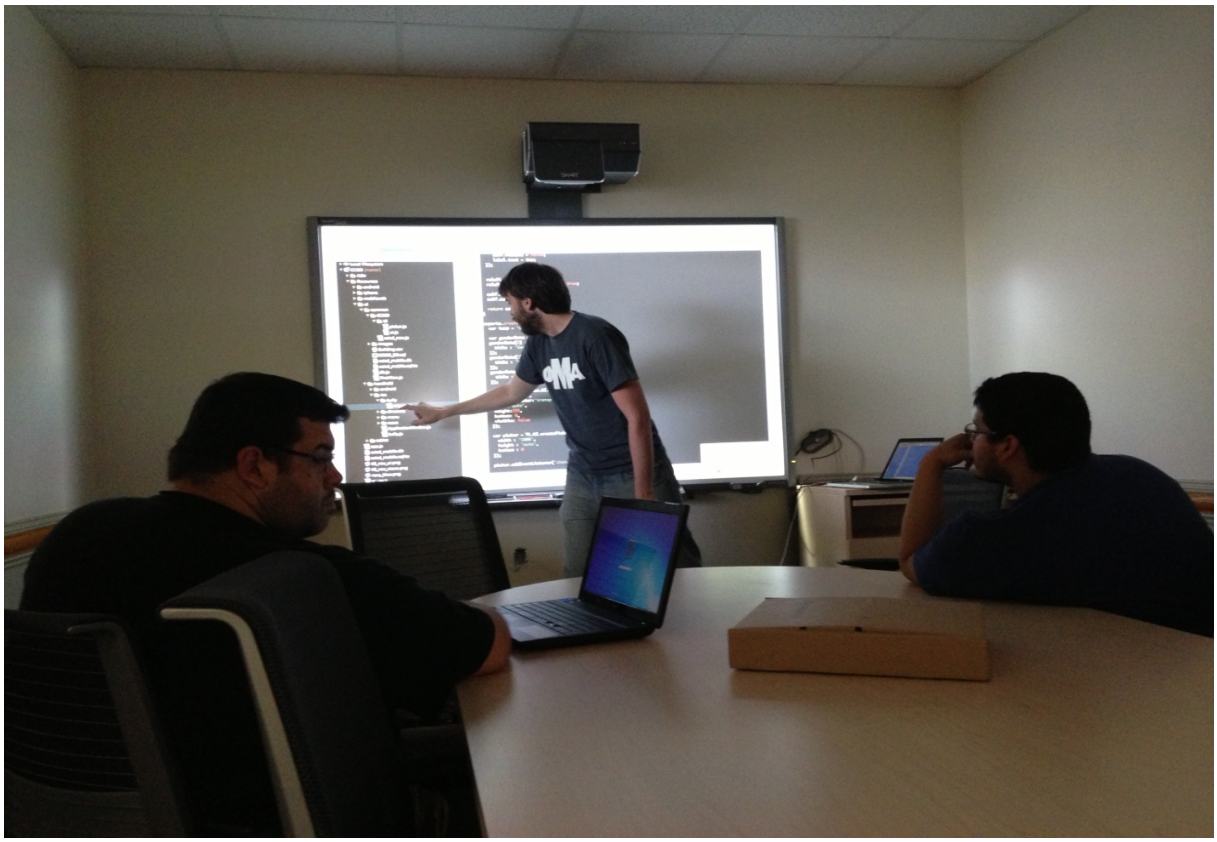


Figure 9. A CCISD Project Temple Meeting.

was selected from the development members of SOAR SI version 1. Therefore, the team took advantage of former experienced members by means of productivity, and the consequences of knowledge lost through employee turnover was hypothesized to decrease.

5.5 Release Stage

In the release stage, the shared knowledge was put into practice for new projects. Temple 9 was created for the CCISD project (Figure 9). The Temple Master and one of the Temple Apprentices experienced the Knowledge Temple technique in SOAR SI version 1. The other Temple Apprentice was new to the mechanism of the Knowl-

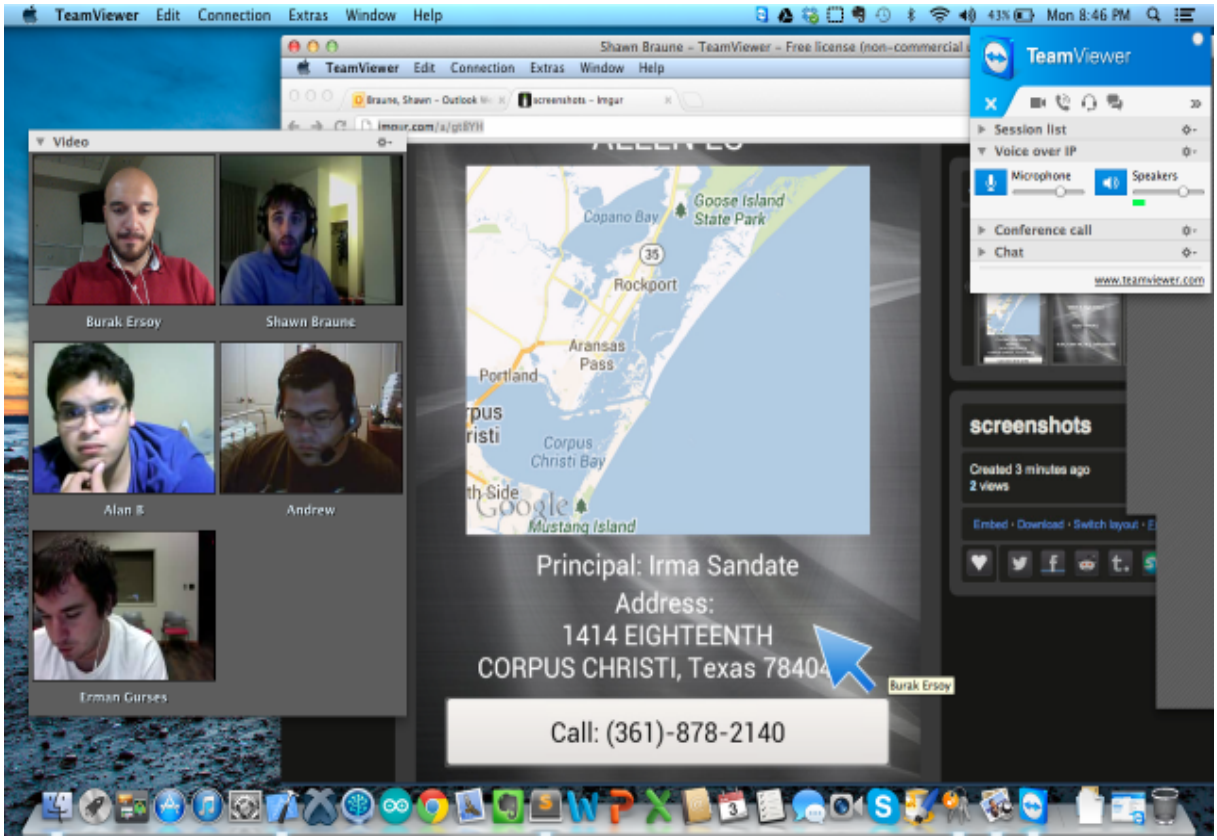


Figure 10. A CCISD Project Temple Meeting through Team Viewer.

edge Temple. The technique was not introduced to the newcomer. Experienced members of Temple 9 welcomed him, and they naturally started the development process. The team also utilized Bitbucket for the CCISD project. Bitbucket helped knowledge sharing studies through code insight and code review. Moreover, Temple 9 followed the development progress in Bitbucket. The CCISD Temple Apprentices were more prepared than the SOAR SI Temple Apprentices because the CCISD Temple Apprentices made use of the SOAR SI outcomes, which were stored in Bitbucket, for both development and knowledge sharing.

Figure 11. A Client Collaboration Meeting.

The Knowledge Temple technique allowed Temple 9 to continue their team studies in a distributed environment. For the Summer 2013 semester, the Temple Master

of Temple 9 had to leave Corpus Christi for another job. However, the Temple stayed together because of the flexible behavior of the proposed technique. Temple 9 continued their development both in the Zone 2 and Zone 1 phases of the Knowledge Temple technique. As shown in Figure 10, they performed Zone 3 meetings through Team Viewer. Hence, a knowledge sharing environment was established through collaborative development. The Temple 9 Apprentices also performed peer parallel programming. They worked on the same subject simultaneously to create, capture, and learn the tacit knowledge of the Temple Master.

Figure 12. A Museum Project Temple Meeting.

In addition, Temple 9 arranged client collaboration meetings. All the Temple members attended client meetings (Figure 11). The Temple Master was the one

who led the informative, project status update, and future plan conversations. At the same time, the Temple Apprentices worked as note takers, experiencing meeting customs, and each of them had the chance to meet with client for future necessities.

Figure 13. Museum Project Temple.

The initiative of using the Knowledge Temple technique rather than the application development fashion continued also in Temple 10 studies. Temple 10 was responsible for the Museum project, for which there was not a former knowledge repository. Therefore, they started Zone 3 and Zone 2 sessions for knowledge sharing and project design workshops(Figure 12). They utilized the proposed technique not only as software development training but also in other job requirements of the project. For instance, Temple 10 utilized the Knowledge Temple technique for as-

sembling the portable interactive projection system (Figure 13). As a result, the Temple members became a part of every aspect of the project.

CHAPTER 6

EXPERIMENT RESULTS

Evaluating empirical software engineering research was a complicated process. To acquire reliable results, the contribution of Temple members was analyzed through the Bitbucket platform and a questionnaire was administered after the Knowledge Temple experiment. Moreover, observational experiences from applying the Knowledge Temple technique in a small agile development team was shared. The Knowledge Temple questionnaire was a variation of a questionnaire by Palmieri [48], which was a main instrument used to evaluate the effect of pair programming as a knowledge management approach. Some questions were altered to best fit the proposed Knowledge Temple technique. Moreover, preliminary analysis was performed on the data to ensure integrity.

6.1 Team Member Contribution

Bitbucket is a web-based hosting service, which offers revision control, code insight and code review. Therefore, Bitbucket was utilized for development, knowledge sharing, and evaluating the development progress. All the team members had their individual accounts on the Bitbucket system. Through this account, they accessed the code of the project and contributed to the project by module development. Whenever a team member accomplished a working copy of the module, s/he submitted this version through the version control system. As shown in Table II, the number of submissions from Temple Masters and Temple Apprentices was evaluated. In addition, the production modules were designed with as comparable size and complexity as possible. This was largely feasible due to the nature and type of the projects that

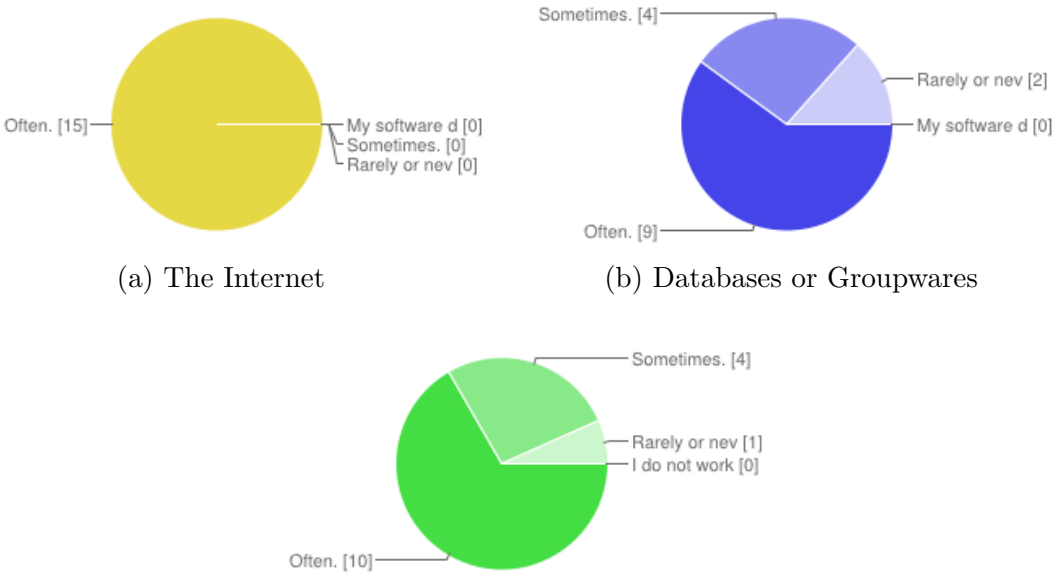
the experiment was conducted on.

For SOAR SI version 1, 256 submissions were recorded in Bitbucket. The three Temple Masters completed 141 submissions and the eight Temple Apprentices achieved 115 submissions. As a result, the Temple Masters developed 55% of SOAR SI version 1. It was expected that the three Temple Masters lead productivity; however, the contribution from the eight Temple Apprentices was higher than expected. The reason for this was the on-the-job learning culture of the Knowledge Temple technique.

Project Name	Total Submissions	Temple Masters	Temple Apprentices
SOAR SI Version 1	256	141 (55%)	115 (45%)
CCISD	115	65 (56.5%)	50 (43.5%)

Table II. Submission Results.

In the CCISD Project, 115 submissions have been archived at the time of this research. The CCISD project is still under development. The Temple Master completed 65 submissions and the two Temple Apprentices achieved 50 submissions. Consequently, the productivity contribution of the Temple Master was higher (56.5%) than the two Temple Apprentices (43.5%) as expected. However, this result established the fact that the Temple was exchanging and building knowledge through the Knowledge Temple technique process while they were developing software. Moreover, the Temple Master acknowledged that the Temple Apprentices were able to apply the gained knowledge from the former project into the CCISD project.



(a) The Internet

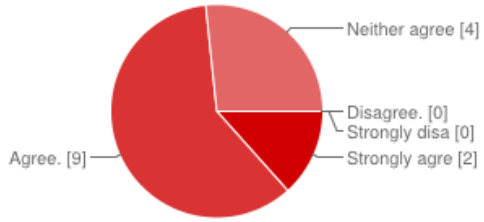
(b) Databases or Groupwares

(c) Co-workers

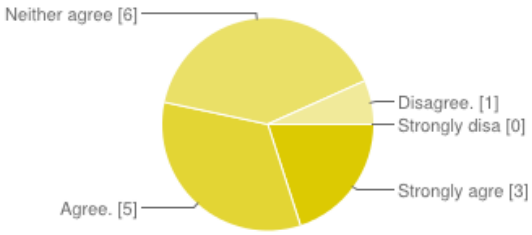
Figure 14. Knowledge Sharing Sources.

6.2 Questionnaire Results

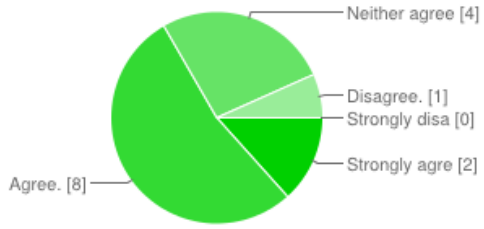
Fifteen responses for the Knowledge Temple Questionnaire were received, which was the total number of experiment participants. Through the single-blind experiment approach, the questions could not be asked directly regarding the Knowledge Temple technique. The first section of the survey investigated the participants' knowledge source penetration for knowledge building behavior (Figure 14). Internal publications, design documents, external publications, the Internet, co-workers, classroom or online courses, and databases or groupwares as a knowledge source were the foci. The participants indicated almost equal usage of internal publications (77%), design documents (73%), external publications (66%), and classroom or online courses (74%) for knowledge sharing and knowledge building purposes. However, the Internet (100%), databases or groupwares (87%), and co-workers (84%) were voted as the



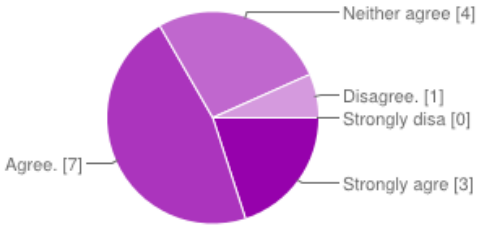
(a) Creating Knowledge through People and Technological Resources



(b) Finding, Organizing, and Documenting the Knowledge



(c) Acquiring Knowledge from Outside Sources



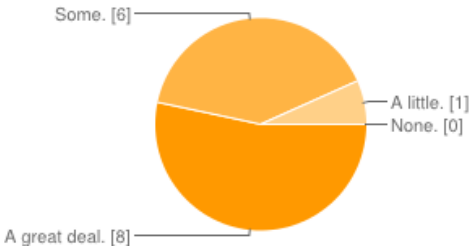
(d) Making Knowledge Accessible for Anytime, Anywhere

Figure 15. Knowledge Creation and Accessibility.

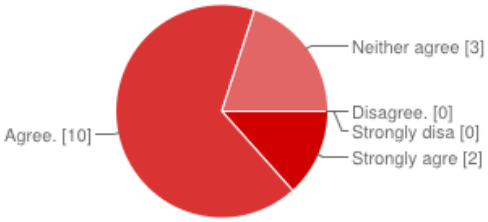
most widely used sources. Therefore, utilizing the Internet and web-based knowledge sharing tools, such as Bitbucket, Dropbox, Google Drive, TeamViewer, Skype, or Google Hangouts, was a very important research perspective for the Knowledge Temple experiment. Moreover, combining the power of the web with the competence of the participants' co-workers created an influential knowledge sharing culture. The participants also recommended workshops and hands-on studies for the open-ended other beneficial knowledge sources question.

In Section 2, the survey targeted the participants' knowledge sharing determination and knowledge lost perspective through the knowledge acquisition, dissemination, and maintenance processes.

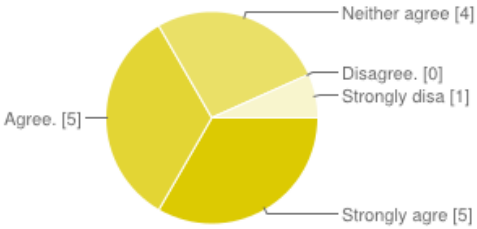
As shown in Figure 15, most of the participants (73%) thought iCORE's agile development team was good at creating new knowledge through its people and tech-



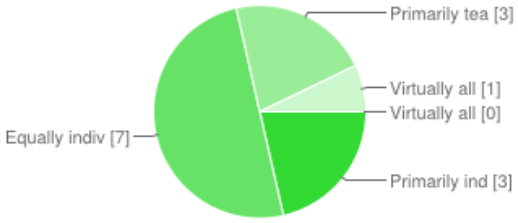
(a) Improving the Quality and Productivity due to Shared Knowledge



(b) Hoarding Knowledge because of Competitive Advantage Over Team Members



(c) Willing to Share the Knowledge Among the Team Members

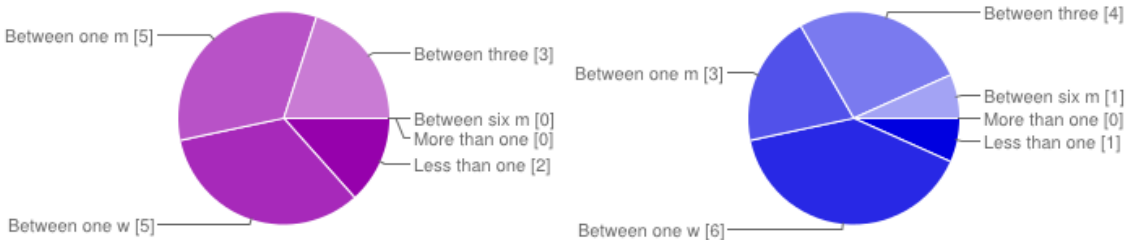


(d) Rewarding based on Individual Technical Accomplishments vs. Team Success

Figure 16. Knowledge Hoarding Effects.

nical resources. However, they (47%) argued that the team was not adequate by means of finding, organizing, and documenting the knowledge possessed through the Knowledge Temple technique. Although 53% of the participants were satisfied with the knowledge creation process, the percentage was expected to be closer to 80%; therefore this is an area that requires more research. The analysis supports both effective knowledge acquisition from outside sources and effective knowledge accessibility. Utilizing web-based repositories and web-based communication allowed a continually approachable environment.

Testing the effects of the Knowledge Temple technique on knowledge hoarding problems was essential considering influential knowledge sharing. As shown in Figure 16, the impact of knowledge sharing among the iCORE team members was highly acknowledged. Analysis of development quality and productivity assessment



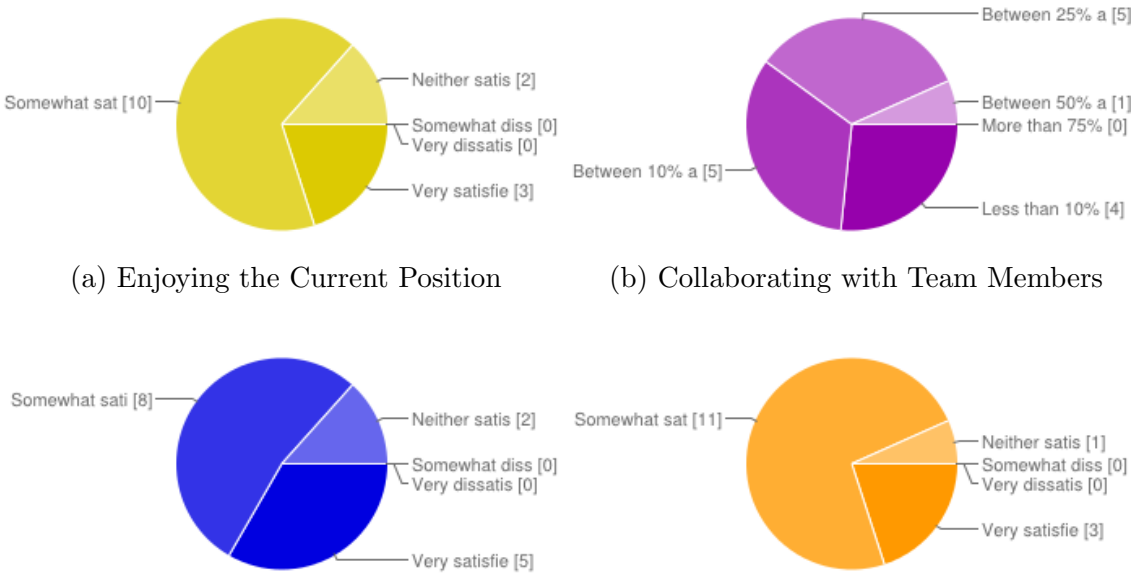
(a) Compensating the Knowledge Loss due to Team Member Turnover

(b) Compensating the Knowledge Loss due to Senior Technical Member Turnover

Figure 17. Knowledge Loss Effects.

question showed that 93% of the survey participants improved their development abilities due to knowledge sharing. Although participants (79%) stressed that their unique knowledge enhanced their competitive advantage over their peers, 66% of the participants gladly agreed to share knowledge with the agile development team. Moreover, the flexible productivity and knowledge sharing culture of the Knowledge Temple technique created an extraordinary workplace where teams and individuals worked simultaneously. The results of the question about the performance rewarding process showed that 50% of the participants determined team success and individual accomplishment are equally effective. Counterbalancing this, an equal amount of participants argued the case considering "primarily individual accomplishments, but also some team success" (21%) or "primarily team success, but also some individual accomplishments" (21%).

Experiencing mandatory employee turnover was a challenge for the Knowledge Temple experiment; however, the experiment participants presented courage and confidence through compensation of knowledge loss due to team member turnover (Figure 17). According to the results, team members can compensate for knowledge loss of less than a month (46%) or between one month and three months (33%). Moreover, the effect of losing an expert member did not change the trust of residual



(a) Enjoying the Current Position (b) Collaborating with Team Members

(c) Working in a Small Team with 2 Peers (d) Working in a Small Team with a Peer

Figure 18. Demographical Workplace Information.

agile development team members. The team members (47%) admitted to covering the knowledge loss due to expert member turnover.

In Section 3, the sociological factors in the workplace were evaluated. The participants (87%), who performed the Knowledge Temple technique, were satisfied with their working environment. As shown in Figure 18, 66% of the participants presented collaborative work between 10% and 50% of their daily working hours. Moreover, the agile development team members felt confident in a working environment both with a peer (93%) and with 2 peers (86%).

6.3 Observational Results

Experiencing the drawbacks of pair programming changed the development perspective. The programming level difference of the agile development team members did not allow for the application of pair programming successfully. However, the power

of pair programming was not undervalued. The development team members required a flexible working environment where they could accomplish both application development and knowledge sharing. The iCORE environment empowered the Knowledge Temple technique with its nature. Having different levels of programmers facilitated a working environment as a team of three. Therefore, the expert developer could continue development and share knowledge. At the same time, the average and novice developers could build knowledge and contribute to application development. Having three developers in a team, influenced by the expert developer, allowed much more adaptable and responsive team work.

In the Knowledge Temple experiment, one of the most important decisions was selecting the Temple Master. The Temple Master was responsible for the Temple and controlled the Temple through guidance. Every Temple had its own rules and way of accomplishing requested job duties. However, the Temple Master was the one who ensured the productivity of the project and knowledge sharing progress among the team. This management allowed the Temple Apprentices to contribute more while they were learning through their own efforts, pair studies, or Temple unification.

The selected theme, Star Wars^{TM}, notably increased the motivation of the Temple Apprentices. The idea of being Yoda was a big impulse compared to being a leader or a master for a team. It is worth noting that Star Wars^{TM} may not always be the best theme for any development environment. Therefore, another theme could be selected if required. However, it is important to choose a theme that can conceptualize the hierarchy, the mechanism, and the communication of the Knowledge Temple technique.

The unique environment of iCORE offered a high employee turnover through graduation of the team members; therefore, it was difficult to observe the effects

of the Knowledge Temple technique for employee turnover. However, the graduated members reported that they felt the loss of the team working environment at iCORE.

Another challenge in iCORE was the tight deadlines of the agile projects. Therefore, the Temple of three experts or the Temple of one expert, one average, and one novice were created for most of the projects. The Temple of three experts hastened the development speed of the projects and assisted Temple Master growth for different Temples. On the other hand, the Temple of one expert, one average, and one novice enhanced the productivity and knowledge sharing simultaneously. It was also the best fit for the varied levels of iCORE developers. The Temple of one expert and two novices was also utilized in the Knowledge Temple experiment. This Temple style enabled the knowledge sharing and active learning for the novice developers; however, productivity decrease was reported by Temple Masters. Therefore, building the Temple was an essential part of the Knowledge Temple technique requiring a good knowledge level observation among the development team members. Moreover, the project requirements influenced the Temple building process through appropriate developer selection.

The Knowledge Temple technique built a working culture for iCORE. The hybrid setting of the Knowledge Temple technique streamlined the agile development team members by the adaptation process. The amenity of the Knowledge Temple mechanism simplified the knowledge sharing process. Traditionally, newcomers hoarded the knowledge that they possessed through application development; however, the iCORE culture oriented all the team members to team success rather than individual accomplishments.

Finally, small agile development teams require internal growth from their developers in the areas of development continuity and quality, due to the difficulty in

hiring external competent developers. The Knowledge Temple technique facilitates the team to share and build knowledge between team members. Having three different zones to communicate, contemplate, and develop escalates the growth process of successful developers for small agile development teams.

CHAPTER 7

CONCLUSION AND FUTURE RESEARCH

Despite the productive, flexible, and adaptive nature of agile development, it may suffer from knowledge sharing limitations. This includes knowledge loss due to retirement or high turnover rates of skilled professionals and knowledge hoarding due to interpersonal or organizational climate. Thus, the internal growth of developers is highly desirable for a small development team to maintain production quality. The influence of pair programming for software development and knowledge sharing is respected. However, this technique is confronted by time-sharing issues, due to attempting to perform a number of tasks concurrently; motivational loss issues, due to pair level difference; and focus shift to separated tasks instead of a common goal, due to tight deadlines.

The Knowledge Temple was proposed as a knowledge sharing technique for small agile software development teams that supports both software development productivity and knowledge exchange between team members. The Knowledge Temple is a cognitive apprenticeship model, where every Temple has three members: one Temple Master and two Temple Apprentices. There are three zones where Temple members can perform software development and knowledge sharing methods, such as on-the-job-training, solo programming, pair programming, parallel peer programming, pair rotation, and knowledge repository creation.

A single-blind experiment was performed with The Innovation in Computing Research (iCORE) at Texas A&M University-Corpus Christi. Almost all of the development team members were part-time working university students either undergraduate or graduate level. The Knowledge Temple technique was administered

in three different projects with ten varied Temples. To evaluate this empirical thesis study, Temple member's development contributions, a Knowledge Temple questionnaire, and observational outcomes were utilized. The results of the Knowledge Temple experiment illustrated:

- development priority for Temple Masters,
- knowledge sharing availability for Temple Masters,
- knowledge sharing priority for Temple Apprentices,
- development support opportunity for Temple Apprentices,
- flexible scheduling for both development and knowledge sharing processes,
- inspirational small team fashion, and
- motivation continuity through Temple member availability.

Consequently, team member contribution, questionnaire results, and observational results yielded significant evidence that the Knowledge Temple technique for small agile development teams is an effective means of software development and knowledge sharing simultaneously. Moreover, the iCORE team members noted that they enjoyed the experience and declared that their technical skills had been increased. However, this empirical study alone is insufficient to validate the reported benefits of this knowledge sharing and development style. The Knowledge Temple technique should be performed as a teaching technique in academia to evaluate the influence on future generations. In addition, as mentioned previously, a higher number of participants were expected to be satisfied with the knowledge creation process; however, only 53% were satisfied with this process. Therefore, this difference needs

further exploration. Finally, examining the proposed technique in the industry with full-time workers is another way to comprehend the collaborative and cooperative effects of the Knowledge Temple technique.

REFERENCES

[1] Abbattista, F., Calefato, F., Gendarmi, D., and Lanubile, F. Incorporating 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 Information and Communications Technology (QUATIC), 2010 Seventh International 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. *Software, 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 technologies in software engineering with respect to developers' skill level. In *Evaluation 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 Engineering (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 Defense Software Engineering,(Dec* (2003).

[12] Briggs, J. "star wars", model making, and cultural critique: A case for film study in art classrooms. *Art Education 62*, 5 (2009), 39 – 45.

[13] Chatti, M., Schroeder, U., and Jarke, M. Laan: Convergence of knowledge management and technology-enhanced learning. *Learning Technologies,*

IEEE Transactions on 5, 2 (2012), 177–189.

[14] Chau, T., Maurer, F., and Melnik, G. Knowledge sharing: agile methods vs. tayloristic methods. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on* (2003), pp. 302–307.

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

[16] 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.

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

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

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

[20] Duka, D. Agile experiences in software development. In *MIPRO, 2012 Proceedings of the 35th International Convention* (2012), pp. 692–697.

[21] Dyba, T., and Dingsoyr, T. What do we know about agile software development? *Software, IEEE 26*, 5 (2009), 6–9.

[22] Fowler, M., and Highsmith, J. The Agile Manifesto. *Software Development Magazine 9(8)* (Aug. 2001). http://agilemanifesto.org.

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

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

[25] 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.

[26] 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.

[27] Huang, M., and Sun, B. Research on modeling and implementating of knowledge management system in virture enterprise. In *Machine Learning and Cybernetics, 2009 International Conference on* (2009), vol. 3, pp. 1424–1428.

[28] 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.

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

[30] Izquierdo-Cortazar, D., Robles, G., Ortega, F., and Gonzalez-Barahona, J. Using software archaeology to measure knowledge loss in software projects due to developer turnover. In *System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on* (2009), pp. 1–10.

[31] 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.

[32] 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.

[33] Jones, K., Kristof, D., Jenkins, L., Ramsey, J., Patrick, D., Burnham, S., and Turner, I. Collaborative technologies: Cognitive apprenticeship, training, and education. In *Collaborative Technologies and Systems, 2008. CTS 2008. International Symposium on* (2008), pp. 452–459.

[34] Kapell, M., and Lawrence, J. *Finding the Force in the Star Wars Franchise: Fans, Merchandise, and Critics*. Popular culture and everyday life. Peter

Lang Pub Incorporated, 2006.

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

[36] Kitchenham, B., Pfleeger, S., Pickard, L., Jones, P., Hoaglin, D., El Emam, K., and Rosenberg, J. Preliminary guidelines for empirical research in software engineering. *Software Engineering, IEEE Transactions on 28*, 8 (2002), 721–734.

[37] 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.

[38] 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.

[39] 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 engineering* (New York, NY, USA, 2005), HSSE '05, ACM, pp. 1–5.

[40] 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.

[41] 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.

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

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

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

[45] 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.

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

[47] 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.

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

[49] Poff, M. *Pair Programming to Facilitate the Training of Newly-hired Programmers.* Florida Institute of Technology, 2003.

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

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

[52] Roberts, A. Culture, identities and technology in the star wars films: Essays on the two trilogies. *SCIENCE-FICTION STUDIES 35* (n.d.), 156 – 159.

[53] Rong, J., Hongzhi, L., Jiankun, Y., Tao, F., Chenggui, Z., and Junlin, 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.

[54] 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.

[55] 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.

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

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

[58] Savolainen, J., Kuusela, J., and Vilavaara, A. Transition to agile development - rediscovery of important requirements engineering practices. In *Requirements Engineering Conference (RE), 2010 18th IEEE International* (2010), pp. 289–294.

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

[60] 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.

[61] Shaw, M. What makes good research in software engineering? In *Presented at the European Joint Conference of Theory and Practice of Software (ETAPS 2002), Grenoble, France. To appear in the International Journal on Software Tools for Technology Transfer.* (2002).

[62] Sillitti, A., Succi, G., and Vlasenko, J. Understanding the impact of pair programming on developers attention: A case study on a large indus-

trial experimentation. In *Software Engineering (ICSE), 2012 34th International Conference on* (2012), pp. 1094–1101.

[63] 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.

[64] 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.

[65] 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.

[66] 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.

[67] 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.

[68] Tang, A., de Boer, T., and van Vliet, H. Building roadmaps: a knowledge 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.

[69] 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.

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

[71] 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.

[72] Weyuker, E. Empirical software engineering research - the good, the bad, the ugly. In *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on* (2011), pp. 1–9.

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

[74] 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.

[75] 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.

[76] 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.

[77] 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.

[78] 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.

APPENDIX A

KNOWLEDGE TEMPLE QUESTIONNAIRE

From: Ilhan Burak Ersoy - ilhan.ersoy@tamucc.edu

Introduction: I am a graduate student in Computer Science at Texas A&M University-Corpus Christi. As part of my Master's thesis research, I am gathering information about the way knowledge is acquired, disseminated, and maintained by companies and organizations in technology research, development, and service. I am interested in obtaining your input on this subject through this survey. Your participation in this survey will be anonymous, and no one will contact you for additional information or solicitation. The survey takes approximately 15 minutes to complete.

Responses will be collected until Tuesday, July 3rd, 2013.

Section 1: Knowledge Sources

When a person requires some additional knowledge or information to get their job done, they sometimes turn to various sources for the information they are seeking. When you are faced with that situation, how often do you find the information you are looking for in the following sources:

1) Internal publications (such as guidelines or "How-To" documents, not design documents) produced by your software development team.

( ) My software development team does not produce internal publications of this sort.

( ) Often.

( ) Sometimes.

( ) Rarely or never.

2) Design documents, from either the project at hand or a previous project, produced by your software development team.

( ) My software development team does not produce design documents.

( ) Often.

( ) Sometimes.

( ) Rarely or never.

3) External publications, such as books, manuals, journals, or magazines.

( ) My software development team does not permit the use of externalpublications.

( ) Often.

( ) Sometimes.

( ) Rarely or never.

4) The Internet.

( ) My software development team does not permit the use of the Internet.

( ) Often.

( ) Sometimes.

( ) Rarely or never.

5) Co-workers within my immediate team or organization.

( ) I do not work with others.

( ) Often.

( ) Sometimes.

( ) Rarely or never.

6) Courses, either classroom or online.

( ) My software development team does not provide a means for allowing itsemployees to take courses.

( ) Often.

( ) Sometimes.

( ) Rarely or never.

7) A database or groupware, such as Bitbucket, Dropbox, Google Drive, TeamViewer, Skype, or Google Hangouts.

( ) My software development team does not use databases or groupware torecord knowledge or skills-related information.

( ) Often.

( ) Sometimes.

( ) Rarely or never.

8) Other (Please specify with rank as Often/Sometimes/Rarely or never):

.....

.....

.....

.....

.....

Section 2: Knowledge Acquisition, Dissemination, and Maintenance

9) My software development team is good at creating new knowledge through its people and technological resources.

( ) Strongly agree.

( ) Agree.

( ) Neither agree nor disagree.

( ) Disagree.

( ) Strongly disagree.

10) My software development team is good at finding, organizing, and documenting the knowledge it already possesses.

( ) Strongly agree.

( ) Agree.

( ) Neither agree nor disagree.

( ) Disagree.

( ) Strongly disagree.

11) My software development team is effective at acquiring knowledge from outside sources, such as consultants or products.

( ) Strongly agree.

( ) Agree.

( ) Neither agree nor disagree.

( ) Disagree.

( ) Strongly disagree.

12) My software development team is effective at making knowledge accessible to those who need it, when they need it.

( ) Strongly agree.

( ) Agree.

( ) Neither agree nor disagree.

( ) Disagree.

( ) Strongly disagree.

13) Does your software development team have personnel in place specifically responsible for managing knowledge (e.g., Chief Knowledge Officer, Knowledge Project Manager, Knowledge Management Specialist, Knowledge Team, etc,)?

( ) Strongly agree.

( ) Agree.

( ) Neither agree nor disagree.

( ) Disagree.

( ) Strongly disagree.

14) How much improvement in the quality and productivity of your work could be gained by improvements to your software development team's management of knowledge?

( ) Strongly agree.

( ) Agree.

( ) Neither agree nor disagree.

( ) Disagree.

( ) Strongly disagree.

15) I feel/felt the unique knowledge I possess enhances my competitive advantage over my peers when it comes to job promotions, leadership opportunities, and awards.

( ) Strongly agree.

( ) Agree.

( ) Neither agree nor disagree.

( ) Disagree.

( ) Strongly disagree.

16) I am/was willing to share all of the knowledge I possess relevant to my job with my software development team.

( ) Strongly agree.

( ) Agree.

( ) Neither agree nor disagree.

( ) Disagree.

( ) Strongly disagree.

17) In your software development team, how much are rewards based on individual technical accomplishments, versus team success?

( ) Virtually all individual accomplishments.

( ) Primarily individual accomplishments, but also some team success.

( ) Equally individual accomplishments and team success.

( ) Primarily team success, but also some individual accomplishments.

( ) Virtually all team success.

18) If you were to leave your position tomorrow, how long would it take your software development team to compensate for the loss of knowledge you possess? (Please make your best guess)

( ) Less than one week.

( ) Between one week and one month.

( ) Between one month and three months.

( ) Between three months and six months.

( ) Between six months and one year.

( ) More than one year.

19) If the most senior technical member of your software development team were to leave his/her position tomorrow, how long would it take your software development team to compensate for the loss of knowledge he/she possesses? (Please make your best guess)

( ) Less than one week.

( ) Between one week and one month.

( ) Between one month and three months.

( ) Between three months and six months.

( ) Between six months and one year.

( ) More than one year.

Section 3: Demographical Background Information

20) How long have you been developing software?

( ) I am not a programmer in industry or research.

( ) Less than one year.

( ) Between one year and two years.

( ) Between two years and five years.

( ) Between five years and ten years.

( ) More than ten years.

21) How long have you been with your software development team?

( ) Less than one year.

( ) Between one year and two years.

( ) Between two years and five years.

( ) Between five years and ten years.

( ) More than ten years.

22) How satisfied are you with your job?

( ) Very satisfied.

( ) Somewhat satisfied.

( ) Neither satisfied nor dissatisfied.

( ) Somewhat dissatisfied.

( ) Very dissatisfied.

23) Approximately how much of your day is spent collaborating with others?

( ) Less than 10%.

( ) Between 10% and 25%.

( ) Between 25% and 50%.

( ) Between 50% and 75%.

( ) More than 75%.

24) How satisfied are you working in a small team with 2 peers?

( ) Very satisfied.

( ) Somewhat satisfied.

( ) Neither satisfied nor dissatisfied.

( ) Somewhat dissatisfied.

( ) Very dissatisfied.

25) How satisfied are you working in a small team with a peer??

( ) Very satisfied.

( ) Somewhat satisfied.

( ) Neither satisfied nor dissatisfied.

( ) Somewhat dissatisfied.

( ) Very dissatisfied.