3-2010

Designing an Information Systems Development Course to Incorporate Agility, Flexibility, and Adaptability

Chuan-Hoo Tan

City University of Hong Kong, tancho@comp.nus.edu.sg

Wee-Kek Tan

National University of Singapore

Hock-Hai Teo

National University of Singapore

Follow this and additional works at: <https://aisel.aisnet.org/cais>

Recommended Citation

Tan, Chuan-Hoo; Tan, Wee-Kek; and Teo, Hock-Hai (2010) "Designing an Information Systems Development Course to Incorporate Agility, Flexibility, and Adaptability," *Communications of the Association for Information Systems*: Vol. 26 , Article 10.

DOI: 10.17705/1CAIS.02610

Available at: <https://aisel.aisnet.org/cais/vol26/iss1/10>

This material is brought to you by the AIS Journals at AIS Electronic Library (AISeL). It has been accepted for inclusion in Communications of the Association for Information Systems by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.

Communications of the Association for Information Systems

Designing an Information Systems Development Course to Incorporate Agility, Flexibility, and Adaptability

Chuan-Hoo Tan

Department of Information Systems, City University of Hong Kong

ch.tan@cityu.edu.hk

Wee-Kek Tan

Department of Information Systems, National University of Singapore

Hock-Hai Teo

Department of Information Systems, National University of Singapore

Abstract:

Managing changing business environments and requirements in Information Systems (IS) development is becoming increasingly important. IS development project courses could thus benefit from the infusion of agile, flexible, and adaptable processes. This article reports two years of continuous effort in refining the IS development project course in the National University of Singapore to illustrate how undergraduate students could be effectively taught the agile IS development approach. The course design process was based on the system approach model to instructional design and followed the principles of instructional events suggested by Dick and Carey [1991] and Gagne et al. [1992]. Input from students, as well as pedagogic principles drawn from the extant literature, were used to rigorously refine the course design over time. The final course design exhibits five distinctive characteristics. The two most important characteristics were (1) the adoption of a hybrid agile methodology incorporating the best practices in both coding and project management drawn from three established agile methodologies, and (2) the introduction of requirement shocks at appropriate junctures to train students to react to changes in business requirements. The effectiveness of the course design was assessed using objective measures of learning operationalized as the numerical scores obtained by students. Statistical analysis results indicate that the students who were taught using the final course design obtained higher scores than those who were taught with the initial design. Our course design effort was, therefore, deemed successful.

Keywords: agile information systems development, course design

Volume 26, Article 10, pp. 171-194, March 2010

The manuscript was received 9/27/2008 and was with the authors 4 months for 3 revisions.

I. INTRODUCTION

The business environment is constantly changing, and the issue of whether an Information Technology (IT) solution meets users' requirements is no longer dependent on whether the ultimate product delivered conforms to the initial plan, but more importantly whether it satisfies the customers at the time of system delivery [Lyytinen, 2005]. Commensurate with this trend, we propose that Information Systems (IS) development project courses for undergraduate students could benefit from the infusion of agile, flexible, and adaptable development processes [see, for instance, Allen et al., 2003; Hedin et al., 2003].

Specifically, student developers should learn to apply the concepts of agility, flexibility, and adaptability [Reichlmayr, 2003] to be more responsive in today's dynamic business technology environment. This is because developers using the rigid and time-consuming traditional Systems Development Life Cycle (SDLC) [Avison and Fitzgerald, 2003] are often unable to make changes to the system during the later development stages [Morien, 2005]. In comparison, students adopting the agile approach have been able to react promptly to changes in user's requirements, thus producing higher quality system [Muller and Tichy, 2001]. One of the key reasons is that the agile approach emphasizes the delivery of a working code and effective user communication over exhaustive documentation [Lindvall et al., 2002]. Consequently, students will not be over-burdened with documentation tasks, but can instead focus on the coding and testing tasks.

This article summarizes two years of our experience (from January 2006 to November 2007) in conducting an IS development project course that seeks to prepare our IS undergraduate students for the increasingly dynamic business environments of today through emphasizing the concepts of agility, flexibility, and adaptability. In particular, our proposed IS development course is based on the four agile principles [Beck et al., 2001]: (1) individuals and interactions over processes and tools; (2) working software over comprehensive documentation; (3) user collaboration over contract negotiation; and (4) responding to change over following a plan [Schuh, 2005].

The system approach model of instructional design [Dick and Carey, 1991] provided the theoretical foundation for our agile course design process. In addition, we took into consideration the feedback provided by the students themselves, as well as additional pedagogic principles drawn from the extant literature. This rigorous approach produced the final version of our course design, which exhibits five distinctive characteristics. These include (1) the adoption of a hybrid agile methodology incorporating the best practices in both coding and project management drawn from three established agile methodologies; (2) the introduction of requirement shocks, i.e., changes to the initial set of requirements stipulated in the project specifications at appropriate junctures, to train students to react to rapid changes in business requirements; (3) team-based guidance emphasizing close collaboration between students and the team advisors acting as business users; (4) small team size of between six and seven students performing concurrent tasks; and (5) emphasis on iterative and incremental delivery of integrated working software solutions over comprehensive documentation. These characteristics are consistent with the general agile principles.

Before we proceed, it is to be noted that the course to be presented is applicable to business schools, information schools, engineering schools, and/or computing schools where the students are technically competent. Should schools wish to embark on conducting this course, a good set of preparatory courses in programming and software engineering should first be put in place.

The remaining sections of the paper are presented as follows. First, we present a critical review of the relevant theoretical background. Next, we offer an overview of the course. Subsequently, we provide a detailed description of the course design process before concluding with discussions on the course and its design process.

II. THEORETICAL BACKGROUND

Information Systems Development Course Design

Developing large scale IS in support of organizational business processes is a critical skill that is fundamental to any IS educational program [Avison and Cole, 2006; Topi et al., 2007]. Researchers and educators alike have, therefore, taken a keen interest in deriving a set of best practices in IS development course and project work design.

At a macro level, prior research has focused on two pertinent issues. First, it is necessary to provide students with real world IS development experience in order to inculcate in them the required complex skills sets when they enter

the workforce [Jones and McMaster, 2004]. For instance, capstone project courses involving real business clients have been carefully designed to facilitate close interaction between students and clients leading to the successful delivery of workable business solutions [Allen et al., 2003; Van Vliet and Pietron, 2006]. Second, other educators [e.g., Farhoomand, 2004; Hackney et al., 2003; Wang, 2003] have also advocated the use of the case study approach in designing IS development courses that simulate real-world problem solving to induce high-order thinking among students. This proves useful when real world projects are not accessible. At a more micro level, course design efforts have focused on specific fragments of the overall IS development process. These include the choice of data modeling techniques [Golden and Matos, 2006], system analysis and design techniques [Mahapatra et al., 2005], and development technologies [Meso and Liegle, 2005].

Despite these efforts, there remains a paucity of research examining IS development course design with respect to the IS development methodology to be adopted [McGann et al., 2007], with a few notable exceptions. For instance, although Van Vliet and Pietron [2006] noted the increasingly prevalent use of modern agile methodologies, such as eXtreme Programming (XP), they reasoned that traditional SDLC continues to be an appropriate choice for teaching IS development. Other educators who have embraced agile methodologies have, unfortunately, focused largely on adopting specific techniques and principles of popular agile methodologies, such as XP [Allen et al., 2003; Hedin et al., 2003; Kivi et al., 2000; Laplante, 2006]. Such an approach often fails to address the inherent deficiencies of agile methodologies, such as poorly maintained documentation, limited ability to build reusable software artifacts, and an overly flexible process that might not be appropriate for inexperienced student developers [Schneider and Johnston, 2003; Turk et al., 2002]. A more holistic framework would seek to leverage the advantages of agile methodologies while addressing these deficiencies. In brief, a comprehensive course design framework for IS development that revolves around IS development methodology is lacking.

This pressing issue needs to be addressed given that the majority of our IS graduates are inadequately trained in applying a methodological approach to IS development [Kim et al., 2006], even though it is essential in producing quality systems [Necco et al., 1987]. Accordingly, we propose a holistic agile IS development course that is geared toward training students to develop large-scale systems in a dynamic environment that is characterized by rapidly changing business requirements [Lyytinen, 2005]. Specifically, our course design incorporates a hybrid agile methodology together with several course design principles and pedagogical techniques intended to leverage the benefits of the various agile methodologies adopted while addressing their deficiencies [Schneider and Johnston, 2003; Turk et al., 2002]. In the next section, we review several agile methodology concepts that are relevant to our present course design effort.

Agile Practices and Education

The agile software development approach has received significant academic attention in recent years due to its widespread application in the commercial world [Nerur and Balijepally, 2007; Mathiassen and Pries-Heje, 2006]. In particular, recent surveys of industry practitioners on the adoption rate of agile methodologies ranges from a moderate level of 46 percent [Davidson, 2008] to a high level of 69 percent [Ambler, 2008]. Beyond simple adoption, the majority of the respondents who are currently using at least one agile methodology has indicated that the agile approach leads to higher productivity (82 percent) and better system quality (77 percent) while lowering the overall cost of system development (72 percent) [Ambler, 2008]. One of the most important success factors of agile methodologies is that they are able to facilitate an increasingly prevalent dynamic process of software development in the industry [Nerur and Balijepally, 2007]. This dynamic process demands rapid and short iterative cycles coupled with the active involvement of all stakeholders in order to deliver software that truly meets the needs of business organizations. It is thus imperative for educators to impart the requisite skills of agile methodology to undergraduate students.

The agile software development methodology posits that a software development team can be more effective in responding to changing environments and user requirements if it capitalizes on its members' skills, stresses collaboration among members, and maintains a close working relationship with system users. Furthermore, a good project should be characterized by functional modularity, short iteration cycles (of one to six weeks per cycle) to enable fast verifications and corrections, removal of all unnecessary activities (e.g., redundant documentation), and an incremental rather than radical approach toward software development [Schuh, 2005].

Agile software development methodology has been increasingly adopted as a replacement for the traditional waterfall model for imparting programming and software engineering skills to students. The typical classroom settings involve students either in pairs or small-to-medium groups, applying various aspects of a chosen agile methodology to complete a software development project. Our review of the extant literature indicates that XP has been the preferred agile methodology of choice. For instance, XP has been used successfully to teach undergraduates production programming by focusing on the usage of user stories, pair programming, unit testing and continuous refactoring, and close collaboration with onsite customers [Allen et al., 2003]. At the small-group

level, XP has proven to be effective in aiding students to deliver a statistical analysis tool for software metrics to reasonably satisfied customers with low defect rates [Kivi et al., 2000]. Furthermore, it is reported that students working on a software engineering project in a medium-sized group setting have been able to develop a good understanding of fundamental problems and techniques in software development [Hedin et al., 2003]. The same group of students also felt satisfied with the use of XP although they noted that the course was intense.

While agile methodology such as XP has been applied to phase out the waterfall model, XP has not been without its critics. In fact, some educators have noted that system development methodologies employed in practice are based on the SDLC, and thus activities performed during SDLC by students in classroom settings continue to be relevant to real world work [Van Vliet and Pietron, 2006]. Moreover, Schneider and Johnston [2003] have suggested that the adoption of agile development practices in tertiary education might not be a straight-forward task, as the agile practices could contradict educational goals or prove difficult to adjust to a learning environment. For instance, pair programming might not benefit the weaker students, i.e., agile methodology might not be suitable for students who lack the necessary experience and maturity of working professionals. Schneider and Johnston [2003] suggested an emphasis on educating students on how to select and tailor a process that suits their needs.

Hence, we advocate that while following the specific methodology (e.g., XP) to teach agile principles is feasible, the specific methods may fail in fitting well with different software developments situations, making fine-tuning or customizing the existing methods inevitable. A more viable approach is to impart to the students a conceptual understanding of agile principles and practical knowledge for applying them. To this end, an incremental approach is to be taken to incorporate the essence of the agile practices into the course design. Specifically, we reference the principles of agile practices that are most promising for our course: (1) mandating multiple releases of the system, (2) increasing the amount of face-to-face communication with students and personalized team-based guidance for them, (3) maintaining continuous efforts toward technical excellence and good system design for catering to requirement changes, and (4) emphasizing the need for a simple, workable and integrated system.

Theories of Instructional Design

Incorporating the various principles of agile practices into a coherent course design that meets our pedagogy objectives of imparting agility, flexibility, and adaptability in IS development to students is a complex task. In our present context, the theories of instructional design [Dick and Carey, 1991; Gagne et al., 1992] are used to guide us in the course development process.

Instruction can be conceptualized as a set of nine events that facilitates the learning process [Gagne et al., 1992, pp. 190]. The learning process itself can be viewed as a form of information processing that involves interpreting the stimulus in the learning environment and deliberating it in the human's short-term memory before a response action can be generated [Gagne and Driscoll, 1988]. In this sense, instruction is closely related to the learning process because it involves providing stimulus material as a form of guidance for learning that directs the learner toward performing the desired learning task [Gagne et al., 1992]. The learning performance is then assessed and the learner is provided with some feedback.

The process of designing instruction emphasizes a systematic approach toward developing instructional specifications that entails the various instruction events [Moallem, 2001]. According to Gustafson and Branch [1997], theories specifying the instructional development process are mainly rectilinear in nature. The system approach model for designing instruction [Dick and Carey, 1991] is one such theory. Rectilinear theories typically begin with a situational assessment of the instructional task involved and a delineation of the instructional goals, followed by the actual instructional development [Gagne et al., 1992]. The instructional development process involves the specifying of the instructional needs and goals, the performance assessment criteria and objectives, the actual instructional strategy for achieving the predefined performance objectives, as well as the selecting of the appropriate media for the instructional material. The process then transits into a series of formative evaluations intended to refine the various portions of the instruction. The entire process concludes with a final summative evaluation that is performed to obtain an overall assessment of the entire instruction. However, Gustafson and Branch [1997] noted that the rectilinear approach is unable to factor in the inherent complexity of instruction development and instead propose the curvilinear model, which uses a multiple iterations approach.

This line of reasoning is consistent with the use of rapid prototyping for instructional design [Tripp and Bichelmeyer, 1990]. Utilizing this approach, the instructional designer would present an initial design to potential learners so that the strengths and weaknesses of the current design are discovered, i.e., by performing formative evaluation [Gagne et al., 1992]. Based on the feedback from the learners, the designer then refines the next design iteration until an optimal instructional design is achieved. In general, instructional design theories consist of the same general steps, including analysis, design, development, and evaluation [Savenye et al., 2001].

To this end, we adopt the various stages in the system approach model for designing instruction [Dick and Carey, 1991]. However, these stages are assembled in a curvilinear fashion [Gustafson and Branch, 1997] similar to the rapid prototyping approach [Tripp and Bichelmeyer, 1990]. Since the final summative evaluation takes place after the formative evaluation, in which the finer details of the instructional design have been refined [Gagne et al., 1992], we would expect the final variant of our agile IS development course design at the end of the fourth semester to be the most effective.

III. COURSE OVERVIEW

Course Background

The IS development project (ISD) is an advanced course offered in our Bachelor of Computing program in National University of Singapore. The course is mainly dominated by a team-based project with a handful of lectures delivered at the beginning of the semester. Typically, students take ISD in their third-year of study and would have already taken prerequisite courses in Java programming, data structures, and algorithms in Java, as well as an introductory course on developing an enterprise information system with Java Platform, Enterprise Edition (Java EE). Students taking ISD are, therefore, assumed to possess adequate Java programming language competency and related software engineering knowledge. The undergraduate students in our course are usually between the ages of nineteen and twenty-six, with a male to female ratio of 60 percent to 40 percent. A significant 30 percent of the cohort is international students. However, language and communication problems are insignificant, since English is the language of instruction and well understood.

The course has three objectives. First, the course provides a suitable avenue for students to practice what they have learned since their sophomore years. Second, the course presents an opportunity for the students to learn about multi-tiered software development architecture, which is common in many organizations. In this course, each team of five to six students is instructed to develop an enterprise-level system using Java EE. To set the stage for the students to build a multitiered system collaboratively for the first time, lectures are delivered to help them master the technology. Third, the course allows students to experience, for the first time, a simulated but realistic software development environment that exerts sufficiently tight time constraints on them to deliver the system within a stipulated short span of time. Each team is assigned an advisor who also acts as the customer interacting with the team members on the requirements of the project system.

Problems with the Legacy Course Design

In the previous semesters, students were instructed to develop an enterprise-level system through one run of the development lifecycle, i.e., the traditional waterfall-based SDLC. The SDLC is a systematic approach that dictates the multiple sequential steps (requirement analysis, system design, coding and debugging, and system testing) which teams take to develop a system. Given its stage-by-stage approach to software development, it is also recognized as the waterfall model. A software project team that functions using the waterfall model can be perceived as the sequence archetype with which the development is often characterized as a set of predetermined order of tasks and execution sequence [Sawyer, 2004].

At the conclusion of the semester, teams demonstrated their systems to one independent evaluator, and a post-mortem was held after the demonstration to highlight the areas for further improvement. Since the entire semester was spent in completing one development lifecycle, there was little opportunity for the students to improve their system based on the feedback. As it was the first time the students experienced enterprise-level system development, many teams faced several fatal problems. First, they spent nearly half of the semester in the analysis and design stages, and later discovered that their detailed design decisions made only a couple of weeks ago had become inconsistent with the code currently being generated. As teams scrambled to meet the system delivery deadline, the students paid less attention to system integration and testing; this resulted in the delivery of poor system quality. This problem draws a parallel with the “locking-in” phenomenon that requires all project requirements to be fully stated and documented before proceeding to actual system design and implementation [Morien, 2005]. This rigid approach does not cater to subsequent changes in the requirements and is one of the root causes of failure in commercial IS development projects [Montalbano, 2007].

Second, once the team had completed the initial system analysis and design phases, the programming tasks were assigned to individual team members, and the degree of communication among the members was dramatically reduced. Some of the teams reported that they met only at the end of the semester to integrate their individual features. Due to the adopted big-bang approach in system integration, the system quality fell below our expectations. This problem was more severe when the features within the system were tightly coupled.

To rectify the problems, we incorporated the iterative IS development approach using a hybrid agile methodology that required the students to have multiple system releases. This view is in accordance with the agile software development methodology [Bittner and Spence, 2006; Rajlich, 2006; Jalote et al., 2004] that emphasizes having several short iteration cycles to enable fast verification and correction (with each cycle running from one to six weeks), removing all unnecessary activities (e.g., redundant documentation), and focusing on an incremental rather than radical approach toward software development [Martin, 2003]. The main objective of having an incremental approach is to provide students with the practical experience of managing a stable, integrated and tested release of a system that is more aligned with the course requirements. Specifically, the advisors (i.e., tutors and/or the lecturer) are able to offer timely feedback only if students are able to showcase the system regularly with incremental delivery of features. In the next section, we discuss in detail our adopted hybrid agile methodology.

Hybrid Agile Methodology

The agile methodology adopted in our agile IS development course is a hybrid adaptation of three commonly used agile methodologies, namely Extreme Programming (XP), SCRUM, and Feature-Driven Development (FDD), which are presented in Figure 1. While XP focuses primarily on the programming aspect, the use of SCRUM (which is management-focused) and FDD (which is balanced between both aspects) ensures that the students can benefit from a more holistic methodology [Martin, 2003]. In other words, such a hybrid methodology is necessary as the students can benefit from the various changes injected into the course only by applying the unique features of each of the three adapted methodologies. Using this hybrid methodology, students are equipped with the necessary skills to perform in-depth analysis of the business domain and translate the business-focused project specifications into a set of coherent system features that are closely coupled.

As in the modern process model, our agile methodology adopts an iterative and incremental structure consisting of four key phases: (1) planning, (2) feature sub-task iteration, (3) executable release and evaluation, and (4) post-evaluation review. The planning phase essentially corresponds to the feasibility assessment and analysis stages in the traditional waterfall SDLC methodology. However, in our context we adopted the concept creation and requirement specification steps in the agile XP process. In this phase, the team performs a quick analysis of the system requirements that cumulates in the writing of user stories that are sufficient for commencing the development of the first executable release. This differs vastly from the feasibility assessment and analysis stages in traditional waterfall SDLC that requires a comprehensive requirement gathering process covering the entire system before the first line of code is written. More specifically, students would need to produce detailed requirement specifications that are likely to be modified due to changes that occur in the midst of the development phase [Highsmith, 2002]. This is particularly true for those requirements that are scheduled to be developed at the later stages. The user stories are then prioritized in terms of their importance in meeting the needs of the end users, together with an estimate of the efforts required to complete each feature. These steps are performed during the planning game. Finally, during the feature iteration planning stage, the team decides on the system features that will be developed in each of the iterations.

The output from the feature iteration planning step is used during the feature sub-task iteration phase to guide the team in scheduling their development work. The team develops the system features in an iteration consisting of three sub-tasks, namely designing, programming, and testing. More precisely, the team is further broken down into pairs of programmers, each fully responsible for completing all the sub-tasks required to deliver a specific feature. This is in agreement with the pair-programming concept in XP [Allen et al., 2003]. In summary, one of the key features distinguishing our agile teams from the traditional teams in the previous semesters is that the agile teams performed concurrent development work during the feature sub-task iteration phase.

The designing step involves the designing of conceptual data models, user interfaces, and software architecture, as well as test cases. The test cases are used for the unit and integration testing prior to the release of an executable. Each of the iterations is a short sprint of between one to three weeks, during which the entire team meets for a daily scrum, i.e., project status meeting in SCRUM, lasting no more than thirty minutes. The daily scrum takes place outside the scheduled lectures and consultations hours and are arranged by the team leaders. The daily scrum provides an opportunity for the team members to discuss the progress of their respective feature task lists and the problems encountered. Reshuffling of the task priority is based on consensus. Such intimate discussion sessions, which are built into our methodology, empower the team leader to better manage the progress of the team. This is another unique characteristic of our agile team compared to teams in the previous semesters which tended to meet on a weekly basis.

The output from the planning and feature sub-task iteration phases forms the bulk of the documentation to be submitted by the students. Following the experience of other educators, such as Laplante [2006], we do not require extensive documentation, but rather a single hybrid analysis and design report consisting of three sets of documentation items including (1) requirement analysis output in the form of user stories and feature lists and their

Integration of Agile Methodologies

Figure 1: Agile Methodology Adopted in the Course (Adapted and Modified from Schuh, 2005).

priorities; (2) system design output in the form of test cases and the testing results, as well as technical diagrams, e.g. essential unified modeling language (UML) diagrams limited to use case diagrams and class diagrams; and (3) project management Gantt chart. The contents of this report are developed gradually over each cycle of the planning and sub-task iteration phases. Prior to the current agile course design process, the previous semester of the ISD course used the traditional waterfall SDLC. In the waterfall approach, the students had to present detailed write-ups of the user requirements, existing and proposed solutions, detailed algorithm designs and detailed project management risk analysis. Comparatively, the amount of documentation required by the present hybrid agile methodology is substantially modest.

Toward the end of the testing step in the feature sub-task iteration, the executable release is evaluated by the teaching team to simulate the user acceptance testing process in XP. The evaluators perform a walk-through of the executable with the team to ascertain whether the requirements stated in the user stories have been met. The team is provided with actionable comments and suggestions on refining the system in the next iteration in a fashion similar to the sprint review meeting in SCRUM.

These comments and suggestions are then reviewed in the post evaluation review phase when the teams will update their existing design models. In the real business environment, the end user could request changes or

additional requirements after evaluating the release. This is addressed in our methodology by allowing the team to react to change requests, or what we term requirement shocks, through developing additional user stories and reconciling any conflict with the existing release. Essentially, the teams are given a chance to respond to customer feedback as mandated in XP. Finally, multiple cycles of the four phases are performed with each cycle cumulating in an executable release until the final system is completed. Each release should contain more features than the previous release incrementally and be consistent with the general prototyping process.

As illustrated by the insert in Figure 1, the third phase of feature sub-task iteration focuses on coding and constitutes roughly 70 percent of the project time that the students were expected to spend in each cycle. The remaining 30 percent of the time is spent on project management, which includes planning and preliminary system designing. Additionally, the fourth phase of post-evaluation review has been specifically earmarked to provide an opportunity for students to manage the changes in the user requirements. The exclusive reliance on XP, which focuses on coding, could sometimes lead to ineffective coding due to inadequate project management [Kivi et al., 2000]. Through the inclusion of project management elements from FDD and SCRUM, our hybrid methodology is more resilient to this deficiency of the agile approach.

The evaluation of the teams is based on their performance on a continuous assessment basis as well as on a final assessment. The continuous assessment includes a formal graded evaluation of the first system release, which constitutes 20 percent of the overall assessment, and an informal evaluation of the intermediate second system release. The final assessment constitutes the majority of the teams' scores, which include the system analysis and design report as well as the final system release, which constitutes 30 percent and 50 percent of the overall assessment respectively.

Course Design Principles

Three general principles were adopted in designing our agile IS development course: (1) emphasizing the in-depth application of our hybrid agile methodology in its entirety instead of selectively choosing only a few practices; (2) creating additional pedagogical techniques to complement the hybrid agile methodology; and (3) ensuring that our course design is suitable for use with agile methodology in general. Each of the three principles is discussed in greater detail in the following paragraphs.

Previous researchers have noted that learning about a comprehensive development methodology at the initial stage is a necessity for inexperienced developers such as students [Van Vliet and Pietron, 2006]. Proper introduction to a comprehensive methodology enables inexperienced developers to eventually acquire sufficient knowledge to adopt various aspects of that methodology to different situational needs. Thus, in the spirit of ensuring that our students are able to apply the various agile practices knowledgeably when they enter the work force, the first principle involves placing an emphasis on a comprehensive development approach rather than just applying particular practices. Specifically, the preceding hybrid agile methodology is applied deliberately into the course design that mandates three distinct system releases. In other words, students follow the methodology rigorously through three cycles, cumulating in the final system release.

Applying the hybrid agile methodology, however, does not present a foolproof course design solution for undergraduates learning ISD for the first time. We identified several critical problems faced by students during the formative evaluation stage of the course design process [Gagne et al., 1992], which was conducted at the end of each semester. These problems were generally associated with students' unfamiliarity with the agile development approach. For instance, students were expected to work more closely with each other in the agile approach, compared to the traditional waterfall SDLC on tasks such as pair programming and daily scrum. Consequently, we observed that team conflicts were more prevalent when students used the agile approach. Accordingly, we devised an additional pedagogical solution involving lectures on team conflict management. Thus, the second principle involves deriving additional pedagogical techniques to complement the agile approach.

Last, the third principle involves using the Critical Adoption Factors for Agile Methodology (developed by McAvoy and Sammon [2005]) to design our course. This methodology identifies a set of decision factors that are used to determine the feasibility of adopting agile methodology for a particular software development project. We employed this methodology to assess the feasibility of adopting agile methodology in our classroom IS development project. There are eleven factors classified along the following dimensions: (1) project dimension—duration of project, acceptance of changes to requirements, and criticality of project; (2) team dimension—team size and skill of team; (3) customer dimension—location of customer and the degree of customer involvement; and (4) organizational dimension—organization and reporting structure, process, documentation requirements, and layout of workspace. The methodology was empirically validated both in an academic and actual industry settings. We adopted the first seven factors covering the project, team and customer dimensions that are more relevant to our context. The last four factors concerning the organizational dimension were not considered, given our classroom context.

Briefly, in terms of the project dimension, our course spans thirteen weeks, with students having to cope with the workload of other modules concurrently. Thus, thirteen weeks may be considered sufficiently short for adopting the agile methodology. The project specification is deliberately designed to be ambiguous with the additional burden of frequent changes in the requirements initiated by the advisors acting as the customers. Additionally, the system domains chosen throughout the four semesters were noncritical to life. In terms of the team dimension, the size of the project group is capped at a maximum of six members. Moreover, students possessed the required technical skill set in IS development from prerequisite courses. In terms of the customer dimension, the use of teaching staff acting as customers of the system to be developed allows the customers to be “onsite” and participate in the system development as an integral part of the project team. In summary, our course design is well-suited for agile methodology.

In the next section, we will guide the readers across the four semesters of course development based on our hybrid agile methodology and the preceding three principles. The changes made in each semester of course revision were applied gradually in accordance with the general guidance of the theories of instructional design.

IV. COURSE DESIGN

Systems Approach Toward Course Design

The objectivist traditional theories of instructional design, such as the systems approach model for designing instruction [Dick and Carey, 1991], posit that learners may be taught to apply previously unfamiliar knowledge through an instructional system that is carefully designed to achieve the intended learning outcomes [Jonassen, 1991]. These theories are thus generally well-suited, given two learning conditions where: (1) the students lack prior knowledge of the area of learning; and (2) the learning outcomes are focused on acquiring and applying new concepts and principles [Moallem, 2001]. The systems approach model is thus a viable theoretical model for guiding our course development since (1) the students taking the ISD course had prior knowledge of mainstream non-agile methodologies only, but had not applied them in a practical setting; and (2) the primary learning outcome of the course was the application of our hybrid agile methodology to complete the assigned project specifications. In particular, we followed the Gagne et al. [1992] prescription of the system approach model [Dick and Carey, 1991], which consisted of ten stages.

However, the system approach model was intended for supporting instructional design at a fine level, and some of the stages were not appropriate in our context. For instance, the imparting of the hybrid agile methodology took the form of a large-scale practical project, and the actual understanding of the methodology was not examined in a question-based test. Thus, we omitted the development of criterion referenced test items. The choice of instructional media was also less relevant, since we adopted a mainly lecture-based pedagogy supplemented by team-based consultations. Moreover, this paper documents the successive refinement of the course over four semesters in which the fourth semester's course was deemed to be the most ideal. Although a final summative evaluation was conducted at the end of the fourth semester, the discussion of the results was deferred to a later section after the presentation of the course design. The remaining seven stages of the system approach model were faithfully followed.

In Stage 1, we identified the instructional goals to be achieved by the students. At the beginning of the first semester, the overarching instructional goal was for the students to apply our hybrid agile methodology in developing their project systems iteratively. This goal formed the cornerstone of our first course design principle. In the subsequent three semesters, we placed a gradually increasing emphasis on getting students to react to changes in business requirements [Schuh, 2005]. The final instructional goal was to achieve a balance of minimum system documentation [Schuh, 2005] with sufficient initial analysis and design work to facilitate subsequent agile and flexible system development. This involved the preparation of a system proposal based on the user stories [Allen et al., 2003] gathered at the beginning of the semester. In other words, the system proposal was the initial draft of the analysis and design report, to be submitted by the students at the end of the semester, containing the output from the initial planning phase. With the instructional goals defined, we then determined the skills required by the students to achieve them during Stage 2. This took into consideration the entry characteristics of our students, which we identified in Stage 3.

In Stage 4, we delineated the performance objectives, which increased in scope and complexity over the four semesters in accordance with the increasing instructional goals. The exact lesson plans and materials were prepared in Stage 5 taking into consideration the skill requirements identified in Stage 2, the constraints noted in Stage 3 and the performance objectives formulated in Stage 4. After the instructional strategy developed in Stage 5

Table 1: System Approach to Course Design

Stages	Semester 1	Semester 2	Semester 3	Semester 4
1. Identify instructional goals.	Iterative IS development using agile methodology.			
	Nil	React to changes in requirements.	React to multiple and rapid changes to requirements.	React to multiple and rapid changes to requirements at the onset.
		Nil	Formalize initial system analysis and design.	
2. Conduct instructional analysis.	Adoption of hybrid agile methodology (see Figure 1).			
	Nil	Adoption of requirement shocks.	Adoption of multiple requirement shocks.	Adoption of early multiple requirement shocks.
		Nil	Adoption of system proposal.	
3. Identify entry behavior, characteristics.	Prior knowledge of mainstream non-agile IS development methodologies but without practical experience			
4. Write performance objective.	Improve overall IS development performance.			
	Nil	Enhance final system release to meet change in requirements.	Enhance final system release to meet multiple changes in requirements.	Enhance first and final system releases to meet multiple changes in requirements.
5. Develop instructional strategy.	<ul style="list-style-type: none">Teach students hybrid agile methodology through lectures.Intensive hands-on guidance through team-based consultations.			
	Nil	<ul style="list-style-type: none">Communication of change(s) in requirements to students through team-based consultations.Guide students to incorporate changes to existing system through team-based consultations.		
6. Design and conduct formative evaluation.	<ul style="list-style-type: none">Use post-semester feedback exercise to solicit students' comments on the course.Internal discussions among members of the teaching teams to revise course for subsequent semesters.			
7. Revise instruction (in addition to the adoption of the hybrid agile methodology).	Start of semester 1: <ul style="list-style-type: none">3 system releases.Additional lectures on project management.Peer review after the submission of each system release.Project specifications provide clear business context, scope and expected deliverables.Compulsory and regular meeting up with team advisors.			
	Nil	Start of semester 2: <ul style="list-style-type: none">Project specifications provided even more information on business domain.External evaluators to provide comments.	Start of semester 3: <ul style="list-style-type: none">Intensive workshop on Java EE held prior to start of semester.Additional second requirements shock.Submission of system proposal.	Start of semester 4: <ul style="list-style-type: none">Stronger collaboration with industry.Requirement shocks brought forward.

had been delivered, we performed a formative evaluation of the course in Stage 6, which formed the input for the revision of the course in the subsequent semester, i.e., Stage 7. The changes made to the course did not affect the hybrid agile methodology itself. Instead, they served as additional enhancement techniques to supplement the entire course to achieve our pedagogical objectives. This is in accordance with our second course design principle.

The details of the entire course design following the system approach across the four semesters are summarized in Table 1. Each stage is represented by a major row, which may consist of one or more minor rows. Reading from left to right and top to bottom within each major row but across the minor rows will reveal the changes made at the beginning of each semester. For instance, in Stage 2, i.e., conduct instructional analysis, we adopted the hybrid agile methodology across all four semesters. In the second semester, we further incorporated requirement shocks into the course design. We then switched to the adoption of multiple requirement shocks in the third semester before finally opting to introduce the multiple requirement shocks earlier in the fourth semester. Moreover, for the third and fourth semesters, we also incorporated the system proposal as an additional deliverable.

Instruction Events of Our Course Design

Instruction events represent the transitions from one learning activity in a lesson to another activity, and are designed to facilitate the learner's learning process [Gagne et al., 1992]. There are altogether nine such events ranging from the initial activities to gain the learner's attention to the teaching materials and communicate the lesson objectively, to the subsequent activities of assessing the learner's performance and helping the learner to apply the newly-gained knowledge in solving other relevant problems. However, these events are more applicable at the micro level of the specific instances of a lesson. In other words, most of the events are not applicable at our course level. In our present context, we focused on four specific instruction events that demonstrate the gradual enhancements we made to our course design (see Table 2).

Table 2: Instruction Events Relevant to Our Course Design

Events	Semester 1	Semester 2	Semester 3	Semester 4
1. Providing learning guidance.	Team-based consultations and discussions.			
	In-semester lectures on methodology, project management and Java EE.		<ul style="list-style-type: none">• Pre-semester intensive workshop on Java EE• In-semester lectures on methodology and project management.	
	Nil	Team-based guidance on change(s) in the business requirements.		
2. Eliciting the performance.	<ul style="list-style-type: none">• 1st system release• Analysis and design report.• Final system release.			
	Nil		System proposal.	
3. Providing feedback about performance correctness.	Internal evaluation feedback.			
	Nil	External evaluation feedback.		
		Feedback on change in business requirements.	Multiple feedbacks on change in business requirements.	
4. Assessing the performance.	<ul style="list-style-type: none">• 1st system release score.• Analysis and design report score.• Final system release score.			
	Nil		System proposal score.	

The provision of learning guidance involved various activities that were aligned with the instructional strategies outlined in Table 1 and were intended to help the students achieve the various instructional objectives. Performance elicitation consisted of various activities that were used to ascertain if the students were able to apply the hybrid agile methodology in their project work. Feedback on performance was provided by internal and external evaluators to help the teams make improvements to their systems. In addition, depending on the system features delivered by the team in the intermediate system releases, additional guidance was selectively given to the teams to help them in reacting to changes in the business requirements. Performance, with respect to how well the teams had applied the hybrid agile methodology in developing the project system and responding to the changes in the business requirements are reflected in the scores awarded to the various system releases.

As illustrated in Table 2, the four instruction events did not remain static, but instead were gradually enhanced over the four semesters, based largely on the feedback given by the students during the formative evaluation of the course design. For instance, a pre-semester intensive workshop on Java EE was introduced in response to feedback from students indicating that they wanted to commence productive coding right at the beginning of the semester: "... maybe can start lectures during the holidays... ." Acquiring the required technical skills could help students derive greater benefits from the agile methodology [Schneider and Johnston, 2003].

The inclusion of the system proposal as part of the performance evaluation was also made in response to students' feedback: "Assessment could be rethought. How about one or two intermediate assessments of 10-15% each...."

Finally, the involvement of external evaluators together with the team advisors in the performance feedback process was made in response to students' requests:

"It's good that we have an advisor who checks on our progress every now and then."

"[need] Help... by suggesting that we consider industry rules and situations... . Proactive in offering suggestions for improvement and alternatives."

The remaining changes, e.g., the provision of team-based guidance on change(s) in the business requirements, were made in accordance with our changing instructional objectives and strategies over the four semesters (see Table 1).

Final Agile Course Design

Through the application of the system approach model of instructional design [Dick and Carey, 1991], the agile IS development course began with the most primitive design in the first introductory semester as reflected in the top panel of Figure 2. Over the subsequent three semesters, the refinement of the course design was based on the outcomes of the formative evaluation as detailed in Table 1. These successive refinements culminated in the final course design shown in the lower panel of Figure 3 and is elaborated upon in the remainder of this section.

General Pedagogy

During the final stage of the course design, instructional guidance to the students commenced two weeks before the start of the semester, with a pre-semester intensive workshop on Java EE. This was followed by a series of lectures over a three-week period to impart to students the necessary methodology and project management knowledge. Concurrently with the lectures, team-based consultations and discussions began in Week 1, running through Week 12. These consultation sessions proved especially useful during the planning phase of the hybrid agile methodology because we placed great emphasis on the business context and business user involvement. The close interaction between the team and the advisor who played the roles of an end-user and a mentor gave rise to two important benefits. An advisor does not only work collaboratively with the team to derive the required usage scenarios but also selects those that are relevant and develops the features list. In addition, the advisor provides guidance to the team in prioritizing the development schedule of each scenario and its corresponding features list based on assumed business requirements.

Project Development Iterations

The project development is segregated into three distinct iterations, depicted in the lower panel of Figure 3 as three long rectangular blocks, each leading to a fully functional system release. In addition, students are required to submit a peer review on the completion of each system release. For each iteration, the students are expected to go through all four phases of the hybrid agile methodology (see Figure 1), although the exact number of cycles is left to the discretion of each individual team. The first iteration spans six-and-a-half weeks during which the students are required to submit a system proposal in Week 3. The first system release was evaluated during the recess week. Thereafter the students performed the post evaluation review phase. The second iteration spans a shorter three-week period. Students were expected to submit the second system release for informal evaluation in the middle of Week 10 before spending the remaining week on post evaluation review. The third and last iteration began in Week 11. At the end of that week, the students were required to submit their team's analysis and design report. The final system release evaluation was performed in Week 13.

This iterative IS development approach using the hybrid agile methodology is in accordance with the agile software development methodology [Bittner and Spence, 2006; Rajlich, 2006; Jalote et al., 2004] that emphasizes having several short iteration cycles to enable fast verification and corrections, reducing each cycle to a duration of one to six weeks, removing all unnecessary activities (e.g., redundant documentation), and focusing on an incremental rather than radical approach toward software development [Martin, 2003]. The main objective of having an incremental approach is to equip the students with the practical experience of managing a stable, integrated, and tested system release that is more aligned with the requirements. Specifically, the advisors (i.e., tutors and/or the lecturer) are able to offer timely feedback only if students are able to showcase the system regularly with incremental delivery of features.

First Semester

Second Semester

Figure 2: Course Design for the First and Second Semesters

During the first two iterations, we also introduced changes to the business requirements of the protagonist organization in the project specifications. This was achieved through two requirement shocks: (1) the first one was administered in Week 4, i.e., in the middle of the first iteration (first arrow in the lower panel of Figure 3); and (2) the second one was administered at the beginning of the second iteration in Week 7 (second arrow in the lower panel of Figure 3). The requirement shocks can be loosely classified as feature modifications, feature enhancements, or new features requirements. These requirement shocks were carefully crafted so that the teams would need to include additional usage scenarios and revise their system design. New features requirement is typically less disruptive with

Figure 3: Course Design for the Third and Fourth Semesters

regard to prior work done. Feature modifications or enhancements, however, may require substantive effort to reanalyze prior work and determine how the new requirements can be fitted into an existing code. If the affected feature is tightly coupled with other features, any additional work involved can be extremely erroneous. Regardless of such issues, the teams are generally expected to reprioritize part of the features set that is to be implemented in the next feature sub-task iteration as well as to re-examine the overall project schedule. The team advisors were briefed to provide close guidance and supervision at this critical stage.

Additional Pedagogical Techniques

In addition to the main agile course design outlined above, a total of six pedagogical techniques were also incorporated to complement part of our second course design principle (see Table 3). These enhancements were made in response to students' feedback gathered during the formative evaluation conducted at the end of each semester. For instance, instead of providing students with a numerical score for their intermediate system releases, we mandated that the respective team advisors and evaluators were to provide the students with detailed and meaningful as well as actionable comments. Lectures on conflict resolution and project management were also delivered to the students. The teams were also asked to constantly update their respective advisors on the progress of the system development and team conflicts, if any. At the end of each system release, students were required to submit peer reviews of their team members so that the team advisors could detect any problems as early as possible and intervene accordingly.

Table 3: Additional Pedagogical Techniques to Complement Agile Course Design

No.	Issues	Comments Made By Students Taking The Course During Post-Semester Feedback Exercise	Pedagogical Technique
1	Feedback	"It would really help if the lecturer can provide a guideline to the things that is to be completed rather than let the student do as much as they want. It becomes really hard to gauge how much we have to do."	Advisor enforced two internal system release evaluations. The first evaluation involved external evaluators.
2	Team dynamics	"it is very difficult to operate as a group"	Lectures offered on managing a project team. Additional on-going guidance offered by the advisors. Peer reviews are required from the students at various intermediate points in the semester.
3	Clarity of requirements	"Since the Project had no exact specifications, it was a little bit tedious on our part to give the Project a specification. Moreover, we were slightly confused as to the direction which we should adopt for the Project... "	Project specification was revised with clearer indication of the business context and scope of the project. Mandate more and regular meet-ups between the advisors and the individual teams.
4	Unclear timeline	"provide the timeline, so students can know how are their progress"	Project specification was revised with clearer indication of expected deliveries.
5	Understanding of the business domain	"Projects specifications must be made clearer. My group has a lot of problems understanding the mechanism of proxy bidding."	Project specification was further revised. The number of pages increased from 19 in the first semester to 35 in the second semester due to explicit inclusion of more diagrams and explanations of the business domain.
6	Project realism	"Increase the difficulty level of the module by involving a real industry project as the project specification"	Stronger collaboration with industry players was enforced with them playing a more active role in defining the project specification.

The remaining techniques involved tweaking the project specifications to increase the clarity and realism of the project requirements. Two novel changes were proposed: (1) that the selection of the business/problem domain be less dependent on the preferences of the lecturer, but rather that they be more of a composite aggregation of students' preferences, industry player choices, IT market trends, course objectives, and scope of evaluation; and (2) selected industry players be invited to first read and then comment on the project specifications to add a touch of realism.

Final Summative Evaluation

According to the instructional design theories, our course design process is not completed until the overall effectiveness of the course design is assessed by a final summative evaluation [Gagne et al., 1992; Gustafson and Branch, 1997]. In particular, the summative evaluation should take place only after the course design has been subjected to a series of detailed revisions based on the outcomes of the formative evaluations [Gagne et al., 1992]. Consistent with this line of reasoning, we performed a summative evaluation of our agile course design at the end of the fourth semester after we had revised the design based on the students' comments, as well as feedback from the teaching team. Building on the performance objectives defined in the design process (see Table 1) and the actual performance assessment of the students' work (see Table 2), our summative evaluation was based on the numerical scores obtained by each team for their first and final system releases. The second system release was assessed informally only and thus no formal score was awarded. Our fundamental assumption is that if each successive refinement of the agile course design was indeed more effective in helping students to improve their system development, then the average scores of the students' system releases should increase gradually.

To ensure that the comparisons of the students' performance were meaningful, it was imperative to keep the assessment criteria consistent across the four semesters. The only exception was a minor tweaking made to the assessment weighting for the first system release due to the inclusion of the system proposal from the third semester onward. Specifically, 10 percent of the weighting was shifted from the assessment of feature breadth to the students' ability to complete the features as scheduled in the system proposal. The detailed assessment criteria are shown in Table 4. In addition, the learning conditions were generally invariant. For instances, the same instructor taught throughout the four semesters using the same development platform. There were, however, two slight variations. First, the tutors employed as team advisors differed slightly across the four semesters. Specifically, we replaced one tutor in the third semester who left the University at the end of the second semester. Second, the domain of the system to be developed was different across the four semesters. This is imperative to prevent plagiarism and more importantly to ensure that the students worked on a system that was relevant to the needs of the industry at that time (see the sixth pedagogical technique listed in Table 3). However, regardless of the domains, the scope of the project system with respect to the amount of business requirements to be addressed was kept consistent across the four semesters. This ensured that the students' overall workload across the four semesters was similar.

Table 4: Detailed Assessment Criteria

Table 4: Detailed Assessment Criteria				
Assessment Component	Assessment Criteria			
	Semester 1	Semester 2	Semester 3	Semester 4
First System Release	Nil		Ability to complete features as scheduled in system proposal—10%	
	Feature breadth—40%		Feature breadth—30%	
	Feature complexity—40%		Feature complexity—40%	
	Overall quality of system—20%			
Final System Release	Level of system integration—10%			
	Feature breadth—30%			
	Feature complexity—30%			
	Aesthetic and functional aspects of user interface—10%			
	Overall quality of system—20%			

The descriptive statistics of the teams and their performance over the four semesters are shown in Table 5. There were altogether 82 teams ($\bar{x} = 20.5$, $\delta = 2.1$) with the mean first system release score being 65.8 ($\delta = 10.6$) and the mean final system release score being 70.8 ($\delta = 11.4$). Based on the university's grading policy, the mean first and final system release scores would correspond to the letter grade of B- and B, respectively. A paired sample t test indicates that the teams' final system release score is significantly higher than the first system release score ($\Delta \bar{x} = 5.0$, $\delta = 10.8$, $t = 4.2$, $p < .001$). Thus the agile course generally leads to a better final system release score.

We performed two separate univariate analyses of variance (ANOVA) on the first and final system release scores respectively. The results indicate that both: 1) the first system release score, $F(3, 78) = 6.7$, $p < .001$; and 2) the final system release score, $F(3, 78) = 3.1$, $p < .033$, are significantly different across semesters. Inspecting the line plots in Figure 4 suggests that the students' performance improved (in general) from the first semester to the fourth semester. Further post-hoc pair-wise comparisons using the Bonferroni adjustment technique (see Table 6) were used to offer more fine-grained examination of performance between any two semesters.

Table 5: Descriptive Statistics of Teams and Their Performance

Semester	Number of Teams	Mean First System Release Score (Standard Deviation)	Mean Final System Release Score (Standard Deviation)
1	20	57.5 ($\delta = 6.4$)	66.0 ($\delta = 6.9$)
2	23	69.6 ($\delta = 5.4$)	69.7 ($\delta = 10.8$)
3	18	67.6 ($\delta = 17.6$)	71.3 ($\delta = 14.4$)
4	21	68.0 ($\delta = 5.0$)	76.2 ($\delta = 10.8$)

Figure 1: Line Plots of the Main Effects of the Semesters on the First and Final System Release Scores**Table 6: Post-hoc Pair-wise Comparison of Students' Mean Scores Between Semesters**

System Release	Pair-wise Comparison of Scores					
	1 vs. 2	1 vs. 3	1 vs. 4	2 vs. 3	2 vs. 4	3 vs. 4
First	$\Delta \bar{X} = 12.0^{**}$	$\Delta \bar{X} = 10.0^{*}$	$\Delta \bar{X} = 10.4^{**}$	$\Delta \bar{X} = -2.0$	$\Delta \bar{X} = -1.6$	$\Delta \bar{X} = 0.4$
Final	$\Delta \bar{X} = 3.7$	$\Delta \bar{X} = 5.3$	$\Delta \bar{X} = 10.2^{*}$	$\Delta \bar{X} = 1.6$	$\Delta \bar{X} = 6.4$	$\Delta \bar{X} = 4.8$

* $p < .05$, ** $p < .01$

With reference to Table 6, for the first system release score, teams in the second, third and fourth semesters performed significantly better than teams in the first semester. For the final system release score, only teams in the fourth semester performed significantly better than teams in the first semester. The effectiveness of the fourth course design was comparable to the second and third course designs since the score differences did not reach statistical significance.

Although the first and final system release scores did not increase consistently over the four semesters, the results indicate clearly that the fourth semester of the agile course design led to significant improvement in the scores, compared to the initial design in the first semester. In addition to the statistical findings, a review of the comments provided by the students during the post-semester feedback exercise across the four semesters indicated that students were generally receptive toward the adoption of the agile course (see Table 7). Compared to the earlier semesters in which the legacy course was used, students trained in the agile course did not make any specific complaints about being unable to complete their project systems to their desired standard due to planning failure or tight deadlines. Based on these findings, we would recommend the use of the fourth design, i.e., the final design, to incorporate the concepts of agility, flexibility and adaptability in IS development.

Table 7: General Comments from Students Taking the Module

<i>"Extremely practical module, learning how it is done in a corporate environment"</i>
<i>"This module helps to train our skills in learning a technology fast enough to complete a project."</i>
<i>"[The course] really give us the general feel of the IT project that we might be taking up in future when we work. Trains us mentally. Also trains us on teamwork."</i>
<i>"The module is very challenging in terms of the time management and skills required."</i>
<i>"This is a very tough but interesting module."</i>
<i>"Gives students a good experience of programming a major project"</i>
<i>"Allows students to have hands-on experience in situations related to a real-world working environment. Allows students to gain valuable experience from this module alone."</i>
<i>"Lets us experience what it is like to actually develop a commercial enterprise system."</i>

V. DISCUSSIONS ON THE AGILE COURSE DESIGN PROCESS

General Discussion

The discussions so far have focused on explaining and rationalizing the gradual development of a course for agile IS development. Specifically, we designed the course based on the supposition that our graduates should be trained to become responsive to a dynamically changing business environment characterized by frequent requirement changes. The efforts of the four semesters of incremental change were not only met with positive responses from the students but the overall course was also validated to be an effective one.

Two key lessons were learned from teaching the course. First, it is necessary to correct students' restrictive mindsets toward agile software development with the aid of a hybrid agile methodology (Figure 1) that harmonizes various best practices. It appears that the outlook of the situational software development approach, which emphasizes the adaptation of a suitable methodology to suit the needs of a particular development situation, looks promising [Fitzgerald et al., 2003]. Briefly, to impart the principles of agility, flexibility, and adaptability within a classroom situation, it is necessary to draw on the best practices of different agile methodologies to balance the coding and project-management needs of budding student developers (see the insert in Figure 1).

Second, the development of a coherent course design is not a mere accidental process, but rather one that is highly rigorous and one which must be grounded on well-established instructional design theories [Gagne et al., 1992]. By anchoring our agile course design process on the system approach model [Dick and Carey, 1991], we benefited tremendously from its meticulous stages of instructional design. For instance, we gradually enhanced our instructional objectives and aligned the various performance assessment criteria with the evolving objectives. Moreover, the formative evaluation stage provided useful information that was used to refine our course design longitudinally, which culminated in a final summative evaluation to assess the overall effectiveness of our course design.

Finer Points of the Agile Course Design

In the subsequent paragraphs, we summarize and present a coherent view of some of the finer points regarding how our course has been structured to seek better alignment of classroom teaching with industry practices for our ISD course.

Preparation Workshop

Pre-semester workshops were conducted for two reasons: (1) to impart to students the adequate knowledge and skills in programming, enterprise-level system architecture, and software development methodology to compensate for any inadequacy in the pre-requisite course, and (2) to allow students to start work on the project immediately after a semester commences. Preferably, preparatory classes or workshops should be conducted at least two weeks prior to the commencement of a semester.

Choice of Project Team Members

Students should be allowed to choose their own team members, as this would impose a certain amount of responsibility on the students so that they feel accountable for their choice of whom to work with. The preparation workshop also provides an early opportunity for potential team members to socialize and understand the individual strengths, weaknesses, and working styles through hands-on exercises.

Agile Approach to Requirements Analysis

Students begin the planning process in Week 1 by formulating usage scenarios. Owing to time constraints, we encourage educators to provide the students with fairly detailed descriptions of user needs in the project requirements. Advisors acting as surrogate users are needed to help teams visualize the system as a whole and prioritize the release schedule of features. Teams also ought to be encouraged to identify features of lower priority that can be deferred or even excluded from the final system. This offers a valuable learning experience for students to notably influence and negotiate course deliverables. The output from the requirement analysis is the list of feature requirements, the dependencies among the features, the complexity of each feature categorized from low to high, the estimated amount of effort needed, and the priorities associated with the features. Students should also be reminded to consider both the complexity and interdependencies of features when refining the release schedules. When requirement shocks are introduced, the list of features should be updated so that more accurate estimations and new priority orders can be derived. The output from the analysis serves as the monitoring device for the teams to plan and track their activities throughout the project.

Agile Approach to System Design

System design is to be evolutionary within iterative executions of the whole development lifecycle that encompasses analysis, design, coding, and testing. Specifically, we encourage educators to adopt test-driven software development, which is an integral part of agile software development methodology. This approach requires both development team members and users, i.e., team advisors, to collaboratively compose test cases prior to code implementation. Thus, teams are compelled to anticipate ways in which the system is to be tested and utilized at the early stages of development. Furthermore, test plans can grow incrementally with each iterative implementation and changes in requirements. Students can use the test cases in two ways: (1) during coding to expedite the development in quick, small increments whereby a feature is deemed to be completed when an entire set of test cases is met, and (2) at the completion of each release cycle to demonstrate an acceptable level of features for that release.

Requirement Shocks

Not all the teams were introduced to the same types of requirement shocks. Before the introduction of each requirement shock, the teaching team would perform an internal ranking of all the teams based on the proportion of proposed features that each team had completed at that time. This ranking did not take into consideration the quality of the features, since it was premature to prejudge the teams' efforts. Teams that had completed less than half their proposed features were considered the weaker teams. Specifically, the weaker teams which were experiencing difficulty coping with the initial set of requirements were spared from making feature modifications and enhancements. Instead, they were asked only to add some simpler features that could be accomplished without having to make substantial changes to the existing design. In order not to demoralize the weaker teams, all requirement shocks were hence privately administered by the respective advisors.

The stronger teams who had completed at least half of their proposed features were given comparatively more complex new features to implement and asked to make simple modifications to existing features. Only the top 10 percent of the teams, typically three or four teams, were asked to make complex modifications to existing features. For the better teams, the approach was to frame the tougher requirement shock as a "challenge" and a "privilege" to them to minimize resentment. In general, the students did not lodge any complaints regarding such difference in treatment, as the pedagogical objective of the requirement shocks was tactfully communicated to the students, and they were fully aware of their own progress and capability.

Regardless of the degree of requirement shocks administered, the assessment of all the teams was based on the same set of criteria as listed in Table 4. This consistent assessment policy was not biased against the weaker teams that were not given the tougher requirement shocks. This was because the weaker teams had an equal opportunity to obtain a good grade by implementing features with sufficient complexity and quality to address the original requirements in the project specification. Moreover, the weaker teams were also able to score well on other criteria, such as user interface and system integration. Comparatively, the stronger teams that were given the tougher requirement shocks could have failed to do well if they were unable to integrate the changes into their base system. It is also important to note that the same requirement shocks and assessment policies were applied throughout the four semesters and thus the results of the final summative evaluation are reasonably conclusive in our opinion.

Future research along the perspective of requirements engineering (RE) may provide additional insights on the use of requirement shocks as an educational intervention. RE refers to the process of identifying and documenting the purposes of the information system in meeting the needs of the key stakeholders [Nuseibeh and Easterbrook, 2000]. Some possible areas include instilling in students the importance of making cost-benefit analyses in response to the requirement shocks as well as the possible use of automated tools to manage the changes effectively [Nuseibeh

and Easterbrook, 2000]. As noted by Hofman and Lehner [2001], requirement changes are associated with benefits and disruptions. Thus, weaker teams can still be given requirement shocks, provided they cause minimal disruption.

Moreover, requirement shocks may be implemented in different ways. For instance, techniques such as hands-on demonstration of prototypes to solicit feedback from users, as well as group elicitation possibly involving industry practitioners [Nuseibeh and Easterbrook, 2000], may inject greater realism and thus trigger more creative and versatile responses from students. Finally, conflict and divergence in the requirement goals of different stakeholders is a common problem in RE that needs to be handled carefully [Van Lamsweerde et al., 1998]. Thus, it is imperative that students should be taught how to handle conflict appropriately. Since our course emphasizes closer interaction between the students and the team advisor, the latter is in a good position to provide the required guidance on resolving conflicts.

System Releases and Feedback

The students should be well-informed and constantly reminded that the bulk of the assessment is based on the ability to deliver a working, integrated IT solution to the problem defined in the project specifications. While we did attempt to impose continuous assessment throughout the semester, the evaluation of the system at the final system release still constituted 50 percent of the entire grade. This coincides with an industry practice where the client pays the software vendor only when the system is delivered. At the end of each system release and at the end of the project, teams should engage in debriefing sessions (i.e., post-mortems) where the evaluators identify not only the software flaws but also opportunities for process and software improvements.

The final evaluation of the system can be conducted in a way that simulates real-world business presentations where teams demonstrate their systems to the clients. In our case, three examiners, namely the lecturer, one external tutor, and the team advisor, were responsible for each team's presentation. The presence of the advisor was (1) to offer moral support to the team as both parties would have forged a strong rapport during the semester and (2) to identify whether any misinterpretation of the requirements (if any) came from the team or the advisor. In the case of the latter, the team should not be penalized. Each team is given ninety minutes to present the system.

VI. CONCLUSION

The primary objective of this paper is to share our experience of conducting a course on IS development, an area of research relatively lacking in extant IS publications [Vessey et al., 2002]. Indeed, this viewpoint is best shared by Bajaj and his colleagues [2005, pp. 478] who argued *"In this rapidly changing core area, one should expect modern textbooks and curricula to incorporate sound research on methodologies, techniques, and leading practice. Yet, there appears to be a relatively low level of SA&D research during the past decade, relative to the needs of the teaching and practice communities."*

With this in mind, we hope this article, which documented two years of our ongoing effort in improving the quality of IS development teaching, has contributed in a way toward enhancing the undergraduate IS program in universities worldwide. In particular, we hope the knowledge gained from our course revamp and refinement will serve as suggestions to peers who are eager to seek improvement in their courses that are related to IS development as well as to IS program committee members who are constantly seeking better alignment of their IS program to the industry's best practices.

REFERENCES

Editor's Note: The following reference list contains hyperlinks to World Wide Web pages. Readers who have the ability to access the Web directly from their word processor or are reading the paper on the Web, can gain direct access to these linked references. Readers are warned, however, that:

1. These links existed as of the date of publication but are not guaranteed to be working thereafter.
2. The contents of Web pages may change over time. Where version information is provided in the References, different versions may not contain the information or the conclusions referenced.
3. The author(s) of the Web pages, not AIS, is (are) responsible for the accuracy of their content.
4. The author(s) of this article, not AIS, is (are) responsible for the accuracy of the URL and version information.

- Allen, E., R. Cartwright, and C. Reis (2003) "Production Programming in the Classroom", *SIGCSE Bulletin* 35(1), pp. 89–93.
- Ambler, S.W. (2008) "Agile Adoption Rate Survey: February 2008", Ambyssoft, <http://www.ambyssoft.com/surveys/agileFebruary2008.html> (current Apr. 22, 2009).
- Avison, D., and M. Cole (2006) "Reflections on Teaching Information Systems Analysis and Design: From Then to Now!", *Journal of Information Systems Education* 17(3), pp. 253–256.
- Avison, D.E., and G. Fitzgerald (2003) "Where Now for Development Methodologies?" *Communications of the ACM* 46(1), pp. 79–82.
- Bajaj, A., et al. (2005) "System Analysis and Design: Should We Be Researching What We Teach?" *Communications of the Association for Information Systems* 15, pp. 478–493.
- Beck, K., et al. (2001) Manifesto for Agile Software Development, <http://www.agilemanifesto.org> (current Apr. 22, 2009).
- Bittner, K., and I. Spence (2006) *Managing Iterative Software Development Projects*, Boston, MA: Addison Wesley.
- Davidson, M. (2008) "Survey: Agile Interest High, But Waterfall Still Used by Many", SearchSoftwareQuality.com, http://searchsoftwarequality.techtarget.com/news/article/0,289142,sid92_gci1318992,00.html (current Apr. 22, 2009).
- Dick, W., and L. Carey (1991) *The Systematic Design of Instruction*, 3^d edition, New York: Harper Collins.
- Farhoomand, A. (2004) "Writing Teaching Cases: A Quick Reference Guide", *Communications of the Association for Information Systems* 13, pp. 103–107.
- Fitzgerald, B., N.L. Russo, and T. O'Kane (2003) "Software Development Method Tailoring at Motorola", *Communications of the ACM* 46(4), pp. 65–70.
- Gagne, R.M., L. Briggs, and W. Wager (1992) *Principles of Instructional Design*, 4th edition, Fort Worth, TX: HBJ College Publishers.
- Gagne, R.M., and M.P. Driscoll (1988) *Essentials of Learning for Instruction*, 2nd edition, Englewood Cliffs, NJ: Prentice-Hall.
- Golden, D., and V. Matos (2006) "Introducing the Unified Modeling Language into the Information Systems Curriculum", *Journal of Information Systems Education*, 17(1), pp. 83–92.
- Gustafson, K.L., and R.M. Branch (1997) "Revisioning Models of Instructional Development", *Educational Technology Research and Development* 45(3), pp. 73–89.
- Hackney, R., T. McMaster, and A. Harris (2003) "Using Cases as a Teaching Tool in IS Education", *Journal of Information Systems Education* 14(3), pp. 229–234.
- Hedin, G., L. Bendix, and B. Magnusson (2003) "Introduction Software Engineering by Means of Extreme Programming", in *Proceedings of the 25th International Conference on Software Engineering*, pp. 586–593.
- Highsmith, J. (2002) "Agile Software Development Ecosystems" in Cockburn, A. and J. Highsmith (eds.) *The Agile Software Development Series*, Boston, MA: Addison-Wesley.
- Hofman, H., and F. Lehner (2001) "Requirements Engineering as a Success Factor in Software Projects", *IEEE Software* 18(4), pp. 5–66.
- Jalote, P., et al. (2004) "Timeboxing: A Process Model for Iterative Software Development", *Journal of Systems and Software* 70(1–2), pp. 117–127.

- Jones, M.C., and T. McMaster (2004) "Addressing Commercial Realism and Academic Issues in Group-Based IS Undergraduate Project Work", *Journal of Information Systems Education* 15(4), pp. 375–381.
- Jonnassen, D.H. (1991) "Objectivist vs. Constructivist: Do We Need a New Philosophical Paradigm?" *Educational Technology: Review and Development* 39(3), pp. 5–14.
- Kim, Y., J. Hsu, and M. Stern (2006) "An Update on the IS/IT Skills Gap", *Journal of Information Systems Education* 17(4), pp. 395–402.
- Kivi, J., et al. (2000) "Extreme Programming: A University Team Design Experience" in *Proceedings of the 2000 Canadian Conference on Electrical and Computer Engineering* 2, pp. 816–820.
- Laplane, P.A. (2006) "An Agile, Graduate, Software Studio Course", *IEEE Transactions on Education* 49(4), pp. 417–419.
- Lindvall, M., et al. (2002) "Empirical Findings in Agile Methods" in *Proceedings of Extreme Programming and Agile Methods—XP/Agile Universe 2002*, Chicago, IL, pp. 197–207.
- Lyytinen, K. (2005) "Agile Modeling, Agile Software Development, and Extreme Programming", *Journal of Database Management* 16(4), pp. 88–100.
- Mahapatra, R., S.P. Nerur, and C.W. Slinkman (2005) "Special Theme of Research in Information System Analysis and Design—III Teaching Systems Analysis and Design—A Case for the Object Oriented Approach", *Communications of the Association for Information Systems* 16, pp. 848–859.
- Martin, R.C. (2003) *Agile Software Development: Principles, Patterns, and Practices*, Englewood Cliffs, NJ: Prentice Hall.
- Mathiassen, L., and J. Pries-Heje (2006) "Business Agility and Diffusion of Information Technology", *European Journal of Information Systems* 15(2), pp. 116–119.
- McAvoy, J., and D. Sammon (2005) "Agile Methodology Adoption Decisions: An Innovative Approach to Teaching and Learning", *Journal of Information Systems Education* 16(4), pp. 409–419.
- McGann, S.T., et al. (2007) "Meeting the Challenge of IS Curriculum Modernization: A Guide to Overhaul, Integration, and Continuous Improvement", *Journal of Information Systems Education* 18(1), pp. 49–62.
- Meso, P., and J. Liegle (2005) "An Exploratory Assessment of the Pedagogical Effectiveness of a System Development Environment", *Journal of Information Systems Education* 16(2), pp. 157–166.
- Moallem, M. (2001) "Applying Constructivist and Objectivist Learning Theories in the Design of a Web-Based Course: Implications for Practice", *Educational Technology & Society* 4(3).
- Montalbano, E. (2007) "IBM Donates Rational Processes to Eclipse", *Computerworld*, www.computerworld.com.au/index.php/id;1414122493;fp;16;fpid;0 (current Apr. 22, 2009).
- Morien, R. (2005) "Agile Management and the Toyota Way for Software Project Management", in *Proceedings of the 3rd IEEE International Conference on Industrial Informatics*, pp. 516–522.
- Muller, M.M., and W.F. Tichy (2001) "Case Study: Extreme Programming in a University Environment" in *Proceedings of the 23rd International Conference on Software Engineering*, Toronto, Ontario, Canada, pp. 537–544.
- Necco, C.R., C.L. Gordon, and N.W. Tsai (1987) "System Analysis and Design: Current Practices", *MIS Quarterly*, 11(4), pp. 461–470.
- Nerur, S., and V. Balijepally (2007) "Theoretical Reflections on Agile Development Methodologies", *Communications of the ACM* 50(3), pp. 79–83.
- Nuseibeh, B.A., and S.M. Easterbrook (2000) "Requirements Engineering: A Roadmap", in Finkelstein, A.C.W. (ed.) *The Future of Software Engineering (Companion Volume to the Proceedings of the 22nd International Conference on Software Engineering, ICSE'00)*, IEEE Computer Society Press.
- Rajlich, V. (2006) "Changing the Paradigm of Software Engineering", *Communications of the ACM* 49(8), pp. 67–70.
- Reichlmayr, T. (2003) "The Agile Approach in an Undergraduate Software Engineering Course Project" in *Proceedings of the 33rd ASEE/IEEE Frontiers in Education Conference*, Boulder, Colorado, pp. 5–8.
- Savenye, W.C., Z. Olina, and M. Niemczyk (2001) "So You Are Going to Be an Online Writing Instructor: Issues in Designing, Developing, and Delivering an Online Course", *Computers and Composition* 18, pp. 371–385.
- Sawyer, S. (2004) "Software Development Teams", *Communications of the ACM*, 47(12), pp. 95–99.

- Schneider, J.G., and L. Johnston (2003) "eXtreme Programming at Universities—An Educational Perspective", in *Proceedings of the 25th International Conference on Software Engineering*, Portland, Oregon, pp. 594–599.
- Schuh, P. (2005) *Integrating Agile Development in the Real World*, Charles River Media.
- Topi, H., et al. (2007) "Revising the IS Model Curriculum: Rethinking the Approach and the Process", *Communications of the Association for Information Systems* 20, pp. 728–740.
- Tripp, S.D., and B. Bichelmeyer (1990) "Rapid Prototyping: An Alternative Instructional Design Strategy", *Educational Technology Research and Development* 38(1), pp. 31–44.
- Turk, D., R. France, and B. Rumpe (2002) "Limitations of Agile Software Processes", in *Proceedings of the 3rd International Conference on Extreme Programming and Flexible Processes in Software Engineering*, Alghero, Italy, pp. 43–46.
- Van Lamsweerde, A., R. Darimont, and E. Letier (1998) "Managing Conflicts in Goal-Driven Requirements Engineering", *IEEE Transactions on Software Engineerin*, 24(11), pp. 908–926.
- Van Vliet, P.J.A. and L.R. Pietron (2006) "Information Systems Development Education in the Real World—A Project Methodology and Assessment", *Journal of Information Systems Education* 17(3), pp. 285–293.
- Vessey, I., V. Ramesh, and R. Glass (2002) "Research in Information Systems: An Empirical Study of Diversity in the Discipline and its Journals", *Journal of Management Information Systems* 19(2), pp. 129–174.
- Wang, M. (2003) "E-Business Application Development with Java Technology and Oracle: The Fortune Invest Inc. Case", *Journal of Information Systems Education* 14(3), pp. 293–300.

ABOUT THE AUTHORS

Chuan-Hoo Tan, is an assistant professor of Information Systems at City University of Hong Kong. He holds a Ph.D. in Information Systems from the National University of Singapore. His research interests include the design and evaluation of consumer-based decision support interfaces, electronic commerce, and technology adoption. His work has been published in journals such as *Information Systems Research*, *IEEE Transactions on Engineering Management*, *Annals of Operations Research*, *Information & Management*, *Electronic Markets*, and *Communications of the ACM* as well as conferences such as International Conference on Information Systems.

Wee-Kek Tan, is an instructor and Ph.D. candidate in the information systems department of National University of Singapore. He holds a B.Comp. in Information Systems (First Class Honors) from the National University of Singapore. His research interests include social computing, the design of online decision aid, human-computer interaction, and information systems development and education. His work has been published in journals such as *Journal of Information Systems Education* as well as conferences such as ACM SIGMIS Computer Personnel Research Conference, IFIP Working Group 8.2 Working Conference, European Conference on Information Systems, and Americas Conference on Information Systems.

Hock-Hai Teo, is an associate professor of information systems and head of the information systems department of National University of Singapore. His research interests are in the areas of IT innovation adoption, assimilation and impacts, information privacy, electronic market institutions, and virtual communities. He has published broadly in journals such as the *ACM Transactions on Computer-Human Interactions*, *MIS Quarterly*, *Information Systems Research*, *Journal of Management Information Systems*, *IEEE Transactions on Engineering Management*, *International Journal of Human-Computer Studies*, and *Information and Management* and has presented at numerous international conferences, including the International Conference on Information Systems.

Copyright © 2010 by the Association for Information Systems. Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and full citation on the first page. Copyright for components of this work owned by others than the Association for Information Systems must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or fee. Request permission to publish from: AIS Administrative Office, P.O. Box 2712 Atlanta, GA, 30301-2712, Attn: Reprints; or via e-mail from ais@aisnet.org.

Communications of the Association for Information Systems

ISSN: 1529-3181

EDITOR-IN-CHIEF
Ilze Zigurs
University of Nebraska at Omaha

AIS SENIOR EDITORIAL BOARD

Guy Fitzgerald Vice President Publications Brunel University	Ilze Zigurs Editor, CAIS University of Nebraska at Omaha	Kalle Lyytinen Editor, JAIS Case Western Reserve University
Edward A. Stohr Editor-at-Large Stevens Institute of Technology	Blake Ives Editor, Electronic Publications University of Houston	Paul Gray Founding Editor, CAIS Claremont Graduate University

CAIS ADVISORY BOARD

Gordon Davis University of Minnesota	Ken Kraemer University of California at Irvine	M. Lynne Markus Bentley College	Richard Mason Southern Methodist University
Jay Nunamaker University of Arizona	Henk Sol University of Groningen	Ralph Sprague University of Hawaii	Hugh J. Watson University of Georgia

CAIS SENIOR EDITORS

Steve Alter University of San Francisco	Jane Fedorowicz Bentley College	Jerry Luftman Stevens Institute of Technology
--	------------------------------------	--

CAIS EDITORIAL BOARD

Michel Avital University of Amsterdam	Dinesh Batra Florida International University	Indranil Bose University of Hong Kong	Ashley Bush Florida State University
Evan Duggan University of the West Indies	Ali Farhoomand University of Hong Kong	Sy Goodman Georgia Institute of Technology	Mary Granger George Washington University
Ake Gronlund University of Umea	Douglas Havelka Miami University	K.D. Joshi Washington State University	Michel Kalika University of Paris Dauphine
Julie Kendall Rutgers University	Nancy Lankton Michigan State University	Claudia Loebbecke University of Cologne	Paul Benjamin Lowry Brigham Young University
Sal March Vanderbilt University	Don McCubbrey University of Denver	Fred Niederman St. Louis University	Shan Ling Pan National University of Singapore
Jackie Rees Purdue University	Thompson Teo National University of Singapore	Craig Tyran Western Washington University	Chelley Vician Michigan Technological University
Rolf Wigand University of Arkansas, Little Rock	Vance Wilson University of Toledo	Peter Wolcott University of Nebraska at Omaha	Yajiong Xue East Carolina University

DEPARTMENTS

Global Diffusion of the Internet Editors: Peter Wolcott and Sy Goodman	Information Technology and Systems Editors: Sal March and Dinesh Batra
Papers in French Editor: Michel Kalika	Information Systems and Healthcare Editor: Vance Wilson

ADMINISTRATIVE PERSONNEL

James P. Tinsley AIS Executive Director	Vipin Arora CAIS Managing Editor University of Nebraska at Omaha	Copyediting by S4Carlisle Publishing Services
--	--	---