120th ASEE Annual Conference & Exposition
FRANKLY, WE DO GIVE A D*MN
June 23-26, 2013

Paper ID #6409

Supporting Software Architectural Style Education Using Active Learning and Role-playing

Dr. John Georgas, Northern Arizona University

John Georgas is an assistant professor in the Department of Electrical Engineering and Computer Science at Northern Arizona University in Flagstaff, Arizona. He holds the Ph.D. and M.S. degrees from the Department of Informatics at the University of California, Irvine and the B.S. degree in Computer Science from California State Polytechnic University, Pomona. His research interests include self- adaptive software systems, software architecture, domain-specific architectures, social aspects of software engineering, software engineering education, architectural styles, and architectural description languages. His doctoral work focused on supporting self- adaptive software using an architecture- and policy-based approach with an emphasis on the robotics domain.

Supporting Software Architectural Style Education Using Active Learning and Role-playing

1. Introduction

One of the most critical aspects in the preparation of students for the rigor of the software engineering enterprise is developing and enhancing their software design skills. Architectural styles, which are refined sets of design decisions that elicit beneficial qualities in systems, are a key element to consider in design. The *client-server* architectural style, for example, guides the construction of systems by promoting a clear separation between server and client components, the latter primarily being consumers of server-provided functionality. Knowledge of architectural styles provide designers with useful starting points for their own designs as well as a better understanding of the effects that architectural compositions have on functional and non-functional properties of software systems—the *client-server* style, for example, promotes a clean separation of concerns but may give rise to scalability concerns as the number of clients grows. Unfortunately, lecture-based modes of instruction are particularly ill suited to fostering learning of architectural styles by failing to wholly capture the dynamism inherent in software operation, and by failing to provide an appropriate context for high student engagement.

To address these challenges in supporting architectural style learning, we have developed an activity that is intended to be more engaging and dynamic than traditional approaches: Our approach centers on couching learning about architectural styles in the context of a game where students role-play software components that cooperatively solve simple problems according to the guidance and constraints provided by particular architectural styles. Students are connected to each other by lengths of string and can only exchange messages along these style-specific communication links; the need and ability to solve problems is similarly allocated in a style-specific manner. Students then work together, playing their individual roles as software components to exchange messages, respond to requests, and resolve ambiguities. The game is primarily intended for use with undergraduate students in their junior year in the context of an introductory course in software engineering, or in their senior year in the context of a more in-depth course in software architecture and design.

This activity makes students an active part of realizing and exploring learning concepts by providing tangible representations of common software engineering idioms and activities as events within the game, such as when a piece of string snapping is equated to an interrupted or dropped network connection—this engages students in the game without resulting in loss of academic rigor in the treatment of the subject matter. This game-centric approach: (a) deeply adopts insights from active learning, making students an integral part of the learning process, (b) provides a dynamic, simulation-like context that is well suited to the dynamic nature of software, and (c) is modular and easily adoptable within existing curricular structures.

Initial evaluation efforts examine student attitudes and perceptions about the game by using a survey instrument consisting of both closed- and open-ended questions: Analysis of survey data provides strong indications that students find the game highly engaging and entertaining, particularly when compared to a conventional lecture-based approach. Furthermore, indicators

also show that students find the activity beneficial to their learning about architectural styles, particularly with regard to tangible explorations of non-functional qualities.

The remainder of this paper is structured as follows: Section 2 offers information on foundational concepts and related work. Section 3 presents our approach in detail, and includes examples of selected architectural styles as well as guidance for adopting educators. Section 4 presents evaluative results on student attitudes about the game, and Section 5 offers concluding remarks.

2. Background and Related Work

Our work is underpinned by the use of architectural styles as idioms for the transmission of software architecture knowledge, insights from learning theories, and by related work in using role-playing techniques in computer science and software engineering education.

2.1. Architecture and Architectural Styles

The design of software systems—ranging from the lowest to the highest levels of abstraction—centers on making design decisions about specific elements and the manner in which these elements interconnect. One of the most critical expressions of software design is commonly referred to as software architecture, and concerns the decomposition of systems using interconnected components and connectors[17]: Components capture computational behaviors, while connectors determine the ways in which components communicate and exchange information. As a result, the overall behavior of a system is determined not just by the functionality of each individual component, but also the characteristics of the architectural configuration defined by the manner in which these elements are interconnected.

While the notion of constructing software systems through assemblies of components and connectors is powerful and strongly represents fundamental ideas about abstraction and modularity[16], it leaves a great deal of unrestricted choice to the hands of the human designer, which may make the design space difficult to navigate. In response, the software engineering community has codified a collection of architectural styles, which are descriptions of compositional patterns and constraints on the assembly of architectural elements that induce desirable properties for particular types of systems or application domains[20]. By introducing style-specific constraints on component types and how these components interconnect, architectural styles codify sets of design decisions for the adopting designer. While they certainly provide a useful starting point for the design of individual systems, styles also act as a valuable knowledge transmission idiom since architectural styles are focused on capturing refined or distilled architectural experience[21] from past development efforts. As a result, architectural styles are a core component of the software engineering curriculum, and provide learners with a solid grounding for their own software designs. Descriptions of selected architectural styles will be presented in Section 3.2, alongside the detailed explanation of our role-playing activity.

2.2. Learning Theory Foundations

Our work is informed by a number of important learning theories: *Active learning*[4] describes pedagogical techniques that are aimed at increasing the level of interaction that a learner exhibits

with the material being learned, the instructor, and their fellow learners—particularly in comparison to a lecture-driven mode of instruction—and the active learning literature identifies games as important vehicles for fostering this type of learning. The roots of active learning can be found in the *learn-by-doing* approach[5], which identifies the interactions that learners engage in as a critical part of the learning process. Our use of a role-playing game, providing a context for and relying on interactions between fellow learners, strongly embodies these ideas.

Similarly constructivist ideas about learning have given rise to a number of related pedagogical philosophies and associated techniques: *Situated learning*[10], for example, identifies the importance of ensuring that the context in which knowledge is gained matches the context in which the knowledge will be applied. *Problem-based learning*[19] promotes the adoption of large-scale, open-ended problems, which is of particular importance in the context of software engineering education[14], as it has helped give rise to the increasingly ubiquitous adoption of real-world, long-term projects in computer science and software engineering curricular programs and motivates the importance of architectural-style learning as a pathway to better prepare students for the design problems these projects entail.

The idea of increasing learner engagement with the material, which our approach pursues, is also important in the context of learner motivation: The *Attention, Relevance, Confidence, Satisfaction* (ARCS) motivational model[9] identifies the named four key factors as fostering a high degree of motivation in the learning process. Our work is primarily intended to support learner attention by promoting a high level of active engagement with the content being learned. Importantly, learner motivation and engagement has been identified as a critical factor in university student success[2].

2.3. Role-Playing in Software Engineering Education

Other educators have also integrated role-playing activities into a variety of courses in computer science and software engineering curricula. In general, these integrations can be classified into two categories: one having students adopt the role of human participants in the software development process, and the other having students take on the role of software or hardware components in a larger system—our own approach falls into the latter category.

Role-playing games are commonly used to provide a context for students to adopt the roles they will play as professionals: One representative approach[18] focuses on students taking on the roles of software developers (for example, project managers, architects, requirements specialists, and developers) in order to interact with customers (played by teaching staff) and explore inconsistencies in understanding project requirements. Other approaches focus on exploring more specific types of software systems, rather than the social aspects of software engineering, such as work targeted toward process models that are well suited to large software product lines[22]. A related approach[7] is focused on the broader software enterprise, with students similarly adopting roles corresponding to conventional software engineering specializations, while also including roles related to software support and marketing.

In other uses of role-playing games, students take part in activities that have them take on the roles of software or hardware elements in systems, similar to our work with having students play

the roles of software components. In the "Classroom Computer" game[13], for example, students take on the parts of hardware elements and play out the sequences of interactions needed for basic computing tasks. Focused on exploring issues related to operating systems and concurrency, other educators have created games where students take on the role of processes that must coordinate to execute instructions[11]. Focused more on the object-oriented paradigm, another approach[3] has students role-play objects that exchange messages, which is similar to our architectural style game, but at a lower level of software design abstraction.

3. Approach

In modular software design, architectural styles provide a key idiom that is highly useful in imparting lessons from successful experiences and designs of the past to learners. The most important feature of architectural styles is that they go beyond simple narratives of design experiences, and capture design expertise that has been refined through careful reflection in an effort to codify important lessons. By providing students with a solid foundation in understanding the applicability, key characteristics, advantages, and disadvantages of architectural styles, educators can provide learners with valuable starting points for their own design activities as well as build expertise in identifying critical design trade-offs.

The instruction of architectural styles, however, remains challenging, primarily due to a fundamental disconnect between the dynamic nature of the software compositions that architectural styles model and the static artifacts most commonly used in instructional settings: diagrams that illustrate the basic structure of systems for a particular style—a view that focuses on showing the "shape" of the architectural style, and directional arrow augmentations that show "traces" of its operation. These techniques fall short, since they focus on using static views for illustrating inherently dynamic concerns: statically showing the shape of architectures is a good start but has limited utility, particularly in styles like peer-to-peer that may exhibit topological changes in their operation. Illustrations of traces only show a single exchange and are not well suited to capturing concurrency. These issues are compounded by the fact that this commonly used approach is lecture-based, and does not strongly promote student engagement.

Our instructional approach is based on couching learning about architectural styles in the context of a role-playing game simulating the operation of a software architecture: Students adopt the roles of software components and then cooperatively produce solutions to simple problems according to the communication and functional allocation constraints of various architectural styles. In essence, this allows for students to create a real-time simulation of the runtime operation of a software system, which helps expose and bring to life issues relating to the guidance provided by architectural styles on the structure of systems, the formative impact of style-specific constraints on functional allocation and communication, and the effects of these constraints on non-functional properties.

The key benefits of our approach include:

- *Active learning*: Students become an active part of the learning process through physical participation in an educational activity that inherently relies on interaction and collaboration with other learners;

- *Dynamic*: The game-like setting exposes concepts related to architectural styles in a dynamic manner that is better suited to the nature of software systems than the learning materials conventionally used; and,
- *Ease of adoption*: Our activity is modular and easily adopted: It relies on commonly and easily available materials, can be used to whatever extent individual educators desire, and can be combined with traditional teaching methods.

The following sub-sections provide a detailed description of the general game activity, a discussion of how the activity can be applied to selected architectural styles, and our perspective on issues related to curricular integration.

3.1. Detailed Description

The game activity begins with a short description of the basics of an architectural style, using a graphical depiction that illustrates connectivity between basic elements—this provides an initial grounding in the particular style, and is a useful reference point for students as the activity progresses. After this initial introduction, the instructor asks the participating students to stand from their desks and assemble as a group in either the front or the center of the classroom, depending on desk arrangement. The instructor then (acting as an architect) assembles the students in a composition using lengths of string to create connections between them in the pattern prescribed by each architectural style.

Once the initial connections are established, the students are given problems to solve in a manner that is customized in different ways for each architectural style. The problems revolve around solving a collection of simple arithmetic problems, such as addition and division. What makes the activity differ from one architectural style to another is that students are allocated the need and ability to solve these simple problems according to the constraints of the architectural style being studied. Consider, for example, an architecture structured according to the *client-server* style: In such a composition, the need to get answers to the arithmetic problems lies with client components—each student playing a client component is given a sheet of paper with the questions that they need to solve. However, the ability to solve problems, under the constraints of the *client-server* style, lies primarily with the student that plays the server component—this student is instructed that they have the ability to solve all arithmetic problems that they are asked to. The only way for the students playing client components to achieve their goal of having a complete set of answers is to request that the student playing the server component provide these answers to them—this closely mirrors the operation of *client-server* architectures.

The choice of such simple arithmetic problems is deliberate: The goal of the activity is to explore and learn about architectural styles, rather than particular problems in computer science or software engineering—the use of simple problems focuses learner attention on architectural style considerations, rather than any complexity found in the problems themselves. Furthermore, the simplicity of the problems being solved sets a very low barrier to entry, as students are not put in a position where they might be reluctant to participate because they feel that they are not well prepared for the problems begins solved—this encourages participation and creates an even playing ground among students of varying abilities.

Formal communication between components—that is communication that relates to asking for the solution of problems or providing answers to such questions—may only take place through the exchange of messages between students. These messages are captured on small pieces of paper, with sticky-notes being particularly helpful since they can be temporarily attached to surfaces or the students themselves. These pieces of paper can only be exchanged along the lines of communication established by the string connections, and always contain the name of the originating student on them in addition to other content dependent on the architectural style being studied: For the *client-server* style, for example, messages from clients to server will contain an arithmetic problem statement and messages from server to clients will contain the answer to a previously received problem statement. Other styles may involve message data that addresses higher-level concerns, such as in the *peer-to-peer* style that involves request forwarding so that messages must also maintain a routing chain of participant names. Of course, informal verbal communication between students takes place freely and (while unrealistic) is useful in resolving ambiguities or clarifying misunderstandings.

With lines of communication established, need and ability to solve problems allocated, and a mechanism for message exchange in place, the activity begins in earnest: Each student independently communicates with other students—according to the restrictions established by the architectural style being simulated—through the exchange of messages, with the ultimate goal of having every student possess the answers to the problems they have a need to solve. Importantly, the use of physical objects, such as string and sticky-notes, as tangible representations of abstract concepts is also aimed at supporting kinesthetic learning modes[6].

3.2. Style-Specific Discussion and Experiences

Since the manner in which the game plays out varies depending on the particular style being studied, the following sub-sections provide a discussion of style-specific concerns and experiences for a selected collection of architectural styles from those commonly studied at the undergraduate level in computer science or software engineering curricula. While we apply this activity to the study of a wide range of architectural styles in our software engineering and software architecture courses (including *layered*, *mobile-code*, *event-driven*, *publish-subscribe*, and a variety of *peer-to-peer* styles), exhaustively discussing the complete catalogue of architectural styles is outside the scope of this paper. The intent is for the selected set of architectural styles discussed to illustrate the richness of the game, particularly in terms of insights into how non-functional concerns are brought to the foreground.

3.2.1. Client-Server

The *client-server* (CS) architectural style[1] is composed of two general types of components: Servers are the primary providers of functionality and data, and they provide services to client components that request them. In general, connections are made between clients and a central server, while client-to-client connections are prohibited.

For our role-play activity addressing this style, each student playing the role of a client is connected to a single student portraying the server—we prefer using a single server, as opposed to more complex server arrangements, as it makes the game more amenable to exploring issues

of scalability. The need to solve problems is allocated to students that portray clients and the ability to solve problems is given to the student that plays the server, which parallels the conventional separation of concerns for the CS style. Student-clients are given sheets of paper with the problems that they need answers to, which are substantially similar from client to client. Student-clients are responsible for initiating and driving communication with the server by exchanging sticky-notes with problems that they desire answers to. Server-students solve the problems they are requested to solve and return the answers to requesting students, with play ending when all student-clients have a complete set of solutions.

One of the most disadvantageous characteristics of the CS architectural style is that the server is a single point-of-failure component that needs to provide services to every single participating client, even as the number of clients increases. This characteristic is particularly well suited for exploration in the game: First, the instructor begins by asking student-clients to begin "flooding" the server with messages, by creating sticky-note messages and handing them off to the server in quick succession—this simulates an increase in request traffic. At the same time, new student-clients are added to the architecture and begin sending their own requests. These changes put a great deal of stress on the student-server: First, they need to maintain an increasing number of connections, which becomes difficult as it involves physically managing the connecting lengths of string. As the number of messages increases, the student-server also begins having difficulty organizing and managing the timely return of client requests. From a pedagogical perspective, mishaps are the most interesting occurrences: dropped strings equate to connection losses, while dropped or misplaced sticky-notes can be equated to unfulfilled requests due to buffer overflows. The instructor can also physically remove the student-server from the architecture or cut all connections to the server, leaving behind student-clients that are ultimately unable to solve their assigned problems; this illustrates the single point-of-failure aspect of the architectural style (students tend to find this rather humorous, so it is a rather engaging "trick").

3.2.2. Blackboard

The *blackboard* (BB) architectural style[15] consists of a central repository of problem statements and data, referred to as the blackboard, which interacts with a number of knowledge sources that are able to provide full or partial solutions to the problems contained in the blackboard. Changes in the blackboard data store may trigger further action from participating knowledge sources.

In our activity, the BB style's connectivity is similar to that of the CS style: One central student acts as the blackboard and is directly connected to a number of students that act as knowledge sources. The allocation of need and ability to solve problems, however, differs drastically: In the BB style, it is the central student-blackboard that is given the need to solve problems, while the ability to provide answers is distributed among student-sources who are simply instructed as to what kinds of arithmetic operations they can provide solutions to. Communication in the BB style game is driven by the student-blackboard: This student notifies—through message exchange—student-sources of the problems that need solutions, simulating a common mode of BB operation with the blackboard pushing requests to knowledge sources. If student-sources have been allocated the ability to solve the particular problem, they communicate the answer to the blackboard; if not, they simply do not respond. The game continues until the student-blackboard has answers to all of the problems given to them.

The characteristics of the BB style provide a good opportunity to begin exploring issues related to decentralization; despite there being more inherently decentralized styles, BB has an easily understandable structure. In the BB style, a core issue of decentralized control is whether the collection of knowledge sources available to the system is sufficient for producing a complete set of solutions to the problems posed to the student-blackboard. The instructor can explore this issue by initiating the game without allocating a specific function to any of the student-sources (omitting, for example, addition from the architecture) or by disconnecting and removing a source while the game is ongoing. This can lead to a situation where the game will, in essence, never have an ending condition, as a complete set of solutions can never be achieved, illustrating to students the non-deterministic nature of decentralized architectures.

3.2.3. Pipe-and-filter

The *pipe-and-filter* (PF) architectural style[20] defines a linear data flow of components (filters) that accept streams of data and incrementally output transformed streams to other components in the architecture (pipe). The style is useful for data processing applications, such as image or telemetry processing, but is also found in the composition of low-level tools in operating systems (for example, a basic application is Unix's | operator).

For our activity, the connections between students for the PF style are straightforward: Student-filters are arranged in a left-to-right linear chain with only a single connection between students; the leftmost student does not have an input connection, while the rightmost lacks an output connection. To simulate the operation of PF systems as stream transformation architectures, the allocation of functionality differs slightly from that of other style games: Each student is allocated the ability to perform a transformation rather than only performing an arithmetic operation (for example, adding 2 to the input, or multiplying the input by 3). For this architectural style, there is no explicit need to solve problems: the game continues as long as values are provided to the architecture: we use a volunteer student to be the source of input values, and the rightmost student-filter simply places the sticky-notes that capture the transformed input on a desk. Communication throughout the PF assembly of students begins with the volunteer student providing a series of input values to the leftmost student-filter—we prefer to begin by using a simple ascending count, for the sake of simplicity. As each student receives an input value, they apply their individually assigned transformation and forward the result to the next student they are connected to in the PF chain. The game continues as long as input values continue to be provided to the architecture.

In terms of studying non-functional properties, PF architectures are useful in examining the effect of performance and reliability of individual components on the system as a whole. This draws from a core disadvantage of the PF architectural style: since filters are connected in a linear chain, a failure or delay in one component interrupts the operation of every subsequent component in the linear data flow. During the game, this can be illustrated by asking the student providing the input values to the PF assembly to do so more rapidly—inevitably, this leads to delays as individual student-filters begin struggling to keep up with the pace, and subsequent student-filters are left with nothing to do while waiting for the previous component's output.

3.3. Adoption and Curricular Integration

The game activity we describe is primarily intended for integration into introductory software engineering courses, which are a common component in most computer science and software engineering programs. Usually offered in the end of the second or beginning of the third year, for conventionally structured four-year programs, this type of course serves as an introductory exposure to basic software engineering principles. For our own offering of this course, the topic of high-level component design and architecture consists of a total of four and a half contact hours, with three of these contact hours devoted exclusively to the study of architectural styles—follow-on modules focus on other aspects of design, such as object-oriented design patterns. After smaller scale initial deployments, we currently use the game activity we present in this paper throughout these three contact hours as the core element of our approach to the instruction of architectural styles (except in cases of evaluative studies, as will be discussed in Section 4).

The time needed to introduce, setup, and run through the game activity ranges between 10 and 15 minutes per architectural style: Student selection will consume about 1 minute and our experience is that four to five students is a good number of participants; we ensure that we ask for a fresh set of volunteers for each architectural style we cover so that as many students as possible have the opportunity to participate, though we do not mandate participation, if a student is unwilling. Initial setup for each architectural style will consume roughly 2 to 3 minutes, and involves the instructor making connections between student-components, explaining the basics of the architectural style being discussed in the process, and appropriately allocating problems and the ability to provide solutions in a style-specific manner.

Once this is done, game play consumes between 7 and 12 minutes: The amount of time spent during game play is highly dependent on the depth of discussion that the instructor wishes to pursue regarding non-functional properties, such as performance, reliability, and decentralization. For example, we frequently pause the game in order to talk in detail about the unreliability of network connections and (literally) cut connections between student-components so that we tangibly demonstrate the effects of lost connections on architectural compositions. More interestingly (and humorously), we also sometimes ask certain students to act maliciously in certain ways. In studying peer-to-peer (P2P) architectures for example, we often ask that one student randomly modify messages that they are forwarding for other student-peers, to study the security and trust characteristics of such architectures.

Instructors can integrate our approach into their courses in whatever way they deem appropriate. This may entail using the game for every architectural style to be covered, which has the advantage of being consistent but may represent a larger time commitment than might be desirable—this is our preferred mode of use, since our course design places substantial emphasis on the study of architectural styles and is built around the time commitment needed. Alternatively, instructors may only use the game for a subset of the styles to be covered, which still provides the benefits of active learning and builds intuition and understanding that carries over into the study of other styles that are presented using more conventional techniques. We sometimes use this selected-deployment mode for learning about variants of architectural styles by, for example, using the game for *event-based* architectures but discussing the variant of *publish-subscribe* architectures[21] without the game activity.

The supplies needed for this game are inexpensive and readily available: Lengths of string are used to form connections between students, and a pair of scissors is necessary for creating lengths of string for the game's initial setup while also being useful for the mid-game instructor interventions described. Sticky notes and markers are also needed, so that students are able to exchange messages. Most of the pre-class preparation is invested in preparing and printing problem sheets, which demand some customization for each style being studied. For example, for the styles discussed in Section 3.2, the CS style requires a number of sheets with problems to match the desired number of clients to be included in the architecture, the BB style requires only a single sheet of problems to be stored by the blackboard component, while the PF architecture requires no premade problem sheets.

4. Evaluation

In our evaluation of the activity we describe in this paper, we have thus far focused on the self-reported attitudes and perceptions of students about the game as well as their perceived gains in learning, while ongoing work is progressing toward evaluating objectively measured gains in skills. Based on quantitative and qualitative data collected thus far, we have found that students respond positively to the activity, enjoy their participation in it, and prefer it to a more conventional, lecture-based approach—the following sub-sections describe our basis for these assertions, discussing the methodology we used, metrics establishing attitude, and both positive and negative feedback we received from students.

4.1. Methodology

The data we report is drawn from a survey instrument deployed in the Fall 2011 offering of our software engineering course, which is the latest course to benefit from the most recent iteration of our game's design. The cohort consisted of 31 students, all of whom were computer science majors: 16 of them were seniors, 13 juniors, 1 was a sophomore, and 1 a post-baccalaureate student earning a second degree. Of the 31 students in the cohort, 28 returned a completed survey instrument, for a 90% response rate to our survey.

So that we could provide students with a point of comparison to more accurately evaluate the architectural style game, we divided our instruction mode for this course offering: half of the architectural styles were discussed using the game activity, and half were presented using a lecture-based approach. The paper-based survey instrument was anonymously deployed in the class session immediately following the completion of the architectural style module and students had 15 minutes to complete the instrument. Our survey instrument design was guided by best practices for survey-based evaluations, with a useful survey found in work by Kasunic[8].

4.2. Student Attitudes

The first part of the survey instrument consisted of closed-ended Likert[12]-type questions allowing responses along a five-point scale, with options ranging from "strongly agree" to "strongly disagree" and including an "undecided" option. These questions targeted the following aspects of student perception and attitude toward the architectural style game:

Table 1. A summary of responses from survey items targeting student perceptions and attitudes about our architectural style game activity.

Survey Item	Frequency Count					Weighted Mean
	Strongly Agree (5)	Agree (4)	Undecided (3)	Disagree (2)	Strongly Disagree (1)	
(S1) Active	14	14	0	0	0	4.50
(S2) Participatory	10	15	1	2	0	4.18
(S3) Engagement	13	14	1	0	0	4.43
(S4) Enjoyable	11	14	2	1	0	4.25
(S5) Improvement	5	21	2	0	0	4.11

- *(S1) Active*: This question targeted the degree of student agreement with the assertion that the game supports active learning better than a lecture-based approach.
- *(S2) Participatory*: The focus of this question was on student agreement with the assertion that the game provided a setting that was more participatory than a lecture-based approach.
- *(S3) Engagement*: This question examined whether students agree that the game was more engaging than a lecture-based approach.
- *(S4) Enjoyable*: This question targeted student agreement with the assertion that the game fostered a higher level of enjoyment as compared to a lecture-based approach.
- *(S5) Improvement*: This question examined student agreement with whether the game activity improved their learning about architectural styles.

Table 1 summarizes the results from this part of the survey, showing frequency counts and weighted means for the responses to each survey item; the weighted mean was calculated using the values indicated in the table for each ordinal response category.

The results of this part of the survey instrument are encouraging, and show that students generally respond positively to the game: Responses to item S1 show that 100% of students surveyed either “agree” or “strongly agree” that the game provided a more active mode of instruction, as compared to a lecture-based approach. This provides a strong indication that the activity satisfies our goal of promoting active student participation in the learning process. Results from item S2 show that 89% of students found the activity to be more participatory than a lecture-based approach. While this certainly provides an indication that the activity succeeds in increasing the level of participation during architectural style learning, we do note that 3 students were either “undecided” or “disagreed” that this was the case. One possible explanation for these results may stem from the fact that we split our coverage of architectural styles between a conventional lecture-based approach and use of the game, which resulted in not every student in the class having the opportunity to actively participate in the game.

For item S3, we note that 96% of students responded that they “agreed” or “strongly agreed” that the game was more engaging than a lecture-based approach, which provides support to the conclusion of having succeeded in our goal to use a more engaging mode of instruction for

architectural styles. Item S4 similarly provides positive indications that the game activity was more enjoyable than a lecture-based approach, with 89% of students "agreeing" or "strongly" agreeing with this statement. With item S5, we intend to capture student perceptions about their own learning: With 93% of students "agreeing" or "strongly" agreeing that the game provided improvements in learning about architectural styles, there are strong indications that students felt that the game activity was beneficial—while the weighted mean is lower than that of other responses, it remains strongly positive.

4.3. Open-Ended Feedback

The final part of the survey instrument solicited students for open-ended feedback on their thoughts about the game activity, its level of usefulness to learning about architectural styles, how it compares to a lecture-based approach, and areas of possible improvement. We organize a subset of the open-ended feedback we received along three trends: positive reactions, suggestions for improvements mechanics, and fundamental drawbacks of the approach.

4.3.1. Positive Reactions

The open-ended feedback from most students was positive, mirroring the positive reactions captured in the Likert-type survey items previously discussed. A number of students focused their feedback on how engaging the activity is: One said that the game provides "far more engagement than note taking," another that "[the game] was a fresh and entertaining learning experience," while another stated that the game "woke me up and maintained attention." This feedback reinforces the indicators from the closed-ended survey items, and increases our confidence that the activity is successful in being a more active and engaging instructional mode.

Students also focused their comments on the benefits of the approach for learning about architectural styles: For example, students commented on how the game allowed them a clear view of software architecture and its dynamic operation, by identifying that the game allowed them to "[have] an easier time visualizing the architecture," that "it was nice to see the styles step by step," and that the "most useful was anything that included sticky note passing, [it] helped to see what happens where there is a lot of information traffic."

Other students focused on the exploration of specific non-functional properties that the game fostered, by saying that "the most useful part to me was being able to understand the points of failure better," that "it was fun watching the demonstrations especially in relation to load increases and scalability," that it was useful "visualizing the architectures in an organic fashion, it was funny when connections were lost, to me," and that "the scalability was exaggerated by the human element and was very clear because of it." These responses are particularly encouraging to us, as they provide indications that students can quickly grasp the learning benefits of the game approach, which is an evaluative aspect that transcends the enjoyment gained by the activity to address the desired learning outcomes at the heart of architectural styles.

Finally, we identify a trend in student responses that points out the appeal of the architectural style game for a variety of learning modes: For example, one student identified that the game supported collaborative and team-based learning by saying that their favorite part of the activity

was “being able to work as a team with a small group of fellow students while demoing a style.” Others appreciated the visual and auditory elements of the game, which are a natural by-product of physical participation and informal verbal communication between students while the game is ongoing, by saying that the game is “more visual which helps me learn, visual examples seem to be hard to come by in CS,” and “it can be a huge help for visual learners, it also holds some benefits to auditory learners.”

4.3.2. Improvements to Mechanics

Soliciting open-ended responses also provided us with important student-driven suggestions for improvement. Two important issues emerged from student feedback: The first is that it is sometimes difficult for observers to maintain awareness of the details of game-playing students’ interactions. For example, one student stated that it is “sometimes hard to see connections and info passed through notes” and another suggested, “instead of passing notes, maybe pass different colored notes? something that makes it easier to see how data travels.” Another student recommended using a medium other than sticky notes for information exchange and stated, “perhaps find a better method than sticky notes, throw balls with numbers?” We plan to re-visit the design of our game and our choice of materials in light of these suggestions, though moving away from sticky nodes—a medium that is useful for quickly generating representations of custom messages—would reduce the game’s expressiveness. The second trend is that students were disappointed at some of the overhead involved in the initial setup of each style-specific game; one suggested that participants be “given specific instructions ahead of time” and another that we should “decide participants before starting the activity; this would decrease down time.”

4.3.3. Drawbacks and Negative Reactions

While the overwhelming number of open-ended responses was positive, students also identified a number of fundamental drawbacks. The first drawback is that participants in the game are unable to take personal notes while the game is ongoing, since they are actively engaged in simulating software components: Two students point out that their least favorite part of the activity was “not being able to take notes well while participating” and “not being able to take good enough notes,” respectively. Another student said that they “personally found it harder to retain what I learned on a specific architecture when I was one of the participants.” This issue is difficult to address while maintaining the active nature of the game activity, as providing the opportunity for participants to take notes during the game would introduce pauses, which interrupt game-play and delay completion. We are investigating improved pre- and post-game elements to address this, such as a short period where the instructor summarizes how the game went during which student participants have a chance to catch up on their note-taking.

A second student-identified drawback centers on the fact that the game gives up a degree of formality, as compared to a lecture-based approach that delves into minute details during the presentation of an architectural style: One student said that “some of the aspects were hard to understand without going through the description in a more formal way,” while another reported that “some fine details are lost that could have been retained in a lecture.” This feedback has motivated us to investigate how to best integrate more traditional materials, such as diagrams and outlines of stylistic constraints, during the game activity.

5. Conclusion

Architectural styles are a core element of the software engineering curriculum, as they convey information about design features that have been established to be beneficial for specific types of systems or application domains. As a result, they are valuable to the fledgling designer by providing important starting points and valuable insights into beneficial design practices and the effects of design decisions on software non-functional properties. Conventional methods for architectural style instruction, however, tend to use static artifacts to represent inherently dynamic aspects of software operation and lack a high degree of student engagement. We have developed an instructional activity to address these challenges, by creating an architectural style role-playing game that has students play the parts of software components interacting with each other according to the guidance and constraints provided by specific architectural styles being studied. Our approach strongly adopts insights from active learning, is intended to be highly dynamic and engaging to students, and is easy to adopt within existing curricular structures.

Initial evaluation results focusing on student perceptions of the game and its benefits to learning are highly positive: Student frequency counts and weighted means from Likert-type survey items strongly indicate that the architectural style game activity is more active, participatory, engaging, and enjoyable than the lecture-based equivalent. These results are further reinforced by open-ended student feedback, which has also been valuable in exposing potential areas of improvement in our game design, such as better pre- and post-game discussions that better prepare and debrief participating students. While further study is needed to objectively measure potential increases in learning through outcome-based assessment means, initial results also indicate that students perceive the game as fostering important improvements in learning.

6. Acknowledgements

We extend our gratitude to the Computer Science students of Northern Arizona University for participating in this work and to André van der Hoek for the inspiration behind early versions of this role-playing activity. This research is supported in part by the National Science Foundation under Grant numbers CCF-1016134 and CCF-1017408.

Bibliography

[1] Andrews, G. Paradigms for Process Interaction in Distributed Programs. *ACM Computing Surveys.* 23(1), p. 49-90, 1991.

[2] Astin, A.W. *What Matters in College? Four Critical Years Revisited*. Jossey-Bass, 1993.

[3] Bennedsen, J. and Caspersen, M.E. Programming in context: a model-first approach to CS1. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'04).* p. 477-481, ACM Press. 2004.

[4] Bonwell, C.C. and Eison, J.A. *Active Learning: Creating Excitement in the Classroom*. Jossey-Bass, 1991.

[5] Dewey, J. *Democracy and Education: An Introduction to the Philosophy of Education*. The MacMillan Company: New York, NY, 1916.

[6] Fleming, N.D. and Mills, C. Not Another Inventory, Rather a Catalyst for Reflection. *To Improve the Academy.* 11, p. 137-149, 1992.

[7] Henry, T.R. and LaFrance, J. Integrating role-play into software engineering courses. *Journal of Computing Sciences in Colleges.* 22(2), p. 32-38, 2006.

[8] Kasunic, M. *Designing an Effective Survey.* Carnegie Mellon University, Report, 2005.

[9] Keller, J.M. and Suzuki, K. Use of the ARCS Motivation Model in Courseware Design. In *Instructional Designs for Microcomputer Courseware* Jonassen, D.H. ed. Lawrence Erlbaum: Hillsdale, NJ, USA, 1988.

[10] Lave, J. *Cognition in Practice: Mind, Mathematics, and Culture in Everyday Life*. Cambridge University Press: Cambridge, UK, 1988.

[11] Leverington, M., Yüksel, M., and Robinson, M. Using role play for an upper level CS course. *Journal of Computing Sciences in Colleges.* 24(4), p. 259-266, 2009.

[12] Likert, R. A Technique for the Measurement of Attitudes. *Archives of Psychology.* 140, p. 1-55, 1932.

[13] McNichols, K.H. and Fadali, M.S. The classroom computer: A role-playing educational activity. In *Proceedings of the 29th Annual Frontiers in Education Conference (FIE'99).* 3, IEEE. 1999.

[14] Navarro, E.O. and van der Hoek, A. On the Role of Learning Theories in Furthering Software Engineering Education. *Software Engineering: Effective Teaching and Learning Approaches and Practices. HJC Ellis, SA Demurjian and J. Fernando (Eds.). IGI Global.* p. 38-59, 2009.

[15] Nii, P.H. The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures. *AI Magazine.* 7(2), p. 38-53, 1986.

[16] Parnas, D.L., Clements, P.C., and Weiss, D.M. The Modular Structure of Complex Systems. *IEEE Transactions on Software Engineering.* 11(3), p. 259-266, March, 1985.

[17] Perry, D.E. and Wolf, A.L. Foundations for the Study of Software Architecture. *ACM SIGSOFT Software Engineering Notes.* 17(4), p. 40-52, October, 1992.

[18] Redondo, R.P.D., Vilas, A.F., Arias, J.J.P., and Solla, A.G. Collaborative and role-play strategies in software engineering learning with Web 2.0 tools. *Computer Applications in Engineering Education.* 2012.

[19] Savery, J.R. and Duffy, T.M. Problem Based Learning: An Instructional Model and its Constructivist Framework. In *Constructivist Learning Environments: Case Studies in Instructional Design* Wilson, B. ed. p. 135-148, 1996.

[20] Shaw, M. and Clements, P. A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems. In *Proceedings of the Computer Software and Applications Conference.* p. 6-13, August, 1997.

[21] Taylor, R.N., Medvidovic, N., and Dashofy, E.M. *Software Architecture: Foundations, Theory, and Practice*. John Wiley & Sons, 2009.

[22] Zuppiroli, S., Ciancarini, P., and Gabbrielli, M. A Role-Playing Game for a Software Engineering Lab: Developing a Product Line. In *Proceedings of the 25th IEEE Conference on Software Engineering Education and Training (CSEE&T).* p. 13-22, 2012.