Raspberry Pi Creativity: A Student-Driven Approach to Teaching Software Design Patterns

Chad Williams, Stan Kurkovsky
Department of Computer Science
Central Connecticut State University
New Britain, Connecticut, U. S. A.
{cwilliams, kurkovsky}@ccsu.edu

***Abstract*—One of the primary goals of computer science education is preparing students to apply what they learn in the classroom to any problem they may encounter in the future. Due to the rapidly changing landscape of technology, regardless of the comprehensiveness of their coursework, at some point students will be required to apply their knowledge to a context they have not encountered before, and understand how what they have learned applies. One of the content areas where this is particularly relevant is helping students recognize the variety of technologies and applications in which object oriented design patterns can produce significant practical benefits. To foster this aptitude, we sought to leverage the students' own creativity with a Raspberry Pi and a multitude of sensors, displays, cameras, and actuators to boost exploration of technologies and student understanding of how course concepts apply outside of constructed examples. This paper presents the experience of using student creativity with the Raspberry Pi to drive their learning in an upper level software design patterns course.**

Keywords— Raspberry Pi; sensors; student projects

I. Introduction

A common challenge in any educational context is helping students progress from knowing how to apply concepts they are taught in constructed examples to seeing how these situations emerge in practice and discovering both the reason for the pattern as well as how to apply it. Students can be guided to identify these examples in the real world and learn how the concepts apply with classroom examples. An ideal situation, however, would be for the students to not only encounter these real situations on their own, but also find that they need to identify how to solve them, thereby deepening the understanding of how the concepts they are learning can be applied in a variety of contexts.

Software development is a process combining technical, social, and creative elements. According to [15], "Software Engineering is primarily a social and creative process, where the creativity, skill, and co-operation of developers, users, and procurers determine the quality and effectiveness of the developed software." Being able to leverage students' creativity in a similar way was the motivation for shifting our approach to teaching our upper level Software Design Patterns class. Specifically, we wanted to use the students' own creativity to motivate their learning and encourage them to encounter and solve common software design challenges. To create this scenario, the class was redesigned: rather than strictly assigning software-based projects, student teams were given Raspberry Pi computers and a large collection of sensors, displays, cameras, and actuators and were told to create their own project. The only restrictions imposed on their work was that the project must use the majority of design patterns covered in the syllabus in ways that made sense within the application context.

This work describes the experience of using student creativity in a Raspberry Pi environment to stimulate learning the practical applicability of content in an upper-level software design patterns course. We examine how the platform was leveraged to drive experiential learning of the reasons specific design patterns are used as well as the need for good software development practices. We provide a sampling of projects developed by student teams, and discuss a number of lessons learned that could benefit other educators looking to incorporate a similar approach in their coursework.

II. Background Work

A. Teaching design patterns

Reusable software design patterns and their role in developing higher quality, more maintainable, and extensible code has been well studied [16,39,42]. By the late 1990s, the significance of including the study of design patterns had been argued as not only being critical, but also even an essential component of any computer science curriculum [3]. As Astrachan et al. noted, one of the reasons studying design patterns was deemed important is because it helps students shift their mind set when they approach software problem solving. Instead of simply coding a solution, students consider the broader use or potential use of an area of code and design a solution that takes this context into account. Incorporating design patterns within CS curricula has been examined in terms of effectiveness and understanding the greater context from the beginning of their studies, in courses like CS1 and CS2 [1], all the way to being covered in the last year of an undergraduate program [7]. Among these studies, there is a general recognition that while students can recognize patterns early in the curriculum through guided lessons, as students advance through their course work, there is more opportunity to let students lead this discovery. Common approaches to instruction range from having students directly compare a pattern-based solution to a more traditional solution [6], to supplying a problem and asking students to identify and implement the appropriate pattern [37,41]. A challenge in this area is helping students bridge the

978-1-5090-5920-1/17/$31.00 ©2017 IEEE

gap from more directed learning to identifying the appropriate pattern given a set of more general requirements [32]. One of the ways to approach this is to motivate the learning by placing the problem in the context of something the student is interested in to encourage the pursuit of the end result. A frequently used motivator of this learning approach has been games and game design. Examples of these include, the Game of Life [41], *n-in-a-row* games [19], and arcade style games [18]. A common reason given for using games for teaching, beyond the motivating factor, is that they often involve the need for significant collaborations and interactions of patterns within the scope of the game which [18] notes as being a key element in students being able to appreciate complex design patterns. The study goes on to note that while contrived projects can sometimes allow students to see benefits, in order to be able to synthesize the use of patterns on their own, students need to be taught through multi-pattern collaborations.

B. Creativity as a learning tool

For our class we sought to build upon many of the elements that made student learning successful, such as leveraging student motivation in a multi-pattern project, but in a way to take even more advantage of student creativity in the process. The goal of putting students completely in charge of defining the direction of their project, including how to incorporate design patterns, was to allow the student to take control over their own learning. We sought to take advantage of the benefits Harel and Papert observed in constructionist learning, where software design itself was the learning environment [1]. While Harel and Papert's work examined this type of student-led learning in fourth graders, they noted that many of their observations likely translated to this type of learning in general. Key observations they saw were that by students being in control of their own learning, they developed better problem solving skills. When students were in charge of defining what they wanted, they were empowered to discover problems they wished to solve and develop an awareness of the skills and processes needed to resolve the problem they proposed. This helped them develop cognitive flexibility in looking into different solutions and being willing to work on problems that were more difficult without giving up because they were the ones defining the motivation. This process resulted in students being more articulate about the planning and design tasks, which is our ultimate goal: not only can the students use the patterns, but also they know why they want to use a particular pattern to solve a problem.

The study of computing, be it in the form of computer science, information technology, or software engineering, usually attracts people who have a desire to create and experience new things, which can take the form of games, web or mobile apps, robots, programmable controllers, or palm-sized computers. Creativity, however, is usually not among the terms commonly associated with the stereotypical notion of computing [33].

However, when creativity has been incorporated in CS1 the results have been higher scores in knowledge, self-efficacy in applying the knowledge, and increased study time [38]. Efforts have been described in implementing a design studio within computer science programs that would provide a sandbox for developing creative skills of students within the framework of human-computer interaction courses [13,20,22].

The significance of creativity and creative thinking has been explored particularly well in the area of requirements engineering. In particular, creativity is crucial in discovering new requirements and has been shown to lead to better overall solutions [29,34,36].

Media computation approach has been shown to be an excellent tool for allowing students explore and practice their creativity in introductory computer science courses, especially for non-majors [22]. Another potential benefit of leveraging creativity is creating courses that are more appealing to women [35].

C. Developing creative analysis skills

Computer science education literature often mentions creativity as an integral component of design skills with the false assumption that if students can design software systems, that automatically makes them creative (e.g. [2]). This approach, however, does nothing to stimulate students' divergent thinking skills. In a typical scenario involving a student project with a design component, there is typically exactly one candidate solution, which never gets brainstormed or evaluated against other possible solutions, which is then implemented in a step-by-step fashion prescribed by the project guidelines.

Although there is no commonly accepted strict definition of "creativity," researchers agree that this term refers to the ability to identify and solve problems in novel ways [12,40]. There is also a consensus that creativity is not necessarily an innate ability, but rather a trait that can be encouraged and developed [27]. Creativity techniques can help support and stimulate creative thinking and generation of ideas from a broad range of activities [4]. Generation of multiple ideas leading to a range of possible solutions to the problem at hand requires *divergent thinking* skills, in contrast to *convergent thinking* that helps generate 'the right answer' by following a well-known sequence set of logical steps leading to a single solution [21].

Computer science education seems to foster mainly the convergent thinking skills as we encourage our students to solve logical and computational problems that usually have only one correct answer. On the other hand, real-world problems that are being solved by modern software engineers require good skills in both convergent and divergent thinking. Obviously, a software engineer well-skilled only in convergent thinking will not be able to come up with an innovative solution to a problem that cannot be solved using conventional approaches. At the same time, a software engineer excelling only in divergent thinking will be able to generate many novel ideas, but they may lack the convergent thinking abilities needed to carefully evaluate and refine different ideas in order to compare their different qualities and to fully implement these ideas to create a workable solution.

A broad range of techniques for generating new ideas has been described in the literature. Some examples include Osborn's Checklist [30], often referred to as SCAMPER—Substitute, Combine, Adapt, Magnify, Put to other use, Eliminate, and Rearrange—as the methods to change an existing idea; Crawford's Attribute Listing [9], which suggests listing the properties of the object to be improved and systematically trying out different variations; and a number of Lateral Thinking

techniques introduced by de Bono [10], such as random stimulation, provocation, or harvesting. Such exercises not only encourage creative thinking, but also force students to take a different perspective at the subjects at hand. These approaches usually work well as a starting point for a discussion that is enjoyable for all involved. These open-ended exercises all serve the same purpose: they introduce novel ways of looking at problems that may or may not have existing or meaningful solutions.

Employing a combination of divergent and convergent thinking helps students evaluate a range of ideas from a number of perspectives, make observations, evaluate possible designs, and draw conclusions. This divergent thinking process is then repeated over a number of iterations following a path converging on a desirable solution [5].

Creativity and innovation play a crucial role in the development of entrepreneurship skills, which are receiving a renewed attention in many computing and engineering programs [11,17]. Being able to generate new ideas leading to commercially successful products or services is a core principle of entrepreneurship. Exposing students to entrepreneurship skills may go hand-in-hand with fostering creative skills that can be applied equally in both technical and business areas.

The importance of creativity in computing has been emphasized by incorporating it as one of the big ideas in the new Computer Science Principles course curriculum [8]. The emphasis on creating computational artifacts underscores the nature of the discipline focused on designing and developing new things using computers and technology, including web sites, video games, and/or digital media. The aim of this new introductory curriculum is to reframe computing as an inherently creative discipline focused not on routine work, but on extending traditional forms of human expression with help of technology [28]. Although most computer science educators do not need convincing that our discipline does provide a venue for expressing a degree of creativity, do we actually teach more advanced computing concepts and topics in a way that allows students to be creative? What about computational algorithms? What about designing a database schema, or a network layout, or a hierarchy of classes in an object-oriented design? Although creativity certainly plays a central role in creating a solution, do computer science majors feel that their field of study allows them to express their creativity? Do they feel the same way about their future jobs—will they be able to be creative while writing code, designing software systems, working with clients five or ten (or more) years from today? Since many computer science students pursuing employment in the industry will most likely choose a career path in software development or software engineering, we chose to explore student creativity in our upper level design patterns class offered to 3rd and 4th year students at a large regional university in the US.

III. Course goals

The course we chose to apply this approach to was our upper level software design patterns course. The instructor had taught this course three previous times at two different universities using software-based projects. Students taking this course are required to have completed core coursework in object-oriented (OO) programming and data structures. The focus of this course is advancing the students' mindset to move from simply programming a solution towards developing an understanding of more sophisticated design patterns, when they should be used, and being able apply them in practice. In addition, we also address best practices for things like source control, automated testing, and integrating code bases when multiple people are involved in a software development project.

Within this context, by using a creativity-driven approach, we put the students in charge of designing a semester long project and managing requirements that changed over the course of the semester. For example, a more traditional instructor-led approach might give the initial requirements and indicate what areas are likely to change and ask students to design an appropriate solution that considers this flexibility and then introduce these new requirements at a later point in the course. In past offerings of this course, many students would either have difficulty grasping how someone could know something needed to be flexible without having specific requirements, or would have questioned whether this type of scenario is even realistic. Thus, a primary goal in incorporating a creativity-driven approach was to put students in this exact situation where they had a vision of what their eventual general goal would be, but they would need to figure out the details along the way. For example, they might identify some aspect of functionality that would be fun to add a number of enhancements if they had time. Our hope was that a natural result of this type of creative vision would be students anticipating places within the project they would want flexibility, thus driving their need to identify the appropriate solution for what was obviously a real scenario.

IV. The platform

One might question our choice of the Raspberry Pi and associated sensors instead of simply giving the students the freedom in choosing a project platform. A number of factors led to the selection of the Raspberry Pi. A similar creativity-driven approach actually had been used in previous iterations of this class. Specifically, students had been given very loose requirements to choose between designing an arcade style game or to create a "life simulator" where organisms could dynamically evolve. With this loose guidance, they were given creative freedom to interpret what they wanted that to mean. With this approach, students also learned through being motivated to extend their learning to explore how to incorporate features they thought would be cool; however, for the most part, student groups tended to stay within their comfort zone by using programming languages they previously knew. The majority of exploration was using graphics libraries that had interfaces that were fairly easy to integrate and only had a limited number of places that really pushed students to consider the benefits and tradeoffs of using one pattern over another for a specific place in the application.

Given this context, using the Raspberry Pi platform with Grove components presented a lot more opportunities that required students to stretch their comfort zone. Regardless of what direction the students chose, their path was likely to require integration of multiple programming languages and interfaces that would benefit from applying the design patterns they were going to learn. For example, while the wide range of Grove components are well documented, at the time this class was

offered, the sample code for using the majority of these components was either in Arduino Sketch files or Python interfaces. By requiring students to make their main code base in Java, the result was that while students could choose whichever components they wanted, they typically had to make one of three choices:

- Figure out how to use patterns to integrate different programming interfaces, often using patterns in both programming languages;
- Learn how patterns could be applied to wrap system command line calls to appear as any other Java interface; or
- Study how the code in other languages interacted with the hardware to provide specific functionality, mimic the interaction with the hardware using the Pi4J library[1], and finally, apply design patterns to create their own clean interface to hide the underlying complexity.

Regardless of which approach students took, it would require them to learn how to apply what they were learning to a new context.

Other factors further supported the choice of the Raspberry Pi. The environment highly encourages deploying code to the device from a shared code base rather than working just with students' own code on their local machine until the last phase and then integrating. Specifically, each group of students was assigned only two Raspberry Pis. Thus, while code can be transferred to the Raspberry Pis directly, since each team member needed to frequently move their code to the device to test their code (they did not have the sensors on their own machine) the result was teams constantly integrating their code and moving the integrated code base to the device. Given that two of the other focuses of our design patterns class are working in a team environment with a shared code base and the importance of creating automated regression tests, this situation was ideal for having students discover on their own the importance of using best practices in this area.

Finally, the Raspberry Pis were an inexpensive solution, with each team of five or six students being given a kit that cost just under $275. For the duration of the semester, each student team received:

- Two Raspberry Pi kits, which included Raspberry Pi 3 model B with WiFi and Bluetooth connectivity, a USB power supply, a 16GB Micro SD card with Raspbian OS, an enclosure case, two heatsinks, and an HDMI cable;
- 12 Grove components, which included a sound sensor, a temperature and humidity sensor, a light sensor, a relay, a button, an ultra-sonic sensor, a rotary angle sensor, an LCD RGB Backlight, a buzzer, and a red, blue, and green LED;
- Two GrovePi boards, which allow Grove sensors to be attached directly to the Raspberry Pi; and

- One Raspberry Pi Camera Module

V. COURSE FORMAT

At the beginning of the course, students were told that they would be asked to come up with a semester long project that would have "sprint" checkpoints throughout the semester. The limits of the specific requirements for the project were:

"For the course long project you have the freedom to create whatever you want. The project is not based on building any specific application or use of components (sensors, output components, camera), but rather the primary source of evaluation will be based on the proper use of design patterns and testing to accomplish your goals. The key aspect of the project is figuring out the type of additions to your project that would benefit from the use of specific patterns we will cover. Simply coding a pattern correctly, but not using it in a place that is appropriate will not receive full marks." To further encourage creative solutions, teams were informed that guests from local industry, from a variety of technical and non-technical backgrounds, would vote on which project was most creative. The winning team would then be rewarded extra points towards their final grade.

Within the first four weeks, students were required to self-select a group of five to six students and submit a general idea of their project vision. This vision was not required to identify any specific components that would be used, but rather was aimed at them identifying their general direction (e.g. a robot that does xyz or an alarm that can be activated and turned off through multiple different environmental sensors). The purpose was twofold: First, make sure the teams gave some forethought to areas that needed to be considered in their designs throughout the semester. Second, to give the instructor a chance to provide feedback if the scope was appropriate.

One class was dedicated to Git workflow and its use in a team environment, and a short tutorial was given in how to connect to the Raspberry Pis via VNC remote desktop. No other class time was dedicated to either the Raspberry Pi environment or the components. All other lectures focused on teaching the

[1]http://pi4j.com/

design pattern content of the textbook Gamma et al. [16] and creative thinking toward software design problems.

A. Lecture format

Lectures would begin by introducing the new pattern(s) in the context of instructor provided examples and resulting code. Then in the same class period rather than a more traditional approach of presenting a problem and having the students figure out how the new pattern could be applied to solve it, the model was flipped to have students create a concrete problem where the pattern applied. Student groups had ten minutes to go to the white board to come up with their own example of the pattern given loose constraints such as:

- Create an example where [current pattern] makes sense within the context of your choice of board game; or
- Your team has been asked to represent some aspect of a car. Create a scenario where it would make sense to use [current pattern] and draw the design; or
- Create a scenario where what you might want to use [current pattern] and one or more other patterns we have covered *that cannot include any previous example we have discussed.*

Throughout this creation period, students were encouraged to brainstorm their ideas aloud among their group as they came up with the Unified Modeling Language (UML) to present on the board. During their discussions, the instructor would walk among the different groups and help clarify their understanding of the pattern or suggest tweaks to their scenario or design that better exemplified the pattern's use.

Once the groups had drawn their designs, the class would reconvene and each group would explain the scenario they came up with, presenting how their UML solved that problem. The instructor would then present them with additional theoretical requirements outside of their original design along the scenario they had created. In front of the class, the group would then have to show and explain how their design would have to be modified in order to support the additional functionality. Thus, the class' creativity drove the examples they were learning from, and groups would learn from each other how best to articulate when each pattern was needed and its benefits in future flexibility.

B. Course project

Using GitHub Classroom, a Git repository was created for each group that served as their evolving code base for the length of the semester. The deliverables for the project were broken down into four two-week sprints for the final 8 weeks of the course. The requirements for each sprint was that the code base must include approximately two new design patterns that were covered at any point previously in the course. Teams were encouraged to explore including any component, technology, or third party library that helped them achieve their vision of the project, but it was up to them to determine what type of addition would best demonstrate where one of these particular design patterns made sense.

The design and implementation of each checkpoint would be done on a "development" branch off of the "master" project branch. When a checkpoint was due, all deliverables (UML, implementation, and automated test cases) would be submitted as pull requests to the master branch. The feedback features of GitHub's pull request would then be used to allow the instructor to provide overall feedback on the deliverables as well as comments on individual lines of code before the pull request was merged with the master branch. For the next checkpoint, groups added their additional functionality as well as their revisions and/or corrections based on feedback from the previous checkpoint. For each interim checkpoint, it was made clear that grading of the checkpoint would be based on the correctness of the application of the design pattern(s) and its implementation within the context of the application, but the overall functionality of the project would not be evaluated until the final project was presented. The reasoning behind this approach was to encourage students to explore including new functionality that could have applications to their end vision, without needing to worry about it not adding visible benefits initially. This also had the practical benefit of making the checkpoints easier to grade without requiring the instructor to reconfigure each project's unique set of hardware component connections to the Raspberry Pi to evaluate the overall project.

VI. Example Student Projects

Below we present a sample of the variety of projects that resulted from encouraging students to embrace their creativity, and evidence of their grasp of how the course content could be applied in a variety of real contexts. The patterns referred to below are based on the pattern names in [16].

A. Emotional Robot

The students' stated goal was to create a robot that could react to an image and display how it was "feeling." This team sought to use design patterns to create a modular system that would allow adding a variety of new emotions, personality types, and responses easily or even dynamically creating new ones.

The basic structure of their project was to use a camera attached to the Raspberry Pi to perceive the environment. Based on a variety of configuration options, the robot would then display its emotional reaction to the image through connected variable color LEDs as well as, through Twitter integration. Thus, resulting in not only does the Raspberry Pi create a visual display of the emotion, but also a Tweet of a textual description of its emotion. The details of their implementation demonstrated an excellent understanding of the benefits of the patterns they included, when they should be used, and how to implement them in a number of complex integration scenarios. An example of this was how they had incorporated a combination of Factory and Strategy patterns applied in a variety of contexts to allow the user to select at runtime how the robot should interpret images. Specifically, whether the robot should use average color of the image as the primary new input affecting emotion, or use integration with the Google Cloud Vision API to react to what was identified in the picture. For example, when a picture was held in front of the camera, one strategy might interpret the image as on average "blue" and make the robot experience "crippling depression." While another strategy might identify the same image as likely containing the Eiffel Tower, the sun, and a face expressing joy.

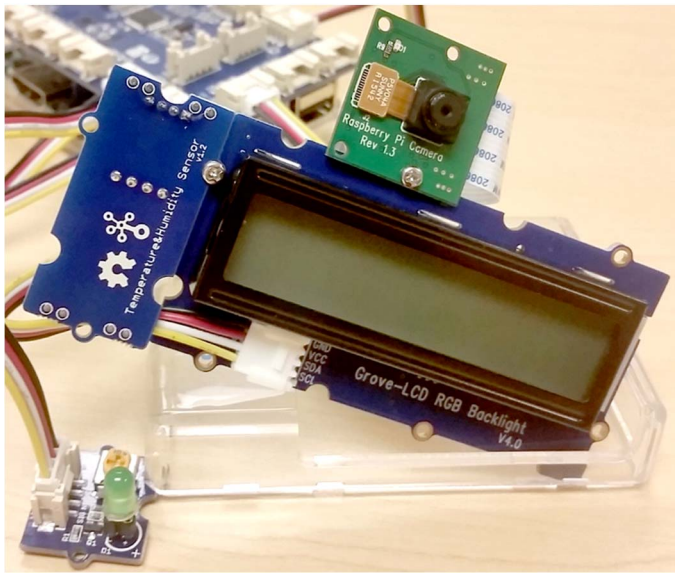
This alternative interpretation of the same image might make the robot experience "joy" and "bravery".

Other aspects that demonstrated an understanding of pattern applicability was the use of the Composite and Factory patterns to allow the robot's mood to be composed of a variety of current and prior emotions, and based on whether the user specified the robot was pessimistic or optimistic, the same labels perceived in an image might produce "anxiety" or "bravery." This composition of emotions that made up the current mood would then be used with the Builder pattern to construct a textual representation of combinations of the emotions and the overall intensity. This would culminate in a LED adjustment and a Tweet along the lines of "I see a [person] and it makes me feel [quite] [comfortable, happy, and brave]."

B. Time-lapse environment sensing camera

For this project, the students incorporated the camera, a LED, and the temperature and humidity sensor with a mix of command line operations and external libraries in other programming languages. The basic functionality of the project was the user could setup the Raspberry Pi to record a series of pictures over an extended period of time, which would be combined to make a time-lapse animated GIF that would be posted to Twitter. The project also incorporated the temperature fluctuations over the course of the capture period to adjust the relative hue of the individual images to reflect it becoming warmer or colder.

In developing this project, they used a variety of patterns to wrap integration with command line calls, and applied the patterns to both Python and Java code to ease integration of open source packages in a more flexible way. Another example where their creativity led to complex problem solving was a solution where they reverse engineered how the electrical signals were used in sample code in other languages to create their own customized hardware interaction. Driven by their end functional goal, they created a Facade pattern to simplify interaction with the sensors, hiding the significant complexity of their Grove hardware interaction. Their project correctly integrated numerous patterns that spanned programming languages and seamlessly brought everything together in a flexible, maintainable design.



C. Alarm clock

Still a different take on the requirements was a sophisticated alarm clock that could alert the user in a variety of ways. These alerts were completely runtime configurable and allowed the user to pick from a variety of flashing patterns for the LEDs and the buzzer could play simple MIDI files. They also used an Adapter pattern to wrap a Python LCD library so that the user could configure the alarm to flash messages as it went off.

D. Motion alarm system

Another group opted to construct a motion activated burglar alarm system. Their project consisted of an alarm that could be set to respond in a variety of ways. Their project incorporated an ultrasonic distance sensor, a button, LEDs, and a buzzer. Within their project, they had used the State pattern to have the functionality of the distance sensor, button, and keyboard input change depending on whether the alarm was armed or disarmed. They demonstrated an understanding of the Strategy pattern in allowing the person arming their device to choose between a constant buzz and illuminated LED, a pulsing buzz and light, or a silent alarm where events were instead logged with no visible cues.

TABLE I. DESIGN PATTERN USE BY STUDENT PROJECT

	Chain of Responsibility	Template	Strategy	Singleton	Iterator	Inner classes	Adapter	Composite	Decorator	Abstract Factory	Factory Method	Prototype Factory	Builder	Visitor	Facade	Flyweight	State	Observer	Command	**Total**
Emotional robot			1		1	1	1	1	1	1			1							**8**
Time lapse		1	1	1	1	1	1	1	1	1		1	1		1			1		**13**
Alarm clock			1	1	1	1	1	1	1	1	1		1							**10**
Alarm system			1	1		1	1	1	1	1			1				1			**9**
TOTAL	**0**	**1**	**4**	**3**	**3**	**4**	**4**	**4**	**4**	**4**	**1**	**1**	**4**	**0**	**1**	**1**	**1**	**0**	**0**	**40**

E. Summary of pattern use

Table 1 shows a summary of Gamma et al. [16] pattern use by project. The patterns shown across the top are ordered chronologically by when they were taught in class. As these results show, a wide variety of patterns were included, but perhaps not surprisingly the latter patterns were not included as frequently as they were largely being taught during the final sprint. It should be noted that "inner classes" is not a one of the patterns in [16], but correct use of how to incorporate it in design was covered and it was counted in the requirement to have eight patterns so it has been shown here to explain how projects addressed that requirement.

VII. LESSONS LEARNED

A. Using creativity to direct learning

Throughout the course, student creativity drove how students thought about the course material and its application. Overall, the students had very positive things to say about this learning style:

- *"It allowed us to take the concepts we were learning in class and apply them in a way that felt relevant to how we might actually use them in the real world."*
- *"This project helped me think more about the design of the whole software that was being created. This was the first time where I was not thinking like an average programmer who is given requirements and just codes. This project helped me get experience in software architecture and how to think of the future as well versus get the project done and ship it out...The project was solidifying the concepts in my brain and removed all questions/doubts I had when the lectures were given."*

The lecture style required students to be able to direct this creativity towards specific goals in short periods of time (10 minute chunks). Initially students found this very challenging, but became much better at this as the semester progressed. In future iterations of this course some kick off activities at the beginning of the semester to stimulate this type of thinking such as Lego Serious Play, which has been shown to improve creative critical thinking, may help in making initial student white board sessions easier [24].

B. Team dynamics

Perhaps one of the biggest impacts from shifting from a more concrete set of requirements to student creativity being the only limitation on projects was the impact of team composition and team dynamics. For example, since the students selected their own teams, while some organized around specific creative interests, others organized by familiarity. Consequently, the teams that organized by familiarity tended to be classmates who were at similar places in their advancement through the major.

For the more senior team this worked to their advantage as they had a strong breadth of knowledge and several of the members had worked in professional team development environments. While the group had some difficulty initially agreeing on a vision, once identified, their experience gave them a wider range of options in terms of each individual developer being able to incorporate their own creative take on the project resulting in the time-lapse environmental camera.

By comparison, the less senior team worked well together in terms of being happy with the contribution of each member and division of work, but as a group, they initially struggled primarily in mindset. From talking with the team, one of their biggest challenges was shifting from being given a rigorous set of requirements (as was more common in introductory courses) to the complete freedom to define their own direction driven by learning goals instead. Once they grew into the idea of identifying solutions for their creative ideas, however, they were quite successful with the enhanced alarm clock project.

The teams that organized by interest had almost the opposite challenge. Identifying the general vision and possible enhancements was easy, but working as a team proved the biggest challenge. When it came to the details, there were disagreements as to the priority of tasks and how tasks should be divided among the group. For one of these groups a natural leader emerged and the joint interest led to some really creative ideas as seen in the emotional robot. With the team where this did not occur, the project's implementation of concepts was good, but they indicated without a well-defined set of requirements there was constant strife related to the division of work (motion alarm system).

Below are examples of direct feedback from students in course evaluations:

- *"Having an open goal forced our group to focus on specific goals, rather than tackling everything at once, and to work using an iterative approach."*
- *"Sometimes the group had conflicting ideas of what needed to be done to meet the requirements we laid out for ourselves because there was no one singular vision with a specific set of product requirements."*
- *"Because it was so open-ended, this project required a lot more face-to-face meeting time to check on the status of the project and clear up any confusion about end goals."*
- *"The open-ended type of project requires a lot more effort from the participants to really work as a team. You have to make sure everyone is on the same page and need a team leader (or leaders) who can organize meetings and has a strong grasp of what the finished product will look like."*

C. Use of Raspberry Pi platform

Real-world hardware. Hardware can sometimes break, wired connections may fail (especially if the entire Raspberry Pi plus sensors assembly is handled without a special care), sensors are not always precise, and the sensors may behave differently depending on the external factors such as network infrastructure and physical environment. These issues can add a lot of frustration and difficulty to debugging the code. As a result, students eventually learn to resolve these situations, which brings them closer to the real-world experiences that are much different compared to many traditional classroom projects.

- *"Raspberry Pis work well for introducing students to software/hardware interaction. Forces students to collaborate."*
- *"Testing with different components with RPi was very tedious."*
- *"I like the use of the Pi as it allows more unique projects to be created."*

VIII. Discussion of Learning Outcomes

While creative projects and maintaining student interest are a positive outcome, just as relevant is the impact on student learning. Having this course taught two previous times by the same instructor with a more traditional software based project presented a good opportunity to assess the learning impact of the changes discussed in this work. As in previous years, the oral exam during the final project presentations, and the final exam maintained a similar style of questions that addressed the following areas: fundamental concepts; given UML, identifying appropriate patterns and refactoring the UML; and identification of appropriate design patterns given a prose scenario and being able to explain the benefits of a pattern's use in that scenario.

Based on these, there was little change in understanding of fundamental concepts such as abstraction, encapsulation, modularity, and polymorphism. However, there was clear improvement in students' ability to identify appropriate patterns given a prose description, and a drastic improvement in students' ability to articulate the benefits of using that particular pattern in the given scenario. The only place where there was a slight decrease in student performance was in being able to recognize and refactor patterns given UML.

Intuitively, the lack of change in understanding of fundamentals makes sense, as there would be little change in the use of these concepts whether given a software project or creating a project of their own design. In terms of the areas of improvement, these seem to reflect the benefits of students needing to identify what type of scenario is needed on their own required students to develop a deeper understanding of the applicability of the patterns. Then once included, they directly observed the benefits of specific patterns in what additional flexibility they provided as they continued to extend their projects as their goals and requirements changed throughout the semester. This understanding was demonstrated in both their ability to recognize the applicable pattern given a description of a different problem, and in excelling at being able to explain the benefits of a pattern as they had experienced them first hand. Regarding the small decrease in recognizing and refactoring patterns from UML, in retrospect this is likely due to the way the material was covered in class and the structure of the project being primarily driven by students creating their own examples and less on identifying weaknesses/opportunities in existing designs. Thus, they were presented with more scenarios of recognizing the applicable patterns that should be applied during the design phase rather than analyzing **and revising** a design created by somebody else. One possible way to address this in future iterations of this course might be to have some (poorly) pre-coded functionality that the groups must use and then when they are well into the project development, have one of the checkpoints include revising and extracting patterns out of that given code.

IX. Summary

Overall, the creativity focused learning approach was well received by the students. Many indicated they learned far more on the project than what was directly relevant to the course material as a result of their own exploration. The common negative feedback was this resulted in the project taking much longer than many other projects because the learning was very non-linear. However, these same students also frequently indicated in the end it was a very rewarding experience.

- *"The project was big but the best and most professional project I have ever done, can brag about it with potential employers!"*

In addition, compared to previous iterations of this course based on a traditional lecture style and a software only based project, learning outcomes were improved in some key areas. Specifically, student answers during verbal and written assessments reflected much better understanding of when different patterns should be used and the practical benefits of their use.

Acknowledgments

Providing each student in our course with a Raspberry Pi kit was made possible by a CCSU AAUP Curriculum Development grant.

References

[1] Alphonce, C., and Ventura, P. 2002. "Object orientation in CS1-CS2 by design." *ACM SIGCSE Bulletin* 34, 3: 70-74.

[2] Amoussou, G.-A., Cashman, E., Steinberg, S. 2007. Ways to learn and teach creativity and design in computing science. In *Proceedings of the 2007 Symposium on Science of Design* (SoD '07). ACM, New York, NY, USA, 12-13. DOI=http://dx.doi.org/10.1145/1496630.1496637

[3] Astrachan, O., Mitchener, G., Berry, G., & Cox, L. 1998. "Design patterns: an essential component of CS curricula." *ACM SIGCSE Bulletin*. Vol. 30. No. 1. ACM.

[4] Boden, M.A., 2009. Computer models of creativity. *AI Magazine*, *30*(3), p.23.

[5] Cennamo, K., Douglas, S.A., Vernon, M., Brandt, C., Scott, B., Reimer, Y., and McGrath, M. 2011. Promoting creativity in the computer science design studio. In *Proceedings of the 42nd ACM technical symposium on Computer science education* (SIGCSE '11). ACM, New York, NY, USA, 649-654. DOI=http://dx.doi.org/10.1145/1953163.1953344

[6] Chatzigeorgiou, A., Tsantalis, N. and Deligiannis, I. 2008. "An empirical study on students' ability to comprehend design patterns." *Computers & Education* 51.3: 1007-1016.

[7] Cinnéide, M. Ó., and Tynan, R. 2004. "A problem-based approach to teaching design patterns." *ACM SIGCSE Bulletin*. Vol. 36. No. 4. ACM.

[8] The College Board, 2014. AP Computer Science Principles. Curriculum Framework 2016-2017. https://secure-media.collegeboard.org/digitalServices/pdf/ap/ap-computer-science-principles-curriculum-framework.pdf

[9] Crawford, R.P., 1968. *Direct Creativity: With Attribute Listing*. Fraser Publishing Company.

[10] De Bono, E., 1968. *New think: The use of lateral thinking in the generation of new ideas*. Avon Books.

[11] Doboli, S., Tang, W., Ramnath, R., Impagliazzo, J., VanEpps, T., Agarwal, A., Romero, R. and Currie, E.H., 2010, October. Panel—Models of entrepreneurship education and its role in increasing creativity, innovation and leadership in computer science and engineering students.

In *Frontiers in Education Conference (FIE), 2010 IEEE* (pp. F1B 1-4). IEEE.

[12] Felder, R.M., 1988. Creativity in engineering education. *Chemical Engineering Education*, *22*(3), pp.120-125.

[13] Ferreira, D.J., 2013. Fostering the creative development of computer science students in programming and interaction design. *Procedia Computer Science*, *18*, pp.1446-1455.

[14] Forster, F. and Brocco, M., 2008, January. Understanding creativity-technique based problem solving processes. In *Knowledge-Based Intelligent Information and Engineering Systems* (pp. 806-813). Springer Berlin Heidelberg.

[15] Fuggetta, A. 1999. Rethinking the modes of software engineering research, *Journal of Systems and Software*, vol. 47, pp. 133-138, 1999.

[16] Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley.

[17] Gates, A.Q., 2010. Discovery, Innovation, and Creativity: The Core of Computing. *Computer*, *43*(2), pp.98-100.

[18] Gestwicki, P., and Sun, F.. 2008. "Teaching design patterns through computer game development." *Journal on Educational Resources in Computing (JERIC)* 8.1: 2.

[19] Gómez-Martín, M., Jiménez-Díaz, G., and Arroyo, J. 2009. "Teaching design patterns using a family of games." *ACM SIGCSE Bulletin*. Vol. 41. No. 3. ACM.

[20] Greenberg, S., 2009. Embedding a design studio course in a conventional computer science program. In *Creativity and HCI: From Experience to Design in Education* (pp. 23-41). Springer US.

[21] Guilford, J.P., 1977. Way beyond the IQ: Guide to improving intelligence and creativity. *Buffalo, NY: Creative Education Foundation*.

[22] Guzdial. M. 2003. A media computation course for non-majors. In *Proceedings of the 8th annual conference on Innovation and technology in computer science education* (ITiCSE '03). ACM, New York, NY, USA, 104-108. DOI=http://dx.doi.org/10.1145/961511.961542

[23] Harel, I., and Papert, S. 1990. "Software design as a learning environment." *Interactive learning environments* 1.1: 1-32.

[24] James, A. 2013. "Lego Serious Play: a three dimensional approach to learning development." *Journal for Learning Development in Higher Education*, No. 6.

[25] Jaramillo, C.M.Z. and Alvarez, M.C.G. 2014. Incorporating Playful Activities in the Software Engineering Teaching. *Developments in Business Simulation and Experiential Learning*, vol. *41,* pp. 248-255.

[26] Lewis, C., Jackson, M.H. and Waite, W. M. 2010. Student and faculty attitudes and beliefs about computer science. *Commun. ACM* 53, 5 (May 2010), 78-85. DOI=http://dx.doi.org/10.1145/1735223.1735244

[27] Michalko, M., 2010. *Thinkertoys: A handbook of creative-thinking techniques*. Ten Speed Press.

[28] Mishra, P. and Yadav, A., 2013. Rethinking technology & creativity in the 21st century. *TechTrends*, *57*(3), p.10-14.

[29] Nguyen, L. and Shanks, G., 2009. A framework for understanding creativity in requirements engineering. *Information and software technology*, *51*(3), pp.655-662.

[30] Osborn, A.F. 1953. Applied Imagination: Principles and Procedures of Creative Problem Solving, New York: Charles Scribner's Sons.

[31] Papavlasopoulou, S. 2016. "Engaging students with Computer Science through creativity: Toward better understanding and improved methods." *Advanced Learning Technologies (ICALT)*, 2016 IEEE 16th International Conference on. IEEE.

[32] Pillay, N. 2010. "Teaching design patterns." *Proceedings of the SACLA conference*. Pretoria, South Africa.

[33] Porter, L. and Simon, B. 2013. Fostering creativity in CS1 by hosting a computer science art show. *ACM Inroads* 4, 1 (March 2013), 29-31. DOI=http://dx.doi.org/10.1145/2432596.2432609

[34] Reiser, S.L, and Bruce, R.F. 2009. Fabrication: a tangible link between computer science and creativity. In *Proceedings of the 40th ACM technical symposium on Computer science education* (SIGCSE '09). ACM, New York, NY, USA, 382-386. DOI=http://dx.doi.org/10.1145/1508865.1509001

[35] Rich, L., Perry, H., and Guzdial, M. 2004. "A CS1 course designed to address interests of women." *ACM SIGCSE Bulletin*. Vol. 36. No. 1. ACM.

[36] Robertson, J., 2005. Requirements analysts must also be inventors. *Software, IEEE*, *22*(1), pp.48-50.

[37] Robins, A., Rountree, J., and Rountree, N. 2003. "Learning and teaching programming: A review and discussion." *Computer science education* 13,2: 137-172.

[38] Shell, D., Hazley, M., Soh, L., Miller, L., Chiriacescu, V., and Ingraham, E. 2014. "Improving learning of computational thinking using computational creativity exercises in a college CSI computer science course for engineers." *Frontiers in Education Conference (FIE)*, 2014 IEEE.

[39] Schmidt, Douglas C. 1995. "Using design patterns to develop reusable object-oriented communication software." *Communications of the ACM* 38, 10 (1995): 65-74.

[40] Vieira, E.R., Alves, C. and Duboc, L., 2012. Creativity Patterns Guide: Support for the application of creativity techniques in Requirements Engineering. In *Human-Centered Software Engineering* (pp. 283-290). Springer Berlin Heidelberg.

[41] Wick, M. 2005. "Teaching design patterns in CS1: a closed laboratory sequence based on the game of life." *ACM SIGCSE Bulletin*. Vol. 37. No. 1. ACM.

[42] Wolfgang, P. 1994. *Design patterns for object-oriented software development*, Addison-Wesley.

[43] Yang, F. and Zhang, W. 2012. Exploration and practice of constructing creative engineering laboratory on software development. In *Computer Science & Education (ICCSE), 2012 7th International Conference on* (pp. 1597-1601). IEEE.