Creative Assessment Design on a Master of Science Degree in Professional Software Development

Cathryn Peoples
Ulster University
United Kingdom
c.peoples@ulster.ac.uk

ABSTRACT

A Master of Science (MSc) conversion degree is one which retrains students in a new subject area within a fast-tracked period of time. This type of programme opens new opportunities to students beyond those gained through their originally-chosen degree. Students entering a conversion degree do so, in a number of cases, to improve career options, which might mean moving from an initially-chosen path to gain skills in a field that they now consider to be more attractive. With a core goal of improving future employability prospects, specific requirements are therefore placed on the learning outcomes achieved from the course content and delivery. In this paper, the learning outcomes are focused on the transferable skills intended to be gained as a result of the assessment design, disseminated to a cohort of students on a Master of Science (MSc) degree in Professional Software Development at Ulster University, United Kingdom. The coursework submissions are explored to demonstrate how module learning has been applied, in a creative way, to facilitate the assessment requirements.

Keywords

Conversion degree, Java, Master of Science (MSc).

1. INTRODUCTION

A Master of Science (MSc) conversion degree is one which retrains students in a new subject area within a fast-tracked period of time. A subject which would be taught within a three-year period in an undergraduate degree is instead taught during one intensive year. This type of programme opens new opportunities to students beyond those gained through their originally chosen degree. For a number, it is critical that what is learnt during the programme improves their employability potential in a field which is new to them. Students entering a conversion degree do so, in a number of cases, to improve career options, which might mean moving from an initially-chosen path to gain skills in a field that they now consider to be more attractive. The conversion degree can help them to gain the required knowledge and skillsets to do so.

With a core goal of improving future employability prospects, specific requirements are therefore placed on the learning outcomes achieved from the course content and delivery. In this paper, the learning outcomes are focused on the transferable skills intended to be gained as a result of the assessment design. Assessments presented in this paper were disseminated to a cohort of students on a Master of Science degree in Professional Software Development at Ulster University, United Kingdom. This is a conversion degree into Information Technology for students from non-IT backgrounds.

To understand the specific reasons that students had become part of the degree programme, and to avoid assuming that it was to improve their employability options, a survey was disseminated at the beginning of the academic year. This was done with the core objective of tailoring the teaching approach to meet their needs. When asked about their reasons for completing the degree, the majority of responses were focused around the fact that students were studying software development with the goal of employment in this field (Table 1). Going deeper into the reasons that students wanted to work in this field, they acknowledged it was due to their passion for technology, and because they identified the IT industry as one with a more certain chance of employment than others. When designing the teaching approach, it was therefore considered to be important to support students moving into this industry and to improve their prospect of doing so.

Transferable skill development was encouraged through the design of the module assessment, with students being assessed on their ability to apply knowledge gained during the teaching period while inherently developing transferable skills in doing so. To achieve this, assessments were shaped around the state-of-the-art in technology. The objective was to select news stories which are reported internationally and which would hopefully appeal to student interest, an approach in line with the belief that, *"… the concept of 'student engagement' is predicated on the belief that learning improves when students are inquisitive, interested or inspired, …*" [1]. Furthermore, "*When a topic connects to what students like to do, engagement deepens as they willingly spend time thinking*" [2]. In line with the objective of inherently gaining transferable skills, it was hoped that selecting popular news stories would give students an opportunity to develop their ability to discuss technical concepts comfortably, to become critical in the selection and application of their knowledge to solve real-life problems, and to appreciate the context of their learning in relation to the wider field.

It is acknowledged in [4] that soft skills such as those described above are particularly important, in recognition of the fact that, *"In an increasingly global, technological economy, they say, it isn't enough to be academically strong. Young people must also be able to work comfortably with people from other cultures, solve problems creatively, write and speak well, think in a multidisciplinary way, and evaluate information critically*" [4].

DOI: https://doi.org/10.22369/issn.2153-4136/12/2/11

Table 1. Student reasons for completing the degree.

Why are you studying for this degree?
"*Interested in computers, hope will lead to employment*"
"*Always wanted to learn coding, love technology, lack of programming knowledge held me back*"
"*To get a job in software development*"
"*Want career as a programmer or developer, as software development has good prospects and I enjoy logic and following patterns*"
"*Interested in programming language, could use skills from masters and linguistic knowledge to develop voice-activated software*"
"*To broaden skills for strong professional career in future*"
"*Interested in computers, but did PE teaching, employment limited*"

It is therefore in an attempt to bridge the gap between gaining the necessary knowledge and making the students employable that the assessments presented in this paper have been designed. The objective of this paper is to present a selection of the assessments which were designed to support student employability after completing the MSc conversion degree.

The remainder of this paper continues as follows. In Section 2, a literature review is presented, which considers how assessments can be designed to maximize student engagement with them, how to apply creativity in assessment design, and how to mark creative assessments consistently. This is followed in Section 3 with presentation of the creative coursework specifications which are the focus of this paper, together with a selection of the solutions in Section 4 to demonstrate how students harnessed their programming skills to fulfil the assessment requirements. Finally, the paper concludes in Section 5.

2. LITERATURE REVIEW

The assessments which are presented in this paper are used to examine a student's grasp of the entire module content and to apply their knowledge to a software development problem in a creative way; they are therefore summative assessments. As defined by the Council for the Curriculum, Examinations and Assessment [5]:

- "*Summative assessment usually takes place after pupils have completed units of work or modules at the end of each term and/or year.*
- *The information it gives indicates progress and achievement usually in grade-related or numerical terms.*
- *It's the more formal summing-up of a pupil's progress.*
- *The information can then be … used for certification as part of a formal examination course*."

Summative assessment is important due to the role it plays in determining a student's understanding of the module content at the end of the teaching period. It is therefore using this teaching material that it can be determined if module learning objectives have been achieved on a per student basis. There are high stakes associated with summative assessments, as it is not possible to revise submissions or receive feedback for improving future work, as is the case with the alternative assessment type, formative. It is therefore important that summative assessments are designed in a way which will maximize the opportunity that students can perform to the best of their ability.

King and English (2015) report that students respond most effectively to assessments which use real world scenarios, with problems contextualized in a way which students can understand [7]. This concurs with an opinion of Kearney and Perkins (2014), in that, real-world problems "*better engage them in their coursework and better prepare them for the world outside the classroom*" as opposed to "*research projects that do not have significance outside of the classroom*" [26]. This, essentially, describes the concept of authentic assessment. "*Authentic assessment is based on students' abilities to perform meaningful tasks they may have to do in the 'real world.' In other words, this form of assessment determines students' learning in a manner that goes beyond multiple choice tests and quizzes*" [27]. These findings therefore validate the effectiveness of the design approach applied to the assessments presented in this paper.

The conclusions reached by King and English (2015) are based on a study for which students were "recruited" as Optical Engineers and asked to build an optical instrument which could be used to spy on people. They concluded from this study that the assessment was appropriate for engineering students, given that it enabled ability to structure the stages of design, construction, and redesign in the development. This was in support of the fact that, "*meaningful STEM-integration is possible when students have the prior knowledge to apply to a well-structured engineering design task*" [7]. It is agreed in this paper that problem-solving ability is more likely when students are providing solutions to problems which they can contextualize, through either viewing them and/or having first-hand experience of the problems involved. It is with this understanding that the assessments presented in this paper have been designed. While it may be unlikely that students can have first-hand experience of the assessment problems presented in this paper, each domain in the assessment was chosen for the reason that the software might be one which they use in their day-to-day lives, e.g. Facebook, or because it is one highly likely to be of interest to anyone involved in technology, e.g. the emotion engine, Pepper the robot. Students were asked to use their technical knowledge to create similar systems, a form of situated cognition which helps them to recognize the placement of their abilities within the wider field and to understand the ways in which popular technologies are created in reality. This was done in recognition of, "*the need to draw explicit connections among topics for retention of learning*" [8]. Furthermore, for students who are new to the IT field, given the requirement for entry onto the degree programme that students have no prior IT education, it was hoped that using state-of-the–art technologies would help them to, "*keep pace with the rapid change and recent development in this era of globalization, …*" [9].

The approach of selecting technologies which students use in their day-to-day lives as the focus of each assessment was an action taken to "*facilitate creativity in which learners are motivated to discover things by themselves*" [9]. This was based on the fact that, "*Intrinsic motivators include fascination with the subject, a sense of its relevance to life and the world, a sense of accomplishment in mastering it, and a sense of calling to it*" [24].

The assessment specifications were presented in detail, and there was limited flexibility in what should be achieved. This is in spite of the fact that, "*There have been calls in the literature for changes to assessment practices in higher education, to increase flexibility and give learners more control over the assessment process*" [10]. Students were not restricted, however, in how they could achieve it, with marks awarded for the creative ways in which their technical knowledge was harnessed. As, "*Research has shown that creativity leads to intellectual development and brain growth, when*

creativity is nurtured …" [9], it was an objective to ensure that students focused their research efforts on how they achieved the solution, as opposed to what they were providing a solution to.

"*Creativity is the application of knowledge and skills in new ways to achieve a goal*" [11]. In having a creatively-designed assessment, it was hoped that this would encourage innovation in the solution presented by the student. This is in line with Kampylis and Berki (2014), who state that, "*Creative thinking is defined as the thinking that enables students to apply their imagination to generating ideas, questions and hypotheses, experimenting with alternatives* …" [12]. Asking students to build a replica system of one they are familiar with using programming techniques gained during the module teaching is one way to allow their creativity to be demonstrated. It is recognized, however, that, "*certain approaches to education may possibly foster greater creativity than others*" [11]. This statement is made specifically in relation to children of school age, with Montessori education using self-directed creativity and collaborative play [14], and Reggio Emilia education focusing on collaboration between children learning from their environment [15]. It is believed that the assessment presented in this paper has similarities to the Reggio Emilia approach, given that the assessment is based on a technology from the wider environment in which students operate. Furthermore, two of the principles of Montessori are to understand the systems of which the world exists and to support the imagination. Again, by basing the developments on software systems which are available from the wider environment, and by asking students to apply their knowledge from the modules which they have been taught, it is believed that both principles are met, helping to verify the suitability of the assessment design.

However, "*There is a lot of risk aversion in relation to assessment design. Staff fear being too creative in case their assessment is too challenging* …" [13]. Furthermore, it is recognized that marking creative work is challenging, given the desire to mark it quickly and the need to mark it consistently. Jackson (2005) of the Higher Education Academy notes that, "*Of all the aspects of creativity the one that poses the greatest challenge to teachers is how to assess/evaluate it*," identifying that some teachers just do not know how to assess such work [16]. This is significant, with the author going on to explain that, "*evaluation is critical to the very idea of creativity*" [16].

"*Students as well as academic staff … often ask the question as to how one marks creative writing. Indeed, they often wonder if it is even possible? Surely, they say, this is a subjective response, a matter of taste?*" [17]. Brookhart (2013), however, proclaims that, "*We can assess creativity …", and demonstrates how with a "Rubric for Creativity*" [18]. This is essentially based on evaluating work according to 1) how the ideas are combined together, with the highest levels of creativity being demonstrated when the, "*Ideas are combined in original and surprising ways* …", and 2) what is communicated, with creativity indicated with "*an original contribution that includes identifying a previously unknown problem, issue, or purpose*" [18]. Undoubtedly, there will likely always be some element of subjectivity when assessing creative work; however, a rubric provides the basis for a standardized approach to achieving this.

Computer software is one approach to assessment which can be electronically and automatically marked [19]. Automated marking of software programmes, however, is at odds with the concept of a creativity focus presented in this paper: Acknowledged by Brookhart, "… *with a broad concept as creativity, there's no single formula that will always work*" [18]. While Hill and Turner (2014) write about "*Code Originality*" [19], this is concerned about similarity between student work as opposed to an original design through its creativity. In [20], an automated system is proposed to assess software programs. This essentially tests its ability to compile, given the entry of input values chosen by the instructor. Therefore, to assess the creativity of a software program, it is unlikely to be possible to exploit automated marking, where, using Brookhart's rubric [18], the creativity is assessed based on the way in which the knowledge is put together.

It is believed that the assessments presented in this paper follow a transformative approach to learning. According to [21], transformative learning is described as occurring in situations where, "… *opportunities [are created] for critical thinking through providing content that introduces new ideas."* It was the objective that this opportunity was presented to students using state-of-the-art technologies which students were required to mimic in their software solutions. "*Transformation then happens in a community as students bounce ideas off one another*" [21]. It was the intention that this would be possible given that all students were set the same task. As part of transformative learning, it is also necessary for the instructor to, "*provide the opportunity for students to act on their new found beliefs*" [21]; it was hoped that this would be achieved through the overall assessment selection.

Practical programming solutions, such as the output required for the assessments presented in this paper, need to be designed in such a way that the software meets a specific target and achieves a certain goal. In addition to this practical level of functionality, submissions are also assessed according to how the module knowledge has been used. This can be contrasted with a research-based task, on the other hand, for which there can be an open and variable outcome, the case for which simply needs to be argued. Problem solving skills help students to work out how to reach the end goal, with critical thinking helping them to select the relevant elements from their learning and creative ability to apply them in a meaningful way. These are important qualities in support of employability: "*Merely having knowledge or information is not enough. To be effective in the workplace …, students must be able to solve problems to make effective decisions; they must be able to think critically*" [25].

The assessments which are presented in this paper have been designed in a manner which would support the International Baccalaureate (IB) Learner profile [6]. IB education is an international education programme delivered to students in school who are aged between three and nineteen years old, which is considered by some to be, "very well-respected by universities" [23]. One objective of the IB programme is to develop student skillsets such that they are internationally minded. It was hoped that this would be achieved in these assessments through the focus on international news stories in the field of technology, with recognition that technical capabilities vary widely across the world. Another IB programme objective is to develop thinkers, able to make decisions in relation to complex problems. It was hoped that this would be achieved in these assessments through empowering students with the necessary knowledge to solve a problem and giving them an interesting domain in which to apply them.

3. CREATIVE SOFTWARE ENGINEERING ASSESSMENT SPECIFICATIONS

At the beginning of the MSc degree programme, students are initially exposed to two six-week modules running one after the other on general Java software development skills, alongside firstly

a six-week module on Computer Hardware and then a six-week module on Operating Systems. In Semester 2, students progress to six-week modules on Data Structures and Databases run in parallel with one another, followed by two six-week modules also run in parallel on Concurrent Systems and Mobile Devices and Applications. The assessment specifications for Data Structures and Concurrent Systems are presented in the remainder of this section.

3.1 Concurrent Systems

Objectives of a module on Concurrent Systems include identifying the need for concurrent systems, providing an understanding of the issues and requirements to be addressed when designing and developing such systems, and providing opportunities to develop practical systems illustrating aspects of concurrent systems.

Two assignments for the Concurrent Systems module were based on Pepper, a humanoid robot [3]. Pepper is an emotion engine designed to make people happy. He does this through delivering jokes based on the emotions sensed from humans. Another assignment on Concurrent Systems was based on the creation of a system using Java to represent operation of the Android operating system. Both assignments were based around the development of systems where concurrent operation is ultimately the focus.

3.1.1 Concurrent Systems: Simulate Pepper Operation

The first assignment was to implement a program using Java that would simulate operation of Pepper. It was required to be a multi-threaded solution, with each thread representing sensed data being fed into the operating system for processing from Pepper's ears, eyes, and hands.

More specific system requirements were also defined in the specification in relation to each thread: Threads should be created to represent sensed data from Pepper's ears, each of which requires 20 bytes of RAM per second. Similarly, Pepper's eyes require 30 bytes of RAM per second, and Pepper's hands require 40 bytes of RAM per second. The total system capacity is 1,000 bytes of RAM. The OS needs 300 bytes of RAM to run, and the supporting activities, including drivers, required by the operating system take up 200 bytes of RAM. After loading the OS so that it is ready to accept workload, there are subsequently 500 bytes remaining for application and other system activity. The CPU processes workload at a rate of 200 bytes per second.

This scenario is essentially the Producer-Consumer problem, with a requirement for multi-process synchronization. The queue into which sensed data arrives is a fixed-size buffer, with a restricted amount of space to support application and system workload. The producers generate the sensed data, passing it to the ports into the operating system via the buffer, which is shared with a consumer. At the same time that the producer is producing workload, the consumer is consuming the data, removing it from the buffer one piece at a time. The robot's engines can be considered to be the consumer, processing jobs and enforcing decisions from the system. The challenge is to ensure that the producer will not try to add data into the buffer if it's full, and that the consumer will not try to remove data from an empty buffer. To avoid these occurrences, the producer either goes to sleep or discards data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer to be empty. The next time the producer puts data into the buffer, it wakes the sleeping consumer.

Each thread runs for a period of time dependent on the sensed motion duration, or the number of bits being stored to disk and the CPU's processing capability. RAM availability influences the operating system's ability to support threads simultaneously.

Marks were awarded in this assessment for achievement of the required functionality (35%), technical quality of the program code (35%), dealing correctly with multiple threads (10%), adherence to good programming practices (10%), and clarity of the instruction sheet/booklet (10%).

3.1.2 Concurrent Systems: Pepper as a Client-server System

For the second Pepper-based assignment, students were organized into pairs and were required to implement a program using Java to simulate a connection between Pepper as the client and a remote database as the server. A server was required to hold tailored responses to be delivered by Pepper based on the "sensed" emotions of humans interacting with the robot. One student in the pair was required to be responsible for the client program and one student for the server program. As Pepper is an emotion engine, the database was required to return jokes to a user when a sad emotion is sensed. As a restriction, a joke was allowed to be returned once only within a session. The user should also have the capability to select a genre of a joke. The "database" on the server side of the system could be held within arrays. The system was required to use TCP sockets at the client and server sides of the network to support communication.

Marks were awarded for achievement of the required functionality (35%), technical quality of code (35%), dealing correctly with TCP socket programming (15%), communication issues (robustness of software, error handling) (5%), and adherence to good programming practices (5%).

3.1.3 Concurrent Systems: Android OS

In another Concurrent Systems assignment, students were required to implement a program using Java to simulate a multi-threaded Android operating system. The system was required to support simultaneous application threads, including a thread to start a BubbleWitch2 session lasting 10 seconds and requiring 100 bytes of RAM per second, and a thread to start a 20-second Spotify stream requiring 250 bytes of RAM per second. A system and management thread was also incorporated, requiring 50 bytes of RAM per second and to execute for a random duration of time once invoked. Controlling execution of the system for the purpose of demonstrating its operation, students were required to implement a thread to install a new security update of 2KB, which requires 150 bytes of RAM per second while installing. Overall capacity within the system is 1,000 bytes of RAM; the OS needs 300 bytes of RAM to run, and the drivers consume 200 bytes of RAM. After loading the OS so that it is ready to accept workload, there are 500 bytes remaining for application and other system activity. The CPU processes workload at a rate of 200 bytes per second.

Marks were awarded for technical quality of the implementation (35%), achievement of the functional requirements (35%), dealing correctly with multiple threads and robustness of the software (10%), and structure and presentation of the program (20%).

3.2 Data Structures

An objective of a module on Data Structures includes to provide students with skills in using and implementing abstract data types. The development of a social networking website, similar to Facebook, was a coursework assignment which lent itself easily to

a module on Data Structures. Students were asked to develop a social networking website using as many different data structures as possible. These would typically be used to retain data associated with each user account. Students were also asked to implement algorithms at the back-end of the system to search this stored data, so, for example, identifying people who might be their friends, or providing a reminder of a friend's upcoming birthday. Their design would be assessed in terms of the efficiency of the operation of the site, a feature which would be influenced by the most efficient data structures, sorting and searching techniques taught during the module. Certain data structures are more appropriate for certain types of information than for others, and it was the student's responsibility to select the most appropriate structure and to justify their choices.

It was therefore an assumption of the system implementation that a repository of information would be retained to support demonstration of the full system functionality, including options on pages which a user may "like" and a number of users who hold accounts with the system which a user may add as a friend.

Appropriate structure selection was important in relation to the efficiency and reliability of its operation, where efficiency is measured by the latency to execute a request and volume of memory processed when executing an operation, and reliability is measured in terms of the effectiveness of recommendations made by the system for individual users.

Conditions were also required to be applied in the scenarios implemented, such as, for example, the requirement that adding a new account would require checking that the user does not already have an account in the system — this would make demands on a searching technique implemented — or that deleting a user account would require deleting the user as a friend of other system users — a feature also requiring efficient searching techniques.

Marks were awarded for achievement of the functional aspects (30%), technical quality of the implementation (40%), effectiveness of the design choices (15%), and originality of the design (15%).

4. SOFTWARE ENGINEERED SOLUTIONS

In Section 4, a selection of the programme code solutions are presented. These are used to demonstrate the ways in which students harnessed the technical concepts learnt during each modules' teaching to create the software solution. The solutions presented are specific to one specific student for each module.

4.1 Concurrent Systems: Pepper Operation Simulation

In a software solution for the first Concurrent System assignment, a student created a Producer and Consumer class, which would communicate with each other via a shared buffer to achieve simultaneous movement of Pepper's limbs.

4.1.1 Initializing the Pepper Programme

To begin program execution, a buffer is initialized with 500 bytes of capacity (queue) (which fulfils the requirement set out in the assessment specification that the original 1,000 bytes available is also consumed by the operating system and the drivers which it needs). A random period of time (length) defines the length of time which the program should execute; a requirement of the coursework is that the program runs through all operations to demonstrate concurrent execution of the robot, and not to require external input to trigger events. After the processor and system and management threads are started, the moveable elements of the robot are invoked, including Pepper's eyes, ears, and hands.

```
public class Pepper {
   public static void main(String[] args) {
      PepperBuffer queue = new PepperBuffer (500);
      Random length = new Random();
      new PepperCPU(queue).start();
      new PepperSenses("System & Management", 50,
      queue, length.nextInt(20), 10).start();
      new PepperSenses("eye 1", 20, queue,
         length.nextInt(20), 10).start();
      new PepperSenses("eye 2", 20, queue,
         length.nextInt(20), 10).start();
      new PepperSenses("ear 1", 30, queue,
         length.nextInt(20), 10).start();
      new PepperSenses("ear 2", 30, queue,
         length.nextInt(20), 10).start();
      new PepperSenses("hand 1", 40, queue,
         length.nextInt(20), 10).start();
      new PepperSenses("hand 2", 40, queue,
         length.nextInt(20), 10).start();
   }// main
}//class
```

At this stage, the system producer and consumer classes are required to both push and pull workload to and from the shared buffer.

4.1.2 Pepper Producer Class

The Pepper Producer class achieves the functionality of generating workload, in the sense of movements from each of Pepper's 'body' parts. These are added into the shared buffer for the processing. In a live Pepper deployment, each sense consumed from the shared queue would result in an aspect of Pepper moving.

The concurrent threads of Pepper's system are represented in this solution using the PepperSenses class.

```
public class PepperSenses extends Thread {
   private int amountOfRam, lengthOfTime;
   private PepperBuffer queue;
   public PepperSenses(String sense,
      int amountOfRam, PepperBuffer queue,
      int lengthOfTime, int Priority) {
      this.setName(sense);
      this.amountOfRAM = amountOfRAM;
      this.queue = queue;
      this.lengthOfTime = lengthOfTime;
      this.setPriority(priority);
   }
   public void run() {
      for (int seconds = lengthOfTime; seconds >
      0; seconds--) {
      if (((int) (Math.random() * 2) +1) == 1) {
         this.pause();
      }
      else {
         this.actionOccurred();
      }
      try {
         sleep(1000);
      } catch (InterruptedException e) {}
      }
   }
}
```

Once the program is initiated (using run()), a pause() is invoked at random to force the program to wait for a period of time to simulate delay between activity associated with each of Pepper's senses. actionOccurred() is invoked during intervals outside the pause periods, which forces workload to be added into the shared buffer queue:

```
public void actionOccurred() {
   queue.put(this);
}
```

This workload is queued for the CPU, as the consumer, to extract and process. This is possible due to creation of the PepperBuffer object within PepperSenses, and initialization of the availableRAM value:

```
public PepperBuffer(int totalRAM) {
   if (totalRAM <= 0)
      throw new IlegalArgumentException("Size
      is illegal");
   this.totalRAM = totalRAM;
   this.availableRAM = totalRAM;
}
```

The put() method within PepperSenses simulates producer functionality. The wait() method is invoked until there is space in the buffer to allow the job to be placed there; notifyAll() is then invoked to communicate to the consumer that there is workload available to be consumed:

```
public synchronized void put(PepperSenses sense) {
   int ram = sense.getAmountOfRAM();
   while(noSpace(ram)) {
      try {
         wait();
      } catch(InterruptedException ex) {
      }
   }
   buffer.add(sense);
   availableRAM -= ram;
   notifyAll();
}
```

The noSpace() method checks if the buffer is full, preventing new jobs from being added, in which case the producer will wait:

```
public synchronized Boolean noSpace(int ram) {
   return (availableRAM < ram);
}
```

4.1.3 Pepper Consumer Class

The consumer removes workload from the buffer as each sensed event becomes available and the consumer is notified of its arrival.

```
public class PepperCPU extends Thread {
 public void run() {
  while(true) {
    while(cpuAvailable >= 0) {
     PepperSense sense = queue.get();
      setCpuAvailableRemove(sense.getAmountOfRAM());
      sense.sleep();
    }
    try {
      this.sleep(1000);
   }catch (InterruptedException e) {
      Thread.currentThread().interrupt();
   }
   setCpuAvailable();
 }
}
```

The consumer first checks if the buffer is empty, which will be the case when there are no jobs to remove. In this event, the consumer will wait until workload has been added to the queue. When workload is present in the queue, on the other hand, jobs are removed by invoking the get() method:

```
public synchronized PepperSenses get() {
   PepperSenses sense;
   while(isEmpty()) {
      try {
         wait();
      } catch(InterruptedException ex) {
      }
   }
   sense = buffer.remove(0);
   availableRAM += sense.getAmountOfRAM();
   notifyAll();
   return sense;
}
```

The consumer then removes the workload and invokes notifyAll() to inform the producer that there is increased space in the buffer to accept new workload. The availableRAM value is also updated to reflect the amount of RAM now available in the queue in response to the robot movement having been dequeued.

The concept of Pepper provided a suitable context to support the development of a concurrent system, fulfilling two of the module objectives to "identify the need for concurrent systems" and to "provide opportunities to develop simple practical systems illustrating specific aspects of concurrent systems." Furthermore, creation of the solution required that students had an appreciation of the main components of a concurrent system, such as the need to have a shared buffer, and one producer and one consumer to use it. This helped to fulfil the learning objective of the module, to "provide an understanding of the issues and requirements to be addressed when designing and developing such systems." Additionally, having this awareness required that the fourth learning objective had been fulfilled to "introduce the underlying principles of concurrent systems." Organizing the development around the concept of Pepper helped students to appreciate the wider context within which their learning exists.

4.2 Concurrent Systems: Pepper as a Client-Server System

In the solution, the client and server were required to connect to the same port in order to communicate, continuously listening on the same socket for communications between each other.

In the solution presented, the client is created and initialized using the PepperClient class:

```
public class PepperClient {
   public static void main(String args[]) {
      Scanner keyboard = new Scanner(System.in);
      String serverName = "193.61.167.145",
      username="", response;
      Socket serverSocket;
      int serverPort = 3829;
      InputStream isFromServer;
      OutputStream osToServer;
      DataInputStream disFromServer;
      DataOutputStream dosToServer;
      try {
         serverSocket = new Socket(serverName,
         serverPort);
         isFromServer =
         serverSocket.getInputStream();
         osToServer = serverSocket.getOutputStream();
```

```
            disFromServer = new
            DataOutputStream(osToServer);
         } catch(Exception e) {}
      }
   }
```

PepperClient is associated with port 3829. Data streams are established to facilitate the communications, in directions both to the server (OutputSteam osToServer) and from the server (DataInputStream disFromServer). Sockets are additionally created to support each data stream.

Data streams and sockets are also established on the server side in parallel:

```
public class PepperServer {
   public static void main(String[] args) throws
      IOException {
         InputStream is;
         OutputStream os;
         DataInputStream disFromClient;
         DataOutputStream dosToClient;
         Socket clientSocket;
         ServerSocket listenSocket;
         int clientInt;
         int serverPort = 3829;
         listenSocket = new ServerSocket(serverPort);
         clientSocket = listenSocket.accept();
         is = clientSocket.getInputStream();
         os = clientSocket.getOutputStream();
         disFromClient = new DataInputStream(is);
         dosToClient = new DataOutputStream(os);
         boolean live = true;
   }
}
```

To facilitate the end-to-end communication, the server also listens on port 3829. After the definition of these classes, the client and the server are in a position to interact with one another. This requires capability to communicate the mood of the user who is interacting with Pepper. It also requires that the database of jokes is created for return to the user in the event that their mood is one of sadness.

Due to the fact that this system is a simulation of a robot, it was necessary to explicitly articulate the simulated user's mood. In a live system, this information might be autonomously collected using the sensors for Pepper's eyes to identify their facial characteristics, or Pepper's ear sensors to detect what the user is saying. In the simulated system, this functionality is achieved by the user entering their emotion using a keyboard, in response to a prompt delivered by PepperClient:

```
public static int askForEmotion(String name) {
   Scanner keyboard = new Scanner(System.in);
   String feeling;
   System.out.print("Tell me " + name + " do you feel
   sad (Y/N: ");
   feeling = keyboard.nextLine();
   if (feeling.equalsIgnoreCase("N")) {
      System.out.println("\nHow do you feel " + name
      + "?" +
      "\n1. Happy" +
      "\n2. Angry" +
      "\n3. Hungry" +
      "\n4. Scared" +
      "\n5. I want you to leave me alone");
      return (keyboard.nextLine().charAt(0)-48);
   }
   else {
      return 6;
   }
}
```

PepperClient asks the user to enter an integer which indicates their mood, in the instance that they are not feeling sad. In the instance that the user reports that, in fact, they are feeling sad, this will be communicated to PepperServer. Jokes are then returned to the user from PepperServer.

```
while(live) {
   emotion = askForEmotion(userName);
      switch(emotion) {
         …
         case 6:
         jokeCategory = jokeGenre(username);
         String reply;
         int noOfReplys;
         dosToServer.writeInt(jokeCategory);
         noOfReplys = disFromServer.readInt();
         for(int index=0; index < noOfReplys;
            index++) {
            reply = disFromServer.readUTF();
            System.out.println(reply);
         }
      break;
      default:
      System.out.println("I didn't understand");
      }
   }
}
```

In the case of selections 1, 2, 3, 4, or 5, Pepper will take the action of exiting the user from the session by closing the socket, or will return a statement to the user depending on them being happy or hungry, essentially any case where they are not sad. In the case that the user is sad, Pepper will return a joke to the user from the joke database. Pepper attempts to return a joke which is personalized for the user, by considering their preferred genre of joke. The user therefore has a choice of selecting a knock-knock joke, a one-liner joke, a chicken-crossing-the-road joke, a computer joke, or a pun.

Pepper similarly provided an appropriate opportunity for students to exploit their learning of Java socket programming in a remote client-server setup. This provided an opportunity for students to demonstrate their "understanding of the issues and requirements to be addressed when designing and developing such systems," by appreciating how a robot in fact interacts with a remote server when responses are selected for return. Again, the "need for the concurrent system" is highlighted in this situation, where it is essential that interactions are delivered in the correct order in the support of a meaningful "conversation."

4.3 Concurrent Systems: Android OS

An OperatingSystem class is created in one implementation, which creates and initializes the shared buffer that retains workload added by the producer and removed by the consumer.

```
class OperatingSystem {
   private int contents;
   private int buffer = 1000, driver = 200,
   operSRun = 300, consumptionRate = 200,
   workload = buffer- (driver+operSRun);
   private long startTime;
   private long endTime, waitTime;
   private int waitCount = 1;

   public synchronized int get() {
      while(workload <= (driver + operSRun)) {
         try {
            wait();
         } catch (InterruptedException e) {}
      }
```

```
      workload = workload – consumptionRate;
      notifyAll();
      return consumptionRate;
      }
      public synchronized void put (int amount) {
         while (workload >= buffer) {
            startTime = System.currentTimeMillis();
            try {
               wait();
            } catch (InterruptedException e) {}
            endTime = System.currentTimeMillis();
            waitTime = (endTime – startTime);
            averageRequestTime(waitCount, waitTime);
         }
         contents = amount;
         workload = workload + amount;
         notifyAll();
      }
   }
```

The OperatingSystem class initializes attributes used to support the scenario, including the buffer size. It also contains the necessary put() and get() methods, which are responsible for allowing workload to be added and removed to and from the queue. When workload is inserted into the queue (put(int amount)), the system first checks that the size of the queue, if the workload were to be added, does not exceed the maximum possible size. In the case that the action would result in a queue of unfeasible size, the wait() method is invoked, meaning that the thread will wait until it has been notified that workload has been removed and that there is now space in the queue. Once the workload can be added, the queue size will be updated, and the consumer thread will be notified that there is workload available for processing, using notifyAll(). The newly added workload will be removed.

Application classes are also defined to support execution of system threads. These threads are responsible for adding workload to the shared queue. As one example, a class Spotify simulates activity for 20 seconds and requires 250 bytes of RAM per second:

```
   private class Spotify extends Thread {
      private OperatingSystem operatingSystem;
      private int number, loops;
      private int long startTime;
      private long endTime;

      public Spotify(OperatingSystem os, int number,
      int duration) {
         operatingSystem = os;
         this.number = number;
         this.loops = duration;
      }
      public void run() {
         startTime = System.currentTimeMillis();
         for (int i = 0; i < loops; i++) {
            operatingSystem.put(250);
            try {
               sleep(1000);
            } catch (InterruptedException e) {}
         }
         endTime = System.currentTimeMillis();
      }
   }
```

Each application thread extends the Java Thread class. The constructor for the thread sets up the number of the thread and the duration of the thread; the duration is used to ensure that the thread executes for the necessary interval of time. Enforcing that the thread executes sleep(1000) results in a one second delay between operatingSystem.put(250) being invoked with each loop iteration, therefore helping to simulate the thread running for a period of time.

The invocation of each application thread simultaneously is controlled using the System class.

```
   public class System {
      private static int count;
      private static long time = 0;
      private static long average;

      public static void main(String[] args) throws
      InterruptedException {
         long startTime;
         long endTime;

      OperatingSystem os = new OperatingSystem();
      BubbleWitch2 bWitch2 = new BubbleWitch2(os, 1,
         10);
      Spotify Spotify = new Spotify(os, 2, 20);
      SystemAndManagement sysAndManagement = new
         SystemAndManagement(os, 3);
      CPU processor = new CPU(os, 1);
      SecurityUpdate securityUpdate = new
         SecurityUpdate(os, 4, bWitch2, Spotify,
         sysAndManagement, processor, 15, 2000);
      startTime = System.currentTimeMillis();

      bWitch2.start();
      Spotify.start();
      sysAndManagement().start();
      securityUpdate.start();
      processor.start();
      securityUpdate.setPriority(1);
      securityUpdate.join();
      processor.stopRunning();
      average = time / count;

      endTime = System.currentTimeMillis();
      }
      public static void averageRequestTime(int
         waitCount, long waitTime) {
         count += waitCount;
         time += waitTime;
      }
   }
```

The BubbleWitch2 and Spotify threads are initialized using their constructors, with information which includes the OperatingSystem with which they are associated, the thread number, and the duration of time which they are required to run. The systemAndManagement thread does not indicate a duration, as it is required to execute throughout the lifetime of the application. Similarly, the processor thread will also be available continuously.

When the SecurityUpdate thread is created, the other application threads are passed through the constructor so they can be joined with SecurityUpdate. Java's .join() method supports one thread waiting while other threads complete execution. Invocation of securityUpdate.join() will result in the securityUpdate thread being paused when another thread with a higher priority is in an executable state. The securityUpdate thread is given a priority of 1. This is the lowest priority which may be assigned to a thread, and enforces that this thread is executed with minimum priority.

```
   public class SecurityUpdate extends Thread {
      private OperatingSystem operatingSystem;

      public SecurityUpdate(OperatingSystem os, int
         number, BubbleWitch2 bWitch2,
```

```
      Spotify Spotify,
      SystemAndManagemetn systemAndManagement, CPU
      processor, int desired, int max) throws
      InterruptedException {
         operatingSystem = os;
         this.number = number;
         this.loops = desired;
         this.max = max;

         bWitch2.setPriority(10);
         Spotify.setPriority(10);
         systemAndManagement.setPriority(10);
         processor.setPriority(10);

         bWitch2.join();
         spotify.join();
         systemAndManagement.join();
         processor.join();
   }

   public void run() {
      …
   }
}
```

The priorities assigned to application threads enable processor capacity to be assigned to each application as it becomes available, depending on the duration of the thread. Invocation of the .join() method in association with each application thread enforces that the securityUpdate thread will be the last to execute.

Use of the Android operating system allowed students to appreciate the role which concurrent systems play in their day-to-day lives. The module learning objective to "provide an understanding of the issues and requirements to be addressed when designing and developing such systems" was highlighted in this assignment, with the need to consume residual memory to support an operating system and additional drivers and then support the application threads simultaneously around that.

4.4 Data Structures

There were two primary aspects of the Data Structures assessment, firstly in terms of the structures used to hold data, and secondly in terms of the algorithms used to organize and search the data structures. Assessments for this module are therefore considered from these perspectives in the following sections.

4.4.1 Setting Up Data Structures

A stack is used to retain the news feed for the social network using the Java Stack class: Following the Facebook approach, the news feed presents the most recent news item, the most recently added element to the stack, first. This is possible due to the fact that the class operates on a last-in first-out approach (LIFO). Extracting the most recent news update can therefore be achieved by "popping" the top item from the stack. New items of news are added by "pushing" them onto the stack, and are continuously pushed down the stack as new items are added.

```
class Stack {
   private int maxSize;
   private User[] stackArray;
   private int top;

   public void push(User j) {
      stackArray[++top] = j;
   }
   public User pop() {
      return stackArray[top--];
   }
   public Boolean isEmpty() {
      return (top == -1);
   }
   public Boolean isFull() {
      return (top == maxSize – 1);
   }
}
```

When items are pushed on or popped from the stack, a counter is maintained (top), allowing the size of the stack to be captured.

As another example of a data structure implemented, an array is used to capture the personal details of users of the system:

```
allUsers[j] = new User(firstName, lastName,
dateOfBirth, homeLocation, emailAdd, password,
employer, school);
```

The array effectively stores several items of the same type.

4.4.2 Organising and Searching Data Structures

A user is added as an object into a sorted list of system members:

```
public void addUser(String firstName, String
   lastName, Date dateOfBirth,
   String homeLocation,
   String emailAdd, String password,
   String employer, String school) {
      for (int j = 0; j < totalUsers; j++) {
      if (allUsers[j].getEmailAdd().
       compareTo(emailAdd) > 0)
      break;
      }

      for (int k = totalUsers; k > j; k--) {
         allUsers[k] = allUsers[k – 1];
      }
      allUsers[j] = new User(firstName, lastName,
      dateOfBirth, homeLocation, emailAdd,
      password, employer, school, currentStatus,
      statusTime);
      totalUsers++;
   }
}
```

The correct position in the array is identified by searching through email addresses, which are sorted into alphabetical order using .compareTo(emailAdd). This compares the email address being added with the email address at the position in the array currently being searched. If the result of the comparison is a positive integer, the email address lexicographically follows the argument string, and it should be added at this position. Items currently in the array beyond this point are shifted down by one position to make space for the new item being added.

New friends are added into a friend list, again sorted according to email address and using an insertion sort:

```
public void emailSort() {
   int in, out;
   for (out = 1; out > totalUsers; out++) {
      User temp = allUsers[out];
      in = out;
      while (in > 0 &&
      allUsers[in – 1].getEmailAdd().
      compareTo(temp.getEmailAdd()) > 0) {
         allUsers[in] = allUsers[in – 1];
         --in;
      }
      allUsers[in] = temp;
   }
}
```

An insertion sort on email address is used to organize the data associated with an individual account so that it is organized in the most efficient way when it comes to search the data. The insertion sort considers each list element from left to right, comparing each one by one. It places the data element in its correct location within the sorted list. The process is repeated until there are no unsorted elements remaining. The algorithm therefore operates by comparing the current email address with the email address which precedes it.

Capability was integrated to support account deletion. This requires that the account is also removed from their friend' lists.

```
public void delete(int userIndex) {
   int indexInFriendLists;
   for (int I = 0; I < totalUsers; i++) {
      indexInFriendLists =
      allUsers[i].friendFind(
      allUsers[userIndex].getEmailAdd());
      if (indexInFriendsLists >= 0) {
        for (int startPosition =
        indexInFriendLists; startPosition <
        allUsers[i].getFriendCount();
        startPosition++) {
          allUsers[i].setFriendsLists(
          startPosition, allUsers[i].
          getFriendsList()[startPosition + 1]);
        }
        allUsers[i].setFriendCount(-1);
      }
   }
   for (int startPosition = userIndex;
   startPosition < totalUsers; startPosition++) {
      allUsers[startPosition] =
      allUsers[startPosition + 1];
      totalUsers--;
   }
}
```

Deleting requires that the friendsList array for each account holder is also searched, and the person who is deleting their account is also removed from their list of friends.

A binary search is applied to find friends within a user's friend list, with the assumption that a friend of a friend is a plausible option for a friend recommendation.

```
public int friendFindEmail(String email,
int lowerBound, int upperBound) {
   int curIn;
   curIn = (lowerBound + upperBound) / 2;

   if (lowerBound > upperBound)
      return -1;
   else if
   (friendsList[curIn].
   getEmailAdd().compareTo(email) == 0)
      return curIn;
   else {
      if (friendsList[curIn].getEmailAdd().
      compareTo(email) < 0)
         return friendFindEmail(email, curIn + 1,
         upperBound);
      else
         return friendFindEmail(email, lowerBound,
         curIn – 1);
   }
}
```

The binary search compares the target with the middle list element. If the values are not equal, the half where the target cannot reside is eliminated, and search continues in the remaining half until successful.

A method is incorporated to search for friends which have a birthday in the current month or in the next month:

```
public void displayBirthdays(int currentUserIndex) {
   Date now = new Date();
   Stack birthdayStack = new Stack(totalUsers);

   for (int i = 0; i <
   allUsers[currentUserIndex].getFriendCount();
   i++) {
      if (allUsers[currentUserIndex].
      getFriendsList()[i].getDateOfBirth().
      getMonth() == now.getMonth() + 1) {
         birthdayStack.push(allUsers[
         currentUseIndex].getFriendsList()[i]);
      }
   }

   if (!birthdayStack.isEmpty()) {
      while (!birthdayStack.isEmpty()) {
         User temp = birthdayStack.pop();
         System.out.println("Birthdays!");
         System.out.println(temp.getFirstName() +
         " " + temp.getLastName());
      }
   }
}
```

A number of friends are associated with each user. The system therefore captures the number of friends and searches through their birthdays to find if any have a birthday in the current month, with the objective of providing the user with a reminder. These identified contacts are subsequently "pushed" onto a stack for temporary storage during the user's session, such that they may be output for information. If the stack is not empty, a user is informed that birthdays of their friends according to their contact list are coming up; the relevant users are popped from the temporary stack.

As with simulation of the Android operating system as a concurrently operating environment with which students have first-hand and day-to-day experience, the social media Facebook-type platform similarly has such relevance to students. Students could appreciate, for example, how functionality is provided by the newsfeed feature; harnessing the use of abstract data types, as a learning objective of the Data Structures module, allowed students to appreciate the software development operating in the backend to support popular social media environments.

5. CONCLUSIONS

A conversion degree provides new opportunities, particularly in relation to employability, by training students in areas for which they were not previously academically accredited. When students were asked about their expectations for the study year, both positive (Table 2) and negative (Table 3) angles were presented.

Student ambitions were clear (Table 2), with expectations including gaining new skills and knowledge, and ideally a job. Coursework cropped up as a weight about which students had negative expectations (Table 3). When considered in relation to their desire for improved employability options, it was therefore important to support students in their ambitions post-degree and remove their coursework fear. It is believed that the creative assessment design helped to bridge these gaps, by exposing students to state-of-the-art technology on an international basis, helping them to understand

the software developments which are essential in their support at the back-end and encouraging the application of knowledge in new ways. In doing so, the creative assessment designs provided a mechanism to facilitate student innovation in their output.

Table 2. Student expectations of the year (positive).

What are you looking forward to this year?
"*Challenge of learning something completely new*"
"*To meet like-minded individuals to create something together to leave with business ideas*"
"*Gaining a masters and a job*"
"*Learning practical programming skills, how computers work, how software programs are created*"
"*To be challenged and learn new skills*"
"*Getting a job*"
"*Gaining experience in software programming*"
"*Challenges that the course will bring*"

Table 3. Student expectations of the year (negative).

What are you not looking forward to this year?
"*Travelling to and from university*"
"*Travelling, exams, formal aspects of course, struggle to sit down and write about a task*"
"*Hard to learn a new discipline at such a fast pace*"
"*Using mathematical formula*"
"*Coursework*"
"*Falling behind, not understanding something*"
"*Exams*"
"*Parts of the lecture which are not understood, mainly due to the terminology*"
"*Written assignment*"

6. REFERENCES

[1] Great Schools Partnership, "*Student Engagement*," The Glossary of Education Reform; Available: https://www.edglossary.org/.

[2] J. McCarthy, "*Learner Interests Matters: Strategies for Empowering Student Choice*," Edutopic, Aug. 2014; Available: https://www.edutopia.org/blog/differentiated-instruction-learner-interest-matters-john-mccarthy.

[3] SoftBank Robotics, "*Who is Pepper*?" Online; Available: https://www.softbankrobotics.com/emea/en/robots/pepper.

[4] C. Gewertz, "*'Soft Skills' in Big Demand*," Education Week, Jun. 2007; Available: https://www.edweek.org/ew/articles/2007/06/12/40soft.h26.html.

[5] Council for the Curriculum, Examinsation and Assessment, "*What is Summative Assessment?*" Online; Available: http://ccea.org.uk/curriculum/assess_progress/types_assessment/summative.

[6] International Baccalaurete, "*The IB Learner Profile*," Online; Available: https://www.ibo.org/benefits/learner-profile/.

[7] D. King and L. D. English, "*Engineering Design in the Primary School: Applying STEM Concepts for Build an Optical Instrument*," International Journal of Science Education, Dec. 2016, pp. 2762–2794; DoI: 10.1080.09500693.2016.1262567.

[8] S. Karen and S. Gregg, "Alternative Assessment – Can Real-world Skills be Tested? Policy Briefs," National Library of Australia, 1993.

[9] S. S. Alfuhaigi, "*School Environment and Creativity Development: A Review of Literature*," Journal of Educational and Instructional Studies, Vol. 5, Iss. 2, May 2015, pp. 33–37.

[10] B. Irwin and S. Hepplestone, "*Examining Increased Flexibility in Assessment Formats*," Assessment & Evaluation in Higher Education, 2012, pp. 773–785.

[11] A. Craft, "An Analysis of Research and Literature on Creativity in Education: Report prepared for the Qualifications and Curriculum Authority," Qualifications and Curriculum Authority, Mar. 2001.

[12] P. Kampylis and E. Berki, "*Nurturing Creative Thinking*," International Academy of Education, UNESCO, 2014.

[13] JISC, "*Transforming Assessment and Feedback with Technology*," Online; Available: https://www.jisc.ac.uk/full-guide/transforming-assessment-and-feedback.

[14] Montessori Northwest, "*What is Montessori Education*?" n.d. Online; Available: https://montessori-nw.org/what-is-montessori-education/.

[15] An Everyday Story, "*What is the Reggio Emilia Approach*?" Online; Available: http://www.aneverydaystory.com/beginners-guide-to-reggio-emilia/main-principles/.

[16] N. Jackson, "Assessing Students' Creativity: Synthesis of Higher Education Teacher Views," The Higher Education Academy, Jun. 2005.

[17] Warwick University, "*Marking Creative Writing*" Online; Available: https://warwick.ac.uk/fac/arts/english/currentstudents/undergraduate/modules/fulllist/second/en232/marking_creative_writing/.

[18] S. M. Brookhart, "*Assessing Creativity*," Educational Leadership, Vol. 70, No. 5, Feb. 2013; Available: http://www.ascd.org/publications/educational-leadership/feb13/vol70/num05/Assessing-Creativity.aspx.

[19] G. Hill and S. J. Turner, "*Electronic Online Marking of Software Assignments*," Progress in IS: Software Engineering Education fro a Global E-service Economy, 2014, pp. 41–48.

[20] A. Venables and L. Haywood, "*Programming Students need Instant Feedback!*," in Proceedings of 5th Australasian Computing Education Conference, 2001.

[21] Learning Theories, "*Transformative Learning Theory (Mezirow)*," Online; Available: https://www.learning-theories.com/transformative-learning-theory-mezirow.html.

[22] J. Bosch and P. Molin, "*Software Architecture Design: Evaluation and Transformation*," Proceedings of IEEE Conf. and Workshop on Engineering of Computer-based Systems, Mar. 1999.

[23] BBC, "*International Baccalaureate*," Online; Available: http://www.bbc.co.uk/schools/parents/international_baccalaureate/.

[24] Vanderbilt, "*Motivating Students*," Online; Available: https://cft.vanderbilt.edu/guides-sub-pages/motivating-students/.

[25] L. Gueldenzoph Snyder and M. J. Synder, "*Teaching Critical Thinking and Problem Solving Skills*," The Delta Pi Epsilon Journal, Vol. L, No. 2, 2008, pp. 90–101.

[26] S. P. Kearney and T. Perkins, "Engaging Students through Assessment: The Success and Limitations of the ASPAL (Authentic Self and Peer Assessment for Learning) Model," ResearchOnline@ND, 2014.

[27] DePaul University, "*Authentic Assessment "Assessing by Doing*"," Online; Available: https://offices.depaul.edu/teaching-learning-and-assessment/assessment/assessing-learning/Documents/AuthenticAssessmentInformationSheet.pdf.