Preparing the Software Engineer for a Modern Multi-Core World

Nasser Giacaman, Oliver Sinnen

{n.giacaman,o.sinnen}@auckland.ac.nz

Parallel and Reconfigurable Computing group

Department of Electrical and Computer Engineering

The University of Auckland, New Zealand

Correspondence

Email: n.giacaman@auckland.ac.nz

Telephone: +64 9 923 8156

Fax: +64 9 373 7428

Address: 368 Khyber Pass, Newmarket, Auckland 1023, New Zealand.

Abstract

Parallel and Distributed Computing (PDC) was traditionally viewed as an advanced subject reserved for elective graduate courses. The last decade has seen two areas with rapid growth, whose synergy is demanding new skills for software engineers in a modern multi-core world. The first has been society's increasing demand for software engineering solutions, evident in the integral role that software plays in daily life. Unlike traditional PDC applications in the scientific and engineering domains, modern software applications are interacting *directly* with millions of users on mainstream laptops, smartphones and tablets. The second trend is that of multi-core processors powering such devices, which is further fueling the potential of software applications. This paper proposes a Modern Parallel Programming Framework that recognizes that successful software engineering in this domain involves a combination of hard skills and soft skills. A course dedicated to this goal is presented and evaluated, incorporating a research-infused, problem-based and active learning approach.

Keywords: Software Engineering, Concurrency, Parallel Programming, Active Learning, Research-Infused Learning, Problem-Based Learning, Soft Skills, Graphical User Interfaces, Object-Oriented Programming.

1 Introduction

Much like Computer Science, the Software Engineering degree teaches students the core elements of hardware and software, while at the same time developing their problem solving skills. However, the Software Engineering degree places a much higher emphasis on the application of that knowledge and takes students through the complete software development process; this includes working in teams to gather requirements, designing, building, testing and deploying the final software product. The objective of Software Engineering is to prepare students for meeting industry needs, by taking responsibility to manage and deliver software projects. Internationally accredited engineering degrees, for example those adhering to the Washington Accord [1] or Sydney Accord [2], are expected to produce undergraduate students that are prepared for engineering practice.

A recent trend in industry has come in the form of ubiquitous multi-core everyday consumer devices, namely laptops, notebooks, tablets and smartphones. Much like traditional parallel computing devices, programmers need to explicitly incorporate multi-threading within their software applications to utilize the underlying hardware. Herein lies a major complication, namely the necessary skill-set required depends on the particular parallel hardware. While some undergraduate courses do teach parallel programming concepts, it is usually in the form of a module within a non-dedicated course (e.g. an operating systems course). The concepts taught typically include students as recipients of well grounded foundational knowledge, rarely directly involving students in the latest parallel programming research.

This paper presents a holistic approach to preparing Software Engineering students for a career in a modern multi-core world. While the emphasis is on Software Engineering, the concepts are equally relevant to disciplines such as Computer Science and Computer Systems Engineering, where the common goal is in preparing students as the software developers that industry seeks. In meeting this goal, a PDC course is discussed with the following three aspects that are believed to be pivotal to preparing students as developers in a modern multi-core world:

1. **Concepts Relevant to a New Domain:** while the NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing (PDC) [3] has mapped the fundamental PDC topics, these need to be put into context of what can be expected of a *modern* software developer. We are no longer talking about parallelizing "simple" scientific and engineering applications [4] that possess large amounts of repetitive inherent parallelism, running on dedicated and known systems. We are now in the realm of irregularly-structured computations with short runtimes, where applications are executed on non-dedicated and unknown target systems [5]. Today, multi-core mobile devices and desktops mean that software developers need to learn how to parallelize applications that were not traditionally PDC candidates [6]. This new domain of parallel computing needs to be in the context of two fundamental characteristics underlying desktop and mobile applications:

   (a) *Interactive* and *user-driven*, with a Graphical User Interface (GUI) [7] that needs to adhere to rules of GUI toolkit frameworks, and

   (b) Predominantly developed in *object-oriented* languages [8], unlike low-level (but speed-efficient) languages like Fortran or C [9] typically used in traditional PDC domains.

2. **Active Learning:** Programmers that have developed their programming strategy tend to learn more effectively than those that have not developed a strategic approach [10]. This is particularly important for software engineering students, whose engineering discipline emphasizes an inductive approach to problem solving [11]. The traditional classroom provides insufficient opportunity to develop the necessary cognitive skills in programming as it lacks relevant learning activities [12], whereas an active learning approach motivates programming students to be more engaged [13]. Debugging practice also promotes cognitive programming skills [14], and it too cannot be learnt in the traditional classroom setting; this is especially difficult in parallel applications, since parallel-specific bugs tend to be non-deterministic (e.g. race conditions), making it difficult to re-

produce the error [15]. Active Classroom Programmer (ACP) [16] can be used to address some of these issues, by allowing students to engage in lessons with parallel coding exercises executing on their own multi-core laptops.

3. **Qualities Beyond Technical Skills:** With the intense competitiveness and rapid developments in the software industry, employers are seeking qualities in developers well beyond technical skills [17]. Employers seek graduates who have demonstrated examples of challenging projects that require teamwork and critical thinking. A research-capable undergraduate computing student has an advantage, especially having proven research experience in an advanced computing area such as PDC [18]. This also borrows elements from problem-based learning (PBL), which aims to improve higher-order thinking, develop life-long learning and self-directed learning [19]. This paper suggests how these characteristics may be integrated into an undergraduate PDC course.

Contributions

An important way to effectively develop the link between research and teaching is to share discipline-specific case studies [20]; here, we share our experience with an undergraduate (senior year) course. We discuss our experiences unifying fundamental PDC topics in direct relevance to every software developer, leading students using a hands-on interactive learning approach. As instructors, we were pleased with the outcomes of this approach; based on student evaluations, we believe students are also satisfied. Table 1 shows that the overarching theme for this paper is to present the **what** and **how** for a modern Software Engineer, both in terms of **hard** and **soft** skills.

While some of these elements have been previously presented [16, 18], a much more holistic framework (known as the Modern Parallel Programming Framework) is presented here. This paper illustrates the pedagogical design decisions in delivering a PDC course that mainstream Software Engineers can value, while connecting the software industry demands and reminding educators not to solely focus on developing

	Hard skills	Soft skills
What	*Apply* PDC programming in context of: • Graphical User Interfaces • Object-oriented languages	• Teamwork • Critical thinking • Communication
How	• Active Classroom Programmer • Projects from everyday desktop and mobile device domains	• Group work • Research-infused learning • Presentations, write report

Table 1: The overarching contribution of this paper is to unify the software industry's expectations for graduates in a modern multi-core software development world. This includes both the technical hands-on PDC skills relevant to everyday domains, as well as non-technical qualities that employers seek. The soft skills are of particular interest to internationally accredited (software or computing) engineering degrees.

student technical skills. A particularly notable contribution is the role of GUI concurrency, and its importance in mainstream multi-core systems. This paper not only presents the underlying concepts to assist PDC instructors in incorporating it into their course, but also presents minimal-effort and illustrative application examples for them to use in their courses.

The rest of this paper is organized as follows: Section 2 presents some background on the Software Engineering course at the University of Auckland, SoftEng751. Section 3 answers the *what* of the hard skills required in a modern multi-core world (in the form of a framework), while Section 4 looks at *how* Active Classroom Programmer is used to give students that practical hands-on experience, by parallel programming on everyday multi-core platforms during lessons. Section 5 discusses the *what* and *how* of qualities beyond hard skills that the software industry demands. Finally, Section 6 presents an evaluation, before concluding in Section 7.

2 Background: SoftEng751

This paper presents experience of teaching a parallel computing course, SoftEng751 High Performance Computing, in the Department of Electrical and Computer Engineering at the University of Auckland. This course is part of the Software Engineering undergraduate degree, available as an elective in the final year of study. As a spe-

cialization of the Bachelor of Engineering (Honours) degree, Software Engineering is recognized by the Institution of Professional Engineers New Zealand (IPENZ) [21], which is a signatory to the Washington Accord [1]. The emphasis in this course is to develop medium-scale software projects, in teams, incorporating parallel programming. Since this course is also offered to Masters-taught students, it also needs to incorporate a research component in the form of an independent and challenging hands-on project.

There is no earlier course in the Software Engineering undergraduate degree introducing parallel programming concepts. Students are taught essential core concepts (Section 3.2) guided by the NSF/IEEE-TCPP Curriculum Initiative on Parallel & Distributed Computing [3], while also allowing sufficient time to endeavor on a larger independent research team project. In line with the Curriculum Initiative's goal, SoftEng751 aims to motivate students by embracing parallel computing as an integral component of their professional career; this is especially important in an engineering discipline such as Software Engineering. Although the course is named High Performance Computing, it does not focus on "classical HPC" content. The course focuses on *applying* parallel programming skills to *modern* software engineering domains directly interacting with people. In this regard, the focus is on applications for multi-core desktops and mobile devices rather than distributed systems.

Students are taught the essentials of shared-memory parallel programming in the first four weeks (Table 2). The 5^{th} teaching week is dedicated to introducing the project topics, while the 6^{th} week is dedicated to testing the material covered in the first four weeks. The remainder of the teaching weeks (i.e. after the study break), involve students presenting their selected topic to the rest of the class; the contents of these presentations are all examinable, with a final test in the following week concluding the presentations. Details of the topics covered, aligned with the PDC Curriculum Initiative, will be discussed in Section 3.2.

Week	Course content	
1	IT	Parallel architecture, threads, thread-safety, synchronization
2	IT	Parallel programming metrics, parallel loops, dependence analysis, scheduling
3	IT	Shared memory (task and data) parallelism using object-oriented libraries, OpenMP
4	IT	GUI concurrency and parallelization, project topics released
5	IT	Discuss project topics, assign projects to groups
6	A, P	Mid-term test, students commence projects
Study break	P	Project work
	P	Project work
7-10	ST, P	Group presentations
11	A, P	Final test, project work
12	P, A	Project work, final project submission

Table 2: SoftEng751 structure. The second column represents how the respective week was used, either instructor-led teaching (IT), assessment (A), "free-time" for project work (P) or student-led teaching (ST) in the form of group presentations.

NSF/TCPP CDER Center Early Adopter

Many elements discussed in this paper have been supported by NSF/TCPP CDER Center Early Adopter program. This is not limited to financial awards that have allowed the purchase of multi-core mobile devices for student projects, and aiding the development of teaching resources (such as some aspects of the ACP tool). As will be illustrated, the curriculum and delivery of SoftEng751 is guided by the goals and recommendations of the Early Adopter program. This course is a successful (partial) implementation of the Curriculum Initiative, but is also extended with modern Software Engineering aspects. Feedback from the EduPar workshops and community has also contributed to further developing SoftEng751 as it is presented here. An outcome of this effort, as presented here, is the easy-to-use resources for readers to immediately utilize in their PDC courses with minimal effort.

3 Framework for a Modern Parallel Programming World

Parallel computing has been actively developed and practiced for many decades now. However, it largely remained a specialized area applied to scientific and engineering domains, remaining somewhat irrelevant to the mainstream software developer. Most computing graduates are employed as application-level developers. Historically, it was permissible for software developers to neglect parallel programming in their formal training, since ubiquitous computing (desktops, laptops) contained single-core processors whose performance continued improving due to faster clock speeds. This allowed the performance of desktop applications to benefit automatically without modifying the software, but this "free lunch" has now come to an abrupt end [6].

This section discusses the fundamental concepts deemed necessary in preparing software and computing undergraduates in a world where they will most definitely need to apply parallel programming. To address this, Figure 1 presents the Modern Parallel Programming Framework (MPPF) with the following components:

1. **Ubiquitous Parallel Computing Platforms:** the first component involves recognizing the sorts of parallel hardware platforms that most software developers will encounter. This is fairly straightforward to motivate, as most students are already consumers of these platforms.

2. **Fundamental PDC Programming Topics:** with the broad range of PDC topics, only a subset of these are required at a curriculum level. While the PDC Curriculum Initiative [3] is of great assistance, our framework only selects a subset of those topics likely to be of direct relevance to most graduates.

3. **GUI Concurrency:** this component acknowledges a topic that is omitted from the PDC Curriculum Initiative, but is most definitely deserving when it comes to concurrency (and parallelism by extension) on the ubiquitous parallel platforms and the nature of applications graduates will develop.

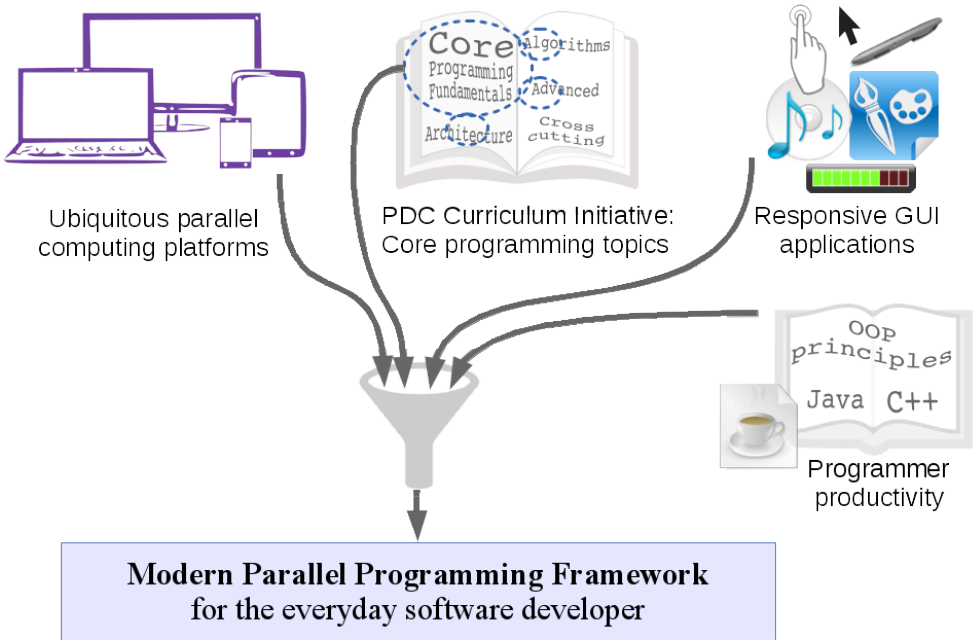
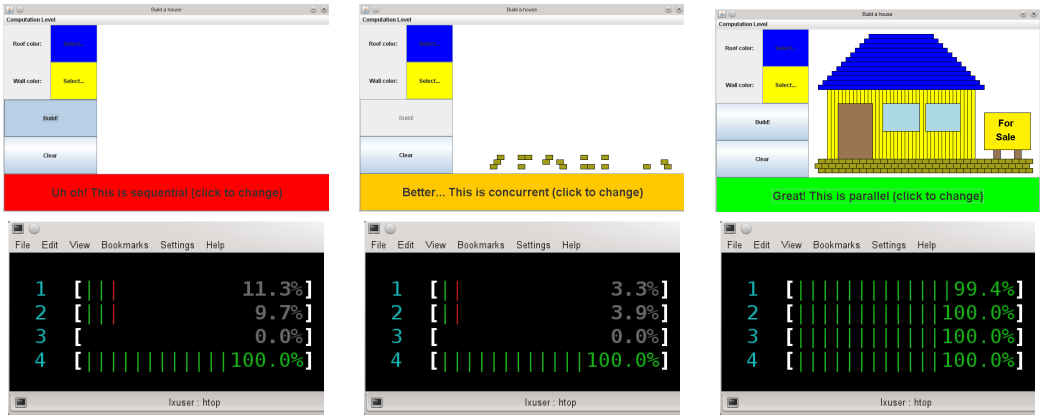


Figure 1: The Modern Parallel Programming Framework proposes the aggregated concepts that software developers require in order to apply parallel programming to mainstream application domains.

4. **Object-Oriented Parallelization:** finally, this component recognizes that traditional parallel computing applications were developed in speed-efficient languages such as C and Fortran. However, for parallelization to be embraced in modern parallel computing domains, it needs to be delivered in a form that promotes the high level of abstraction and programmer productivity.

Before expanding on these four components, Figure 2 presents our attention-grabbing introductory application (available for download [22]) used in the very first lesson to demonstrate these four components integrated together. The application is executed in three modes:

- The *Sequential* mode in Figure 2(a) highlights two points to students: the GUI freezes until the entire computation is completed (and the house appears), plus only one core of the CPU is utilized.
- The *Concurrent* mode in Figure 2(b) offers a slight improvement. The GUI no



(a) Sequential: 12093ms (b) Concurrent: 12279ms (c) Parallel: 4859ms

Figure 2: Illustrative house-building demonstration [22] used to summarize aspects of the Modern Parallel Programming Framework to students in the very first lesson. The application is executed in three modes, while also showing CPU utilization. The resulting speedup is only 2.5x (whereas 4x speedup might be "expected"), which is due to hyper-threading on the laptop's i7-3537U processor.

longer freezes since the house building appears in-progress (Component #3 of the MPPF). However, only one CPU core is utilized (and often takes slightly longer than the Sequential mode to complete). This illustrates the difference between perceived performance, versus actual wall-clock performance.

- The *Parallel* mode in Figure 2(c) presents the ultimate situation, where the GUI remains responsive and the entire computation is completed much faster with all CPU cores being utilized.

The goal of this short 5-minute demonstration promotes that parallel programming (Component #2 of the MPPF) is something they as "modern software developers" are responsible for should they want to develop great software solutions. The application is developed in Java, the very same object-oriented language they can relate to (Component #4 of the MPPF). As this is primarily intended as a classroom demonstration, the runtime of this demonstration is short. Although students still experience a factor speedup (depending on the number of available cores), this demonstration is modest in terms of the potential impact of parallel programming for larger problems. It is therefore worthwhile at this time discussing with students the potential of parallel

programming when applied to longer-executing computations. For example, achieving the same speedup factor for a 1-hour program.

3.1 Ubiquitous Parallel Computing Platforms

Most students arrive to the first lesson already aware of multi-core processors. Immediately when asked "who here has a multi-core laptop/smartphone/tablet?", most are quick to raise their hands. This immediately sets the scene for the context of the course, where students are excited when they realize the course targets parallel computing devices that are literally already at their fingertips. Rather than emphasizing distributed computing platforms, the course is concerned predominantly with shared-memory systems where the processors are within a single system. The ability to upgrade their own systems so easily again captivates students, and reinforces the relevance of parallel computing for their career. One reason why we emphasize on shared-memory systems, is to allow us to focus on the fundamental PDC concepts without the challenges associated with larger distributed systems (including access to them). Since students already have these devices, it also increases their motivation to engage in developing software for them and immediately experience the benefits of parallel programming.

3.2 Fundamental PDC Topics

This section discusses the particular topics as they relate to the PDC Curriculum Initiative, extending the high-level overview of SoftEng751's structure presented in Section 2. The topics presented in Table 3 denote those core fundamental topics presented in the PDC Curriculum Initiative, and are essentially unmodified regarding the recommended Bloom's classification [23]. While these topics are presented as being covered in four 2-hour lessons, it tends to total about 9-10 hours (so about $2\frac{1}{2}$ weeks) of teaching time to include sufficient opportunity for tutorials, examples and exercises. Most of the topics are selected from the *Programming* area, but students are first presented with topics from the *Architecture* area to shed light on the forms

of hardware in parallel computing. Focus remains on shared memory systems, as the course concentrates on parallelizing mainstream parallel computing devices (such as desktops and smartphones, which contain multi-core processors).

While these topics are traditionally fundamental to the PDC field, we present the concepts in light of modern programming languages and libraries. The general philosophy is that students should be able to explore the concepts immediately without requiring any specialized parallel hardware or software packages. To achieve this, the course concentrates on using standardized Java and C++ solutions. Most of the class exercises in this section include taking a sequential program and introducing the appropriate parallelization constructs from the respective object-oriented language. Unlike traditional parallel programming (where program performance is key), a point of difference here is that we prioritize programmer productivity (i.e. get good performance with minimal effort). This is vital, since simple solutions tend to be less error prone.

Table 4 presents another set of lessons and their topics, which tend to differ from the PDC Curriculum Initiative recommendations. While two of these topics are included in the Curriculum Initiative, we explore these to a higher level. This namely includes OpenMP and threading library support, since most modern C++ compilers incorporate OpenMP (e.g. Intel, GCC) as well as libraries with tasks and thread pools. As object-oriented languages dominate desktop and mobile application development, having an intimate level of understanding of these tools is essential for software developers.

Finally, there are two additional topics included in Table 4 that are not covered by the PDC Curriculum Initiative, but are believed to be valuable in developing modern software applications. Lesson #6 (GUI concurrency) will be explained in Section 3.3. Lesson #5 extends tasking, and distinguishes between the different types of tasks. This is in recognition that a given application might incorporate different forms of parallelism (*I/O-bound* tasks versus *CPU-intensive* tasks). For example, an image manipulation application may include both image manipulation/filters (CPU-intensive, making heavy use of the CPU) and image searching (not so CPU-intensive,

Lesson (each 2 hours)	Area	PDC CI Topic	Bloom #	Learning Outcome
1	Architecture	Taxonomy	C	Flynn's taxonomy, data vs control parallelism, shared/distributed memory
		SIMD/Vector	K	Describe uses of SIMD/Vector
		MIMD	K	Identify instances of MIMD (focus on multi-core)
		Simultaneous Multi-threading	K	Distinguish SMT from multi-core (demonstrated with the house building application)
		Multi-core	C	Describe how cores share resources (cache, memory)
		Shared (SMP) vs distributed memory (NUMA)	N	Understand shared memory access vs message passing
2	Programming	Task/thread spawning	A	Write correct programs with Java threads and Runnables
		Data parallel	A	Implement a parallel loop over a data structure containing elements to process
		Synchronization, critical regions	A	Protect critical sections using locks (fair/unfair), `synchronized` keyword, `AtomicInteger` from `java.util.concurrent`
		Deadlocks and data races	C	Understand what they are and how to prevent them
3	Programming	Performance metrics; speedup, efficiency	C	Understand how to compute, what they mean and why they are important
		Amdahl's Law and Gustafson's Law	K	Understand the limits of speedup and weak scaling
		Overhead and load balancing	C	Understand effects of load imbalances and ways to balance loads across threads
		Parallel loops for shared memory (dependencies)	A	Understand collision/dependencies across iterations
4	Algs	Scheduling; dependencies, task graphs	A	Observe how dependencies constrain execution order of sub-tasks
		Algorithmic paradigms; divide & conquer, recursion, reduction	C	Recognize opportunities for parallelism in algorithms such as mergesort

Table 3: Fundamental PDC topics largely from the PDC Curriculum Initiative. They are presented closely aligned to the recommendations in terms of Bloom's classification [23] (Know, Comprehend, Apply). The content is usually covered in about 8-9 hours worth of class time.

Lesson (each 2 hours)	Area	PDC CI Topic	Bloom #	Learning Outcome
5	Programming	Compiler directives; OpenMP	C → A	*Although the PDC CI suggests a Bloom's level of Comprehension, we extend to an Apply level* Understand what the directives mean, apply them to transform a given sequential loop to execute in parallel. Analyze the code to determine what data clauses should be applied, and suggest appropriate loop scheduling policies, including reduce.
		Libraries; thread pools	C → A	*Although the PDC CI suggests a Bloom's level of Comprehension, we extend to an Apply level* Know in detail the existence of libraries that support tasks vs threads, threadpools, recursive tasks, etc. Here we focus on the `java.util.concurrent` package, using Java's built-in `ExecutorService` and `ForkJoin`.
		I/O-bound tasks vs CPU-intensive tasks	→ A	*Currently not in the PDC Curriculum Initiative* Implement different threadpools depending on the nature of the task. For example, an I/O-bound task (e.g. internet search) involves large amounts of waiting, so should not preoccupy a thread from the threadpool dedicated to utilizing the CPU cores.
6	Programming	Concurrency in graphical user interfaces (GUI applications), and similar event-handling applications	→ A	*Currently not in the PDC Curriculum Initiative* This topic applies concurrency and parallelism in the context of GUI applications. Students implement GUI applications that adhere to GUI toolkit threading rules, for example using `SwingWorker`. This requires understanding the role of event-loops, event-queues and event handlers. Off-loading time-consuming computations to background threads to free the event handling thread, and maintain a responsive application that does not freeze

Table 4: Unlike the topics of Table 3, these topics are either taught to a higher Bloom's level or are extending the PDC Curriculum Initiative. Together, these topics are essential for developing the necessary hands-on experience required to parallelize everyday object-oriented desktop and mobile applications.

mostly spent waiting for results). Both the above functionalities have potential to benefit from parallel programming, but need to be treated differently. In the case of performing an I/O-bound task (such as the image search), we do not want to schedule such tasks on the same thread pool. If we do, then that thread is waiting (not performing any work) and our processor core is effectively idle. We show the solutions for these, by using the different forms of thread pools one might find in modern libraries. For example, Java's `ExecutorService` implementation includes a `CachedThreadPool` for I/O-bound tasks, while the `FixedThreadPool` is ideal for CPU-intensive tasks.

3.3 GUI Concurrency

Table 4 mentioned the GUI concurrency topic that is missing from the PDC Curriculum Initiative. To assist instructors, this section elaborates on the fundamental aspects pertaining to this topic. As such, this section is written more as a background conveying the underlying concepts requiring attention. Section 4.2 overviews the **JPDC_GUI_Concurrency** exercise that may be used with ACP, as an example to use in the classroom.

The most distinguishing aspect of desktop and mobile applications is the GUI that allows users to interact with the application. These visual components of the interface comprise of graphical components that serve as both input (e.g. buttons, text fields) and output (e.g. progress bars, message dialogs). Rather than developing these components from scratch, developers use GUI toolkits that provide them.

Figure 3 illustrates *inversion of control*, a distinguishing aspect of desktop applications, where the flow of control is dictated by a framework rather than the application code. In such applications, the GUI toolkit also provides a framework that facilitates the communication with users. *Events* are generated from within the framework code, and developers implement routines, known as *event handlers*, in the application code that responds to the respective events. The control returns back to the framework upon completion of the respective event handler, namely to the *event loop*. This is

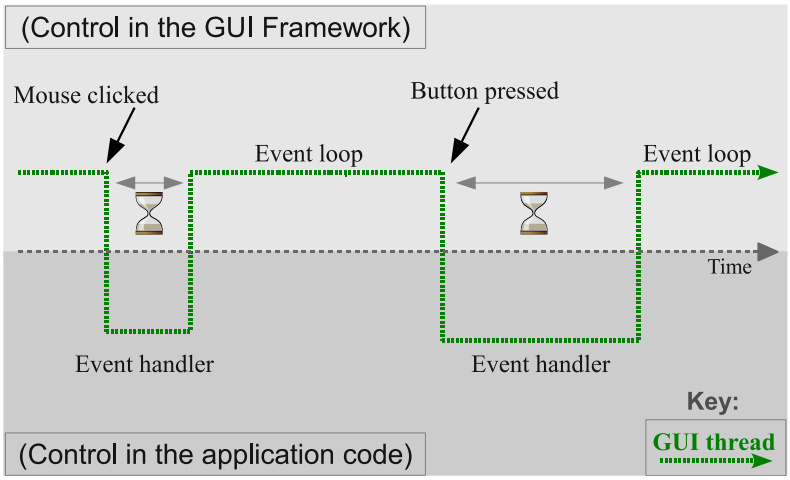


Figure 3: The control flow in a sequential GUI application, where the GUI thread responds to events by executing the respective event handlers. If the handling takes a noticeable amount of time, this delay is perceived by the users as an unresponsive application (the GUI appears to "freeze"). Figure 4 presents a multi-threaded version to allow responsiveness.

very different from batch-type programs without a GUI (where parallel programming is traditionally applied), since now all the control flow is determined by the user rather than the programmer.

In GUI frameworks, the event handling is performed by a dedicated thread named the *GUI thread*. An important aspect of mainstream desktop and mobile applications is their responsiveness to user interactions. To maintain an interactive and responsive application, programmers must ensure the GUI thread minimizes its execution in the application code, therefore remaining largely in the event loop to respond to other potential events. This is ensured by keeping event handlers as short as possible, as illustrated in Figure 4. In the case that a time-consuming computation needs to be processed in the event handler, the computation is off-loaded to a helper thread (Figure 4(A)). This allows the GUI thread to return to the event loop and respond to other events, for example Figure 4(B).

Now that time-consuming code is executed concurrently by helper threads, this leaves the GUI thread available to respond to other events and hence keeping the entire application responsive. Eventually, when the computation completes, the GUI

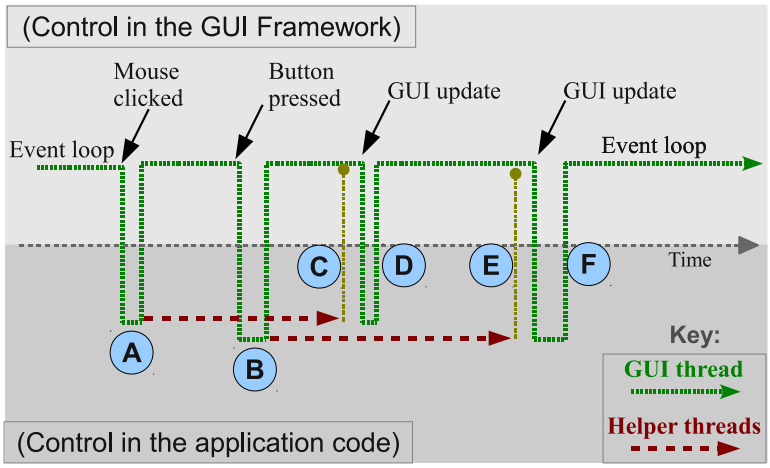


Figure 4: The control flow in multi-threaded GUI application, where the time-consuming computations are off-loaded to background threads to allow the GUI thread to immediately return to the event loop.

may need to be updated to notify the user. In most graphical toolkits, only the GUI thread may access the GUI components. This means that helper threads must request GUI updates by posting an event to the GUI thread (Figures 4(C) and (E)). The GUI thread picks up the event and invokes the respective code to update the GUI (Figures 4(D) and (F)). The reasoning behind this is because GUI toolkits are not thread-safe and must therefore only be accessed by a single and dedicated thread (i.e. the GUI thread).

An important aspect of Figure 4 is that multi-threading has been introduced solely for the purpose of achieving *concurrency* just to maintain a responsive application. In other words, the performance improvement is solely for user perception and is likely to even increase the overall computation time due to the introduced overhead. In order to truly benefit from multi-core processors, students need to extend the concurrency concepts further and introduce parallelism using the fundamental PDC topics mentioned earlier.

3.4 Object-Oriented Parallelization

The overarching theme behind the object-oriented paradigm is that of promoting programmer productivity. The benefits of high-level design considerations, such as encapsulation, code readability and code reuse overpower any potential performance penalties. These advantages have seen object-oriented languages dominate modern software development [24, 25], as evidenced in the popularity of Java (for Android and desktop applications), C# (for enterprise applications using the .NET Framework) and C++. For parallel programming to be embraced by this mainstream software development community, parallel tools and libraries need to be in the spirit of the object-oriented paradigm [8]. Looking at PDC topics in this light is also important in encouraging PDC integration into the wider Computer Science and Software Engineering curriculum. For example, the Computer Science Curricula 2013 [26] has dedicated a "Parallel and Distributed Computing (PD)" knowledge area, as well as incorporating parallelism aspects in the "Systems Fundamentals (SF)" knowledge area.

4 Active Classroom Programmer

This section discusses the integration of Active Classroom Programmer[1] (ACP) in SoftEng751. ACP is based on a collegial model of active learning for programming classrooms, by encouraging an open-learning environment. In addition to promoting good learning practices, it also seeks to promote a sense of belonging among students. It builds on sound industry practices, namely pair programming [27, 28], where each and every student is (loosely) paired up to program *with* the instructor. The guidance provided by the instructor in the problem-solving [10], as well as the debugging training, helps develop the cognitive skills necessary for programming. This is especially valuable for parallel programming, since debugging in front of students is highly beneficial to illustrate parallelization caveat concepts such as race conditions. An in-depth discussion on the ACP model and its background have been presented before [16]; this section therefore concentrates on its integration with a PDC course

[1] `https://acp.foe.auckland.ac.nz/`

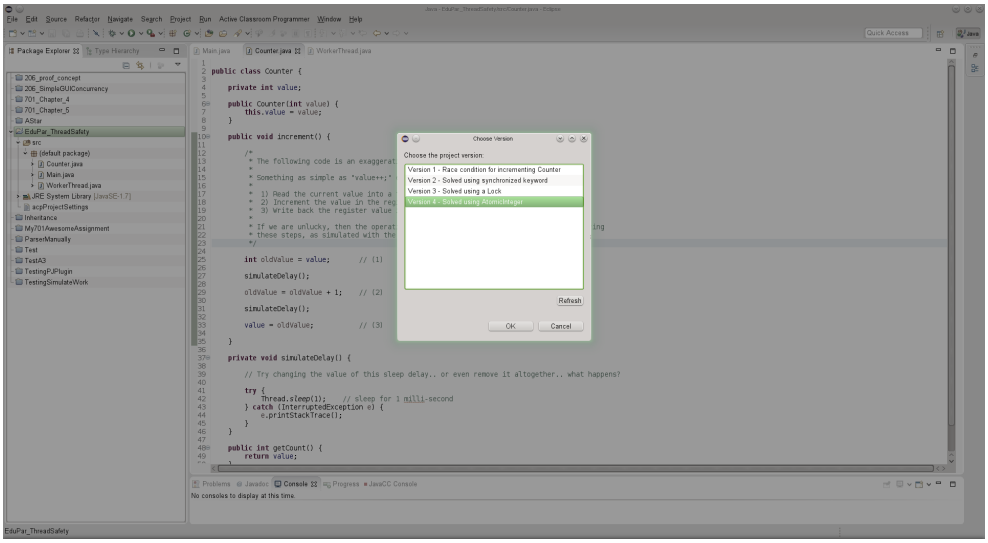


Figure 5: The ACP plugin running in Eclipse. The existence of the plugin is non-distracting. Interactions include right-clicking on an existing project (from the left menu) to sync between versions, or to download a new project altogether (from the Active Classroom Programmer menu at the top). Here, the user is using the *Sync project* feature to select one of the solutions for this simple race condition exercise.

such as SoftEng751.

4.1 ACP Workflow and Features

The idea behind ACP is that the instructor creates an exercise (either during the lesson or beforehand), which is then uploaded to the ACP server. The student will then access that exercise by downloading it to their own Eclipse IDE environment and complete snippets of it during the lesson (as well as after lessons).

ACP distinguishes students from instructors, since the permitted actions differ between these roles. The instructor is able to upload a project inside the Eclipse environment to be shared with the students, by simply right clicking on a project and selecting the "Upload Project" option. The project is immediately uploaded to the server, including any libraries contained within the project directory. A user (either student or instructor) may then download the project via the ACP menu.

Instructors are allowed to upload new versions of an existing project for the students (where each version can be given a meaningful tag). The goal behind this *sync*

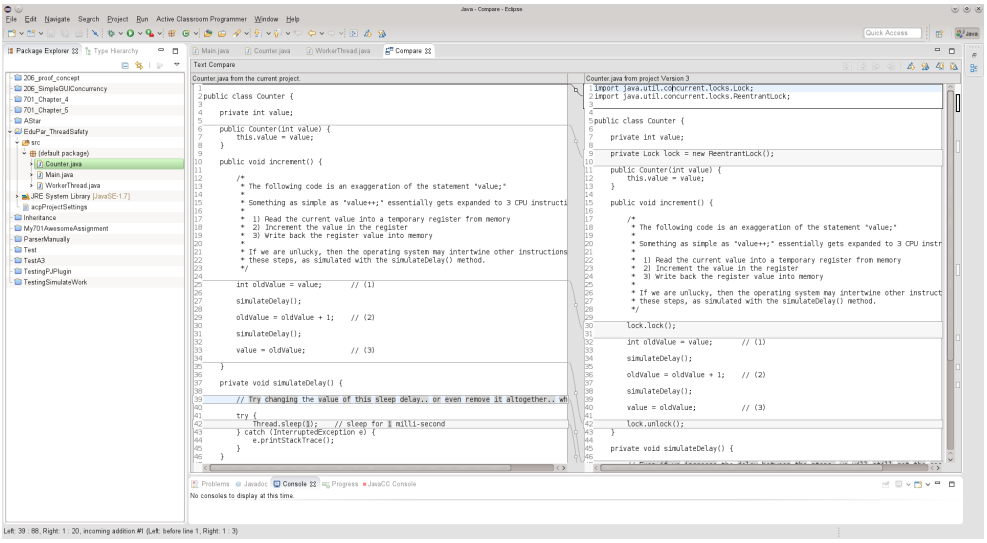


Figure 6: Here the user has selected to view the differences between two snapshots of a project. In this example, the current version is Version 1 on the left (with the race condition), while Version 3 (solved using a Lock) is on the right.

feature is for the instructor to progress through an exercise all the while uploading different versions of the same project. The students would then be able to progress through an exercise effortlessly by switching between the different versions of the project (Figure 5). In the context of classroom usage, the students will sync to the latest version of the project as the instructor adds a new version. This is similar to switching between different revisions in a version control software, except it has been made very simple through ACP.

Since each project contains multiple snapshots (i.e. the sync uploads made by the instructor), an *ACP compare version* feature is integrated so that students can see modifications of source code compared to another version of that same project. Figure 6 shows a version compare of file `Counter.java`, showing how a Lock was used to solve the race condition.

4.2 Example PDC Exercises

This section presents a selection of example applications that demonstrate some of the topics discussed in Section 3. Anyone may run these examples by installing[2] the ACP Eclipse plugin, registering a user account[3] and finally running the examples[4]. When analyzing the code of the examples, the *ACP compare version* discussed in Section 4.1 helps in pinpointing the code modifications between versions.

JPDC_Multi_Threading_Introduction

This first example illustrates the most basic and fundamental concepts underpinning parallel programming. It consists of various progressions that build from the original sequential code shown below:

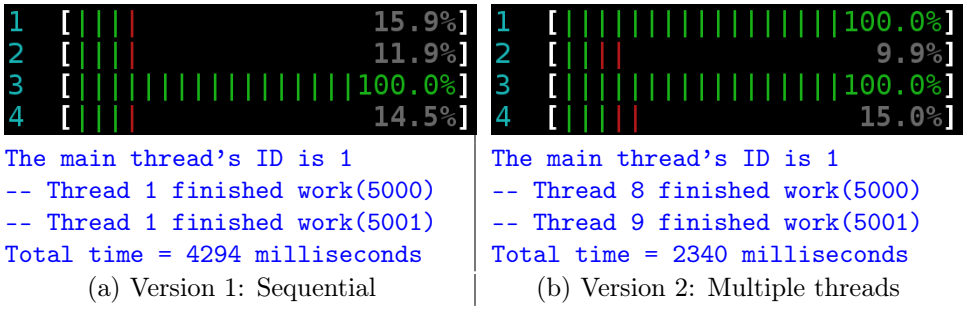
```
public static void main(String[] args) {
  System.out.println("The main thread is " + Thread.currentThread().getId());
  long startTime = System.currentTimeMillis();

  Work.doWork(5000); //-- do the first computation
  Work.doWork(5001); //-- do the second computation

  long totalTime = System.currentTimeMillis() - startTime;
  System.out.println("Total time = " + totalTime + " milliseconds");
}
```

The computational task is represented by the `Work.doWork()` invocation, which prints the ID of the thread that executes it. Figure 7(a) shows the execution of the above sequential code, where both computational tasks are executed by the same main thread. Figure 7(b) represents output of the next progression of this exercise, which involved introducing separate threads to execute each of the `Work` invocations. This ACP project is extended with five versions (progressions), such as demonstrating differences between daemon and user threads.

[2] `https://acp.foe.auckland.ac.nz/install-instructions`
[3] `https://acp.foe.auckland.ac.nz/register`, then use `jpdc` as the passcode
[4] `https://acp.foe.auckland.ac.nz/instructions`



(a) Version 1: Sequential (b) Version 2: Multiple threads

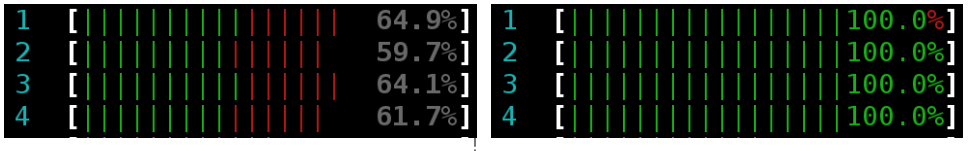
Figure 7: CPU utilization and output printed to the console when executing the (a) sequential version of the *JPDC_Multi_Threading-Introduction* project versus the (b) parallel version.

JPDC_Thread_Safety

This project consists of a simple class called `Counter` with an internal value that may be incremented. If the `increment()` function is invoked by multiple threads simultaneously, this leads to a possible error in its value. The versions of this project progress with an incorrect `Counter` implementation that "works 99.99% of the time". However, the next versions involve inserting a minor delay of one millisecond between the simulated update of the increment, which almost always results in incorrect output. The final versions of the project involve using various approaches to correct the problem, for example using Java's `synchronized` keyword, `AtomicInteger` or `Lock`.

JPDC_Tasking

The goal of this project is to illustrate the overhead incurred in creating too many threads. This is achieved by first presenting some sequential code in Version 1, consisting of a large number of extremely fine-grained computations. The total execution of this sequential code is around 21 seconds. Version 2 follows the same procedure of *JPDC_Multi_Threading-Introduction* by creating a new Thread for each computation. Despite the theoretical parallelism expressed with such a large number of threads, Figure 8(a) shows the poor CPU utilization due to excessive context switching. Not only that, but also the parallel version requires 58 seconds to complete (much worse than the sequential version). This naturally introduces the notion of thread



(a) Version 2: Thread per computation (b) Version 3: Lightweight tasking

Figure 8: CPU utilization for two parallel versions of the *JPDC_ Tasking* project, using (a) a Thread for each computation results in excessive context switching versus (b) more efficient performance with lightweight tasking by aggregating the computations to a smaller thread pool.

pools, where Figure 8(b) represents a better CPU utilization with the parallel version now requiring only 11 seconds.

JPDC_GUI_Concurrency

This project illustrates the difference between concurrency and parallelism in the context of a GUI application (as discussed in Section 3.3). The first versions of the project show a non-threaded GUI application that "freezes" since concurrency is not implemented. After correcting this for the simple one-off computation, the application is extended to incorporate multiple computations. This allows us to demonstrate the importance of intermittent results in a GUI application (i.e. the concurrency aspect), as well as extending to demonstrate how parallelism can also be integrated.

5 Graduate Qualities Beyond Technical Skills

Section 3 discussed *what* topics were relevant to a software developer's career, while Section 4 discussed *how* some of those technical skills can be promoted in a practical learning environment. This section takes a step back, and asks a higher-level question: *why* do students undertake higher education study? This is then closely followed by what industry wants from graduates, and how SoftEng751 targets those expectations using a research-infused [18] and problem-based learning approach.

Rated	Reasons to go to college	Importance
1st	To improve my employment opportunities	91%
2nd	To make more money	90%
3rd	To get a good job	89%
4th	To learn more about a favorite topic	85%

Table 5: From a list of 12 reasons why students undertake higher education study, the top 3 important reasons involved favoring job conditions [29]. While a solid curriculum is pedagogically essential in structuring the topics, students only rate this as the 4^{th} most important reason to undertake higher education study.

5.1 Why Students Study and What Industry Wants

The PDC Curriculum Initiative [3] highlights the most essential PDC technical skills. While a curriculum is useful in identifying *what* are the important topics to teach students, an even more important question to answer is: *why are students undertaking higher education in the first place?* A survey by New America's Education Policy Program shows no surprises that students and their families see higher education as an investment [29]. The top reasons that participants signaled going to college are presented in Table 5, where students rate more favorable employment conditions as motivating them to undertake higher education study.

There is no doubt that having technical knowledge is an essential trait that employers seek in graduates. But as the software development industry leans towards larger scale software systems developed in an agile approach, personal traits become evermore important [17]. Numerous interpersonal traits, or *soft skills*, are sought by employers [30]. In ensuring that their graduates are employable, universities need to ensure that their graduates develop wider skills beyond their immediate job description [31]. A survey of employer hiring intentions showed that teamwork and problem solving skills are the highest rated abilities that employers seek in graduates [32]. Table 6 highlights these qualities, showing that technical skills are only rated 7^{th}. Employers are seeking qualities that are essential for team-based projects, so we should not neglect the development of the student's soft skills in the PDC domain.

Such soft skills, also known as *professional skills*, cannot be taught in the tradi-

Rated	Skill/quality	Weighted average
1st	Ability to work in a team structure	4.61
2nd	Ability to make decisions and solve problems	4.61
3rd	Ability to verbally communicate	4.60
4th	Ability to plan, organize, and prioritize work	4.59
5th	Ability to obtain and process information	4.57
6th	Ability to analyze quantitative data	4.32
7th	Technical knowledge related to the job	4.19
9th	Ability to create and/or edit written reports	3.75

Table 6: Skills and qualities that employers rate as most important in candidates, using a 5-point scale (1=Not at all important, 2=Not very important, 3=Somewhat important, 4=Very important, 5=Extremely important) [32]. As PDC educators we should try to nurture more than just PDC topic knowledge and technical skills, as this is only rated 7^{th} most important by employers.

tional lecture format [33]. One of the best ways to develop these skills that employers seek involves giving students the opportunity to engage in meaningful group projects [34, 35]. For this reason, SoftEng751 students engage in projects that span half the semester, in teams of three students. These projects incorporate elements of research-infused learning (Section 5.2) and project-based learning (Section 5.3). Unlike short-cycle problem solving, the merging of these concepts better reflects the work expected by students in industry [36].

The group project work comprises 60% of the course assessment, and involves three equally weighted components: (i) group presentation, (ii) written report, and (iii) code implementation. As the presentations tend to span multiple weeks, different groups would be at different stages when they present. Therefore, presentations do not assess progress in the project, but rather how well the team articulates understanding of the specific topic and communicates this back to their peers. The report and code are submitted at the end of the semester, and are assessed on written communication and depth of investigation. Student perspectives on how well the group projects address the skills sought in Table 6 is presented in Section 6.2.

5.2 Research-Infused Learning

In order to foster the development of some of these non-technical qualities that employers seek, SoftEng751 promotes critical and reflective thinking in students using a research-infused learning environment [18]. With an ever-changing technical world, students are constantly expected to be ever so more flexible in their acquisition and analysis of the changing skill-sets; in this regard, developing a research-skilled undergraduate student is essential for their career, and not merely just to quench their desire for intellectual insight [20]. The level of research integration within the teaching can vary significantly [37]; in *research-led* teaching, students are mere recipients of research (e.g. the instructor's research is referred to within lectures or supplementary reading), while in *research-based* teaching the students take on the active role of advanced learning alongside the instructors by participating in inquiry-based learning. For completeness (but of less interest to us here), *research-oriented* teaching focuses on research ethos (as opposed focusing on the actual research content) [37]. Figure 9 shows this research-teaching nexus model, extended to include *research-tutored* teaching [38], where students work in groups writing and discussing papers.

Most of the projects focus on shared memory parallel systems, particularly for desktop or mobile systems. Students are provided with a paragraph description of each topic, allowing them to understand the main points underlying the project. In some cases, students may implement the application as a standard Java desktop application, or as an Android app. In all cases, students are reminded that they will be assessed on the parallelization aspects rather than the (general) app aspects – which is necessary for when Android apps are permitted. Examples of projects include generating thumbnails of images in a folder, efficient PDF file searching, developing educational PDC visualizations, or integrating API libraries (such as Flickr, Google Maps, etc) in a parallel context.

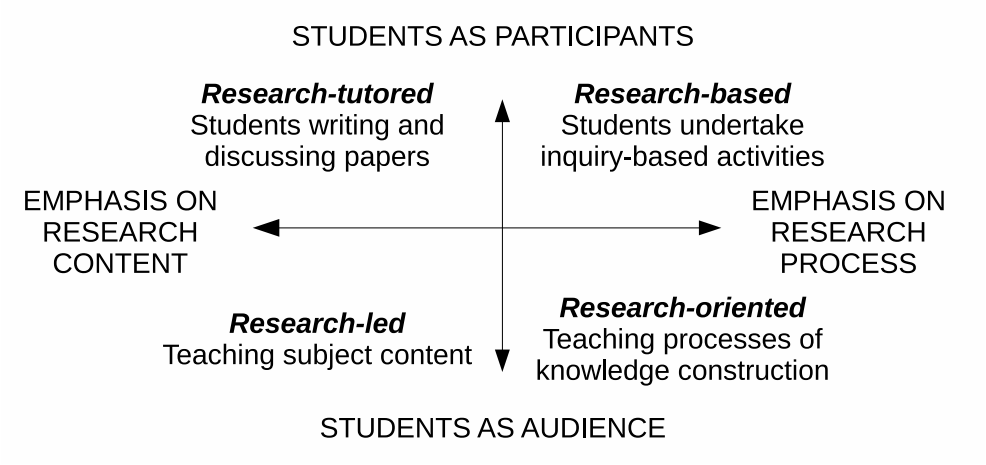


Figure 9: The research-teaching nexus and how it integrates with curriculum design based on content emphasis and student participation in the research (based on Healey [38]).

5.3 Problem-Based Learning

In addition to the research-infused approach, SoftEng751 also incorporates elements found in problem-based learning (PBL) [19, 39]. This student-centered pedagogy is typically used in medical education, but has been applied to a wide variety of education domains [19]. As SoftEng751 needs to primarily concentrate on PDC, only low-maintenance aspects of PBL are embedded within the course. A fully-embedded PBL approach would be burdening and detracts from the core focus of the PDC course, imposing strain on human resources as well as information overload for students [39]. The core characteristics of PBL, and those promoted in the course, include:

- The projects are problem focused, using authentic and ill-structured problems provided as *triggers* (Section 5.2 briefly listed examples, but example project definitions given to students follows). This is a characteristic of PBL to ensure a constructivist approach to learning and the development of generic skills useful for their career [19].

- While the course provides fundamental PDC content, projects allow students to master an advanced PDC topic using a self-directed learning approach. This

promotes responsibility and independence in student learning [39].

- Group learning is essential in promoting teamwork, respect for others, communication skills [39].

Below is an example of a typical project topic. Following PBL ideas, this paragraph is essentially the trigger provided to students:

> *Priority queues (PQ) are an important data structure. Due to their importance, the Java SDK provides PQ implementations (inside java.util). Often PQs are implemented as binary heaps, which are essentially binary trees implements in a flat array. There is also a thread safe implementation in the java.util.concurrent package (PriorityBlockingQueue), but its performance under the concurrent access of multiple threads is limited. Investigate which better thread-safe priority queues have been proposed (e.g. Pipelined PQ, lock-free PQ) and implement at least one such version. Your code can be based on the (Open Source) code of the existing (Priority)BlockingQueue. Compare the performance of your queue with PriorityBlockingQueue in a simple application benchmark.*

As another example, the following topic is more closely related to research tools developed by the instructors (in this case "ParaTask"):

> *Parallel execution environments like ParaTask normally manage the tasks to be executed in one or multiple queues. For dynamic load balancing, these environments then often employ work stealing approaches, where a worker thread without tasks steals from the queues of busy worker threads. Investigate the different work stealing approaches and algorithms in literature. Important distinguishing points are: queue structures and organization, victim selection, stolen chunk sizes etc. Implement your own work stealing algorithms in the ParaTask runtime and compare its performance to the built in algorithms using simple application benchmarks.*

6 Evaluation

The evaluations discussed below are selected samples from 2-3 years of SoftEng751 offerings in which the Modern Parallel Programming Framework has been integrated in the course. This section sheds some light by correlating performance in the course assessments, presenting student perceptions, and understanding ACP usage patterns.

6.1 Correlating Course Assessments

Assessments in the SoftEng751 course are decomposed into five equally-weighted components (Table 2). The first assessment is the mid-term test (T1) that marks the end of the instructor-led teaching. As the focus of the course turns to the group projects, the next assessment comes in the form of each group's presentation (P). The final test (T2) concludes the presentations, by assessing student's grasping of concepts covered during the student presentations. At the conclusion of the semester, groups submit the report (R) and code implementation (C) for their project. By considering the previous two offerings of the course, Table 7 presents the Pearson correlation coefficient for all combinations of the course components. Unsurprisingly all correlations were positive (i.e. above zero), but some were stronger than others. Overall, the 2016 offering tended to have slightly stronger correlations than 2015 (average of 0.434 versus 0.363 respectively). To the instructors best knowledge, there were no notable differences in the offerings of the course between those years; such variances may simply be due to the nature of a problem-based project course, and the manner in which groups functioned as a team.

In both years, the largest strength of correlation was that between the report (R) and code (C) assessments. This is understandable, as both of these assessments were group submissions at the conclusion of the semester. The next-strongest correlations were between the individually assessed tests (T1 and T2), with a medium to large strength of association over the two years. When looking at the other components, slightly different correlations occur from year to year. The existence of the weak correlations, existing in both years, supports the claim that a broader range and

	Individual		Group			Individual		Group		
	T1	**T2**	**P**	**R**	**C**	**T1**	**T2**	**P**	**R**	**C**
T1		0.478	0.387	0.221	0.380		0.675	0.401	0.289	0.387
T2	0.478		0.299	0.052	0.204	0.675		0.385	0.328	0.406
P	0.387	0.299		0.483	0.483	0.401	0.385		0.231	0.426
R	0.221	0.052	0.483		0.646	0.289	0.328	0.231		0.812
C	0.380	0.204	0.483	0.646		0.387	0.406	0.426	0.812	
	(a) 2015, N=99					(b) 2016, N=90				

Table 7: Pearson correlation coefficient for the various assessments in the course: individual Tests (T1 and T2) and group Presentation, Report, Code. (a) In 2015, the strongest correlations were between the group components, particularly the joint report and code submission, and the individual test performances. (b) In 2016, the strongest correlations were again between the group's joint report and code submission, and the individual test performances. However, in both years, there existed weak correlations between some of the other assessments, suggesting the importance of assessing student skills in a number of ways, such as individual versus group work, and written versus oral communication.

more comprehensive set of assessments (as existing in the course) are valuable in holistically developing students beyond mastery of knowledge and technical skills. Even T1 and T2, although they are both individually assessed tests, they still assess different skills (content knowledge from instructors versus conceptual understanding from peers).

The weakest correlation in 2016 was that of the report compared to the presentation; while both of these assessments were collectively undertaken within the groups, this emphasizes the different communication skills demanded in presentations (verbal) versus reports (written). The final test correlations in 2015 were notably weak when compared to the group project components (especially the written report). This emphasizes the differences of understanding a broader range of problem-based PDC issues (as assessed in T2), versus the deeper synthesis and evaluation expected from problem-based investigations as demonstrated through the report.

The remainder of the correlations are rather medium when comparing performance in individual tests to performance in group assessments. This is attributed to the (self-selected) groups naturally containing students of different strengths. These differences in correlation show that integrating such different assessments (group work, individual work, written communication, oral communication) is necessary in developing the

wider variety of skills needed by industry. If only individual and technical assessments are enforced, this does not provide students the opportunity to develop team-based and communication skills.

6.2 Student Perceptions on Problem-Based Learning

Taking into consideration the non-traditional teaching approach, particularly the focus on developing soft skills, it is valuable to see how students perceived this approach. An anonymous end-of-semester self-reported questionnaire was completed by the latest offering of the course, where 69 students responded. The questionnaire focused on four sections: (i) attitudes to student-led presentations, (ii) perceived benefits of giving a presentation, (iii) perceived benefits as an audience member, and (iv) benefits of the group project work. Table 8 summarizes the results, where students tended to have a positive attitude towards the student-led presentations (Q1-Q5). Students overwhelmingly agreed that giving a presentation was a valuable experience (Q6-Q9), while being an audience member was also fairly appreciated (Q10-Q13). Finally, students were generally supportive of the group work and its relevance to their career (Q14-Q18). In particular, Q19 outlines the students perception of developing the top five skills and qualities that employers rate as most important (Table 6 from Section 5.1).

Students were also asked open-ended questions to provide deeper insight. When asked *"How did the student-led presentations add to your knowledge of parallel computing content?"*, most responses complimented the depth and breadth of PDC knowledge that presentations encouraged. Even when some topics were covered by multiple groups, there was appreciation for different solutions to the same problems:

- *"The fundamental lectures aimed at the breadth of parallel computing. While the student-led presentations focused on some of the topics, extra knowledge from presentations could be gained, especially by preparing and presenting our own topic significantly enriched my knowledge on that specific aspect"*
- *"We get a wide range of research, opinions, and ideas. Although some topics*

did have presentations that overlapped in content, there was a lot less overlap than I expected. Just goes to show the wide range of difficulties and solutions in parallel computing"

- *"It was good to know different ways people would think of implementing a specific application. Since we had 3 groups per topic, it was great to see a variety of creativity and ideas"*
- *"Different groups took different approaches in solving the same problems which was very interesting"*

	SA (5)	A (4)	N (3)	D (2)	SD (1)	$\bar{x}$	s
Attitudes to student-led presentations							
Q1. I attended student-led presentations as much as I could	52%	26%	16%	3%	3%	*4.2*	*1.01*
Q2. Separation of the course, into traditional lectures for fundamental parallel programming content and then student-led presentations worked well for this course	23%	40%	20%	13%	4%	*3.6*	*1.11*
See bottom of table for Questions 3 to 5 from this questionnaire section							
Perceived benefits of giving a presentation							
Q6. Giving a presentation was good practice to develop confidence presenting to an audience	67%	22%	4%	4%	3%	*4.4*	*0.97*
Q7. Giving a presentation was a good opportunity to practice my oral communication	67%	24%	1%	4%	3%	*4.5*	*0.95*
Q8. Giving a presentation will help me transfer communication skills from the classroom to the workplace	52%	30%	6%	7%	4%	*4.2*	*1.12*
Q9. The question-handling component of presentations is helpful for my professional training	54%	30%	12%	1%	3%	*4.3*	*0.94*
Q10. As an audience member, observing my peers present was useful in providing me with insights to improve my own presentation skills in the future	54%	30%	12%	1%	3%	*4.1*	*0.98*
Q11. In observing my peers, I sometimes noticed learning gaps or mistakes in their presentation styles that in turn will help me improve my future communication effectiveness	39%	45%	14%	0%	1%	*4.2*	*0.80*
Q12. Having all presentations of the same topic in the same lecture helped reinforce concepts, especially seeing how different groups approached the same problem differently	54%	26%	6%	10%	4%	*4.1*	*1.18*
Q13. The broad range of advanced topics over the past few weeks makes it interesting	33%	39%	17%	6%	4%	*3.9*	*1.07*
Perceived benefits of group project work							
Q14. The problem-based projects help in putting the parallel programming concepts into context applicable to a software engineering career	39%	49%	6%	3%	3%	*4.2*	*0.90*
Q15. I believe that project work and presentations help to develop important qualities beyond technical skills	51%	41%	6%	0%	3%	*4.4*	*0.84*
Q16. Working in groups in an open-ended problem helped critical thinking, by considering different ideas and thinking reflectively	51%	41%	4%	1%	3%	*4.3*	*0.87*
Q17. The group-based project helps develop useful teamwork skills for my career	45%	42%	7%	3%	3%	*4.2*	*0.93*
Q18. I believe that participating in projects and presentations helps improve my employment opportunities	38%	38%	16%	4%	4%	*4.0*	*1.06*
See bottom of table for Question 19 from this questionnaire section							

	(i)	(ii)	(iii)	(iv)
Q3. If I was initially against the idea of student-led presentations, it was because (select all that apply): *(i) Extra work required (ii) Fear of public speaking (iii) Boredom watching my peers present (iv) Not applicable, I was never against presentations [Exclusive option]*	19%	26%	28%	43%

	(i)	(ii)	(iii)	(iv)
Q4. Did your initial attitude of doing a presentation change as the presentations were underway? *(i) Yes, I was initially against but became supportive (ii) Yes, I was initially supportive but became against (iii) No, I always against (iv) No, I always supportive*	17%	16%	22%	45%

	(i)	(ii)	(iii)
Q5. Despite any initial resistance I felt towards the presentations, I eventually valued them because (select all that apply): *(i) Positive outcomes in myself presenting (ii) Positive outcomes watching others present (iii) No positive outcomes at all [Exclusive option]*	80%	46%	9%

	(i)	(ii)	(iii)	(iv)	(v)	(vi)
Q19. Undertaking the project within this course has improved the following of my abilities to (select all that apply): *(i) Work in team structure (ii) Make decisions and solve problems (iii) Verbally communicate (iv) Plan, organize, prioritize work (v) Obtain and process info (vi) None at all [Exclusive option]*	80%	65%	80%	67%	52%	4%

Table 8: Multi-choice responses received from 69 students in the anonymous questionnaire.

6.3 Student Perceptions of ACP

This section discusses feedback from students about ACP, selected from course evaluations over the past few years. The primary focus discussed here is on student perception in the guidance they receive on the programming strategy, in addition to their impression of the overall effectiveness of helping them grasp particular concepts.

Student Satisfaction of ACP: Mixing Theory and Practice

It was encouraging to see that students embraced the ACP learning environment promoted in the course. For the open-ended question *"What was most helpful for your learning?"*, over half the students referred to ACP as being helpful in the course. The following example comments demonstrate that students strongly appreciate a teaching model that integrates both theory and practice:

- *"The way of using ACP and showing demos to explain a concept helped me in understanding the concept more easily"*
- *"ACP allows for better understanding of theory that was taught. Exercises are really good, especially step by step versioning is helpful"*

Guidance on Programming Strategy

One of the aims of using ACP is that students are able to develop their cognitive programming skills by learning from the instructor good programming strategies that is difficult to learn on their own. When asked *"What was most helpful for your learning?"*, the following statement emphasizes the power of programming in front of students by promoting inductive learning:

> *"The practical live examples of using Eclipse/Android in class, as well as the kinds of explanations and discussions held in class. Errors and mistakes made were helpful in the sense that we can not only see how they were caused, but the thought process behind the coding that caused the error, the consequence, and the workaround/fix to correct it"*

This is especially important in learning parallel programming, since debugging is a vital skill when it comes to ensuring thread-safe and efficient parallel code. Many exercises can be developed in conjunction with students, during the lesson, where students see the value of debugging and how it could be done. Another student highlights that instructors should not underestimate the appreciation students feel for the programming guidance when programming in front of them:

> *"I enjoyed the coding sessions in class. It is really helpful because a lot of the content we learn is only helpful when we actually code it ourselves but seeing the lecturer code in class gives us a better understanding of what you are talking about and how to actually implement it"*

Grasping of Course Material

When asked *"In what way, if any, do you feel ACP assisted your learning?"*, many students emphasis how ACP allowed them to better understand the concepts by being active:

- *"Being able to run through examples in my own time outside lectures so I can grasp/understand concepts better"*
- *"Looking through parts and fiddle/modify the code to see differences and understand what happened"*
- *"It helped me understand theory through a practical approach"*

The versioning feature of ACP became a form of scaffolding that assisted the grasping of concepts by allowing students to gradually progress through the project code in their own time:

> *"Better understanding of theory that was taught. Exercise examples are really good, especially step by step versioning is helpful"*

Level	Degree	Enrollment count	GPA cutoff (/9)
Undergraduate	BE-SE	50	6.5
Undergraduate	BE-CSE	14	4.0
Postgraduate	MESTU-SE	19	3.5*
Postgraduate	MESTU-CSE	5	3.5*

Table 9: Distribution of SoftEng751 students based on degree being studied. *Since the Masters-taught students completed their undergraduate studies at overseas universities, a Grade Point *Equivalent* (GPE) is used rather than GPA.

ACP Critiques

Students were also asked for comments about improvements in how ACP could be better, or used better. Less than 20% of the students wrote comments for improvements, suggesting that most were happy with the way it currently works and how it was incorporated into the lectures. Half the comments written were actually non-informative (e.g. "not applicable" or "works just fine"). The only other critiques were requests for better WiFi in the lecture hall, followed by providing more time for students to undertake the in-class exercises.

6.4 ACP Usage and Learning Patterns

As briefly discussed in Section 2, SoftEng751 has a mixture of undergraduate students as well as postgraduate students. Table 9 shows the distribution in the latest course offering, with details of enrollment numbers and their incoming Grade Point Average (GPA) cutoff. The undergraduate students are either studying Bachelor of Engineering - Software Engineering (BE-SE) or Bachelor of Engineering - Computer Systems Engineering (BE-CSE), while postgraduate students are either studying Masters of Engineering Studies - Software Engineering (MESTU-SE), or Masters of Engineering Studies - Computer Systems Engineering (MESTU-CSE). In this offering, it was not expected that students bring their laptops to the lessons; although the ACP exercises were generated during the lessons in front of students, there was no dedicated in-class time for students to progress the exercises. Instead, students were encouraged to attempt the exercises in their own time.

We focus only on the Software Engineering students (BE-SE and MESTU-SE), since the CSE enrollment numbers are noticeably lower. Figure 10 shows the spread of ACP usage for the most-representative cohorts (BE-SE and MESTU-SE) over half a semester, which is up to and including the day of the mid-term test. Most apparent is that the 19 MESTU-SE students were more active in running the ACP exercises outside of class time; these students are predominantly international students that have completed their undergraduate studies in their home country. Despite the BE-SE students having a considerably higher GPA cutoff (Table 9), 30% never executed any ACP project (and another 20% only used ACP for one day). It is therefore assumed that students treated ACP as an additional form of learning support only worthwhile when the concepts presented during the lectures were not fully understood. The difference is statistically significant, with an unpaired two sample t-test revealing a two-tailed $P(\bar{x}_{MESTU} \neq \bar{x}_{BE}) = 0.0018$.

Unfortunately there was no pre-test to help evaluate the effectiveness of ACP to help students learn the parallel programming concepts. However, Figure 11 attempts to better understand who is likely to use ACP. For each of the BE-SE and MESTU-SE cohorts, we define an active ACP user as one that has executed ACP projects for at least two different days over the five week period. Another limitation here is that this only includes ACP server accesses (e.g. download or syncing to a project), since we cannot determine when students re-run previously downloaded projects. From the GPA cutoff, it is apparent that the predominantly domestic undergraduate cohort (BE-SE) is stronger than the predominantly international postgraduate cohort (MESTU-SE). In the case of the stronger undergraduate cohort, only 46% were considered active users. Yet, their performance in the mid-term was similar to their peers that did not. In contrast, the majority of the postgraduate students were active users (63%). For these students, they performed better than their peers that did not. Despite the limitations of low student numbers and no pre-test prohibit confirming ACP's effectiveness, it is still insightful to understand the differences of engagement between stronger and weaker cohorts within the delivery of the course.

To better understand ACP's impact on student performance, we analyze the stu-

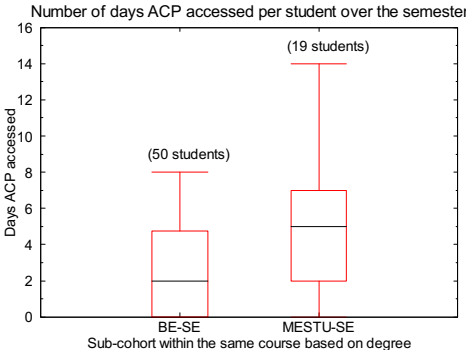


Figure 10: Level of ACP engagement throughout the course based on cohort.

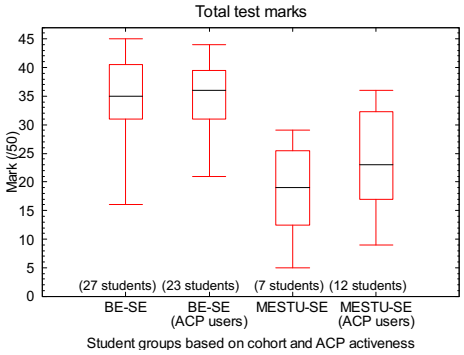
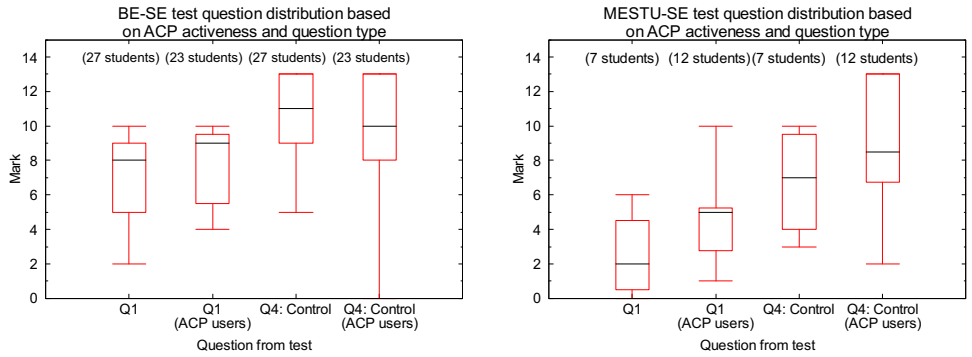


Figure 11: Total marks attained for mid-term test based on cohort and ACP activeness.

dent performance on particular questions in the mid-term test. Question 1 (Q1) of the mid-term related to concepts isomorphic to an ACP exercise that was available for students, while Question 4 (Q4) was used as a control question (had no relevance to any of the ACP exercises available to students). Figure 12(a) shows the performance of the BE-SE cohort on these questions, while Figure 12(b) displays this for the MESTU-SE cohort. Over 50% of the undergraduate students that used ACP performed better than 75% of their peers that did not use ACP for the isomorphic question. This same cohort however performed worse on the control question, suggesting that ACP may have assisted a weaker group. In the case of the postgraduate students, 50% of the active ACP users performed better than 75% of those inactive for the isomorphic question. For the control question, only 50% of this same group performed better than 50% of those not ACP active. While these latter results include smaller sample sizes, and differences that are not statistically significant, the trend is promising that weaker cohorts may benefit.

7 Conclusions

Parallel and Distributed Computing (PDC) has traditionally been considered an advanced topic in computing degrees, usually taught as an elective subject. With the recent trends of mainstream consumer devices becoming ubiquitously multi-core,



(a) BE-SE cohort (b) MESTU-SE cohort

Figure 12: Comparing mid-term test performance based on question type. Q1 was a question that included similar concepts covered in an ACP exercise, while Q4 was a control question that had no relevance to the ACP exercises.

all modern software developer must now possess the practical skills that will allow widespread application of the PDC concepts. In recognition of the vast number of PDC topics, the NSF/IEEE-TCPP Curriculum Initiative presents a framework to encapsulate the complete subject area; such a curriculum proposes "the *what*" of *hard skills* relevant to the PDC field. These PDC hard skills need to be blended with object-oriented and graphical user interface skills, since these are the staple concepts in a modern software engineering world. "The *how*" of achieving these hard skills includes using Active Classroom Programmer (ACP), an active learning approach that provides students with hands-on opportunities to develop parallel applications on desktop and mobile domains. In addition to these hard skills, the software industry also demands soft skills such as teamwork and communication. This paper discusses what these soft skills are, presenting a research-infused and problem-based learning approach that allows students to develop these skills in the context of parallel programming. The range of assessment methods in the course exhibits cases of weak correlations, demonstrating the inherent value of exploring a variety of assessment types. In particular, students recognize the benefits of non-traditional assessments (namely presentations and group work), and appreciate the opportunity to develop communication skills valuable for their career. Hands-on ACP exercises are presented and made available for PDC instructors to use, including access to the ACP tool.

Acknowledgments

We would like to thank the NSF/TCPP CDER Center Early Adopter awards for their support and feedback regarding SoftEng751. Their financial contribution has funded the purchase of a multi-core Android smartphone and tablet for students to use in the course for their projects. The authors have also received support that aided the development of ACP, allowing a repository of PDC exercises to be released to the PDC community. The travel grants have also allowed sharing experiences and receiving direct feedback from the EduPar community over the years.

References

[1] International Engineering Alliance, "The Washington Accord." `http://www.ieagreements.org/Washington-Accord`.

[2] International Engineering Alliance, "The Sydney Accord." `http://www.ieagreements.org/sydney`.

[3] S. K. Prasad, A. Chtchelkanova, M. G. F. Dehne, A. Gupta, J. Jaja, K. Kant, A. L. Salle, R. LeBlanc, A. Lumsdaine, D. Padua, M. Parashar, V. Prasanna, Y. Robert, A. Rosenberg, S. Sahni, B. Shirazi, A. Sussman, C. Weems, and J. Wu, "NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing - Core Topics for Undergraduates, Version 1." `http://www.cs.gsu.edu/~tcpp/curriculum`, 2012.

[4] Geeknet Media, "The state of software development today: A parallel view," 2012.

[5] N. Giacaman and O. Sinnen, "Object-oriented parallelisation of Java desktop programs," *IEEE Software, Software for the Multiprocessor Desktop: Applications, Environments, Platforms*, vol. 28, pp. 32–38, Jan-Feb 2011.

[6] H. Sutter, "The free lunch is over: A fundamental turn toward concurrency in software," *Dr. Dobb's Journal*, vol. 30, February 2005.

[7] H. Sutter, "The pillars of concurrency," *Dr. Dobb's Journal*, vol. 32, August 2007.

[8] Geeknet Media, "Parallel programing: Goals, skills, platforms, markets, languages," 2012.

[9] J. M. Bull, L. A. Smith, L. Pottage, and R. Freeman, "Benchmarking Java against C and Fortran for scientific applications," in *JGI '01: Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande*, (New York, NY, USA), pp. 97–105, ACM, 2001.

[10] A. Robins, J. Rountree, and N. Rountree, "Learning and teaching programming: A review and discussion," *Computer Science Education*, vol. 13, no. 2, pp. 137–172, 2003.

[11] M. Prince and R. Felder, "Inductive teaching and learning methods: Definitions, comparisons, and research bases," *Journal of engineering education*, vol. 95, no. 2, pp. 123–138, 2006.

[12] W.-Y. Hwang, C.-Y. Wang, G.-J. Hwang, Y.-M. Huang, and S. Huang, "A web-based programming learning environment to support cognitive development," *Interacting with Computers*, vol. 20, no. 6, pp. 524 – 534, 2008.

[13] T. Jenkins, "Teaching programming – a journey from teacher to motivator," in *The 2nd Annual Conference of the LSTN Center for Information and Computer Science*, 2001.

[14] G. C. Lee and J. C. Wu, "Debug it: A debugging practicing system," *Computers & Education*, vol. 32, no. 2, pp. 165 – 179, 1999.

[15] J.-D. Choi and S. L. Min, "Race frontier: reproducing data races in parallel-program debugging," in *PPOPP '91: Proceedings of the third ACM SIGPLAN symposium on Principles and practice of parallel programming*, (New York, NY, USA), pp. 145–154, ACM Press, 1991.

[16] N. Giacaman, S. Kalra, and O. Sinnen, "The active classroom: students and instructors parallel programming... in parallel," in *Proc. of 5th NSF/TCPP Workshop on Parallel and Distributed Computing Education (EduPar-15), in conjunction with IPDPS*, 2015.

[17] A. Cockburn and J. Highsmith, "Agile software development: The people factor," *Computer*, vol. 34, no. 11, pp. 131–133, 2001.

[18] N. Giacaman and O. Sinnen, "Research-infused teaching of parallel programming concepts for undergraduate software engineering students," in *Proc. of 4th NSF/TCPP Workshop on Parallel and Distributed Computing Education (EduPar-14), in conjunction with IPDPS*, 2014.

[19] W. Hung, D. H. Jonassen, R. Liu, *et al.*, "Problem-based learning," *Handbook of research on educational communications and technology*, vol. 3, pp. 485–506, 2008.

[20] A. Jenkins, M. Healey, and R. Zetter, *Linking teaching and research in disciplines and departments*. Higher Education Academy York, 2007.

[21] Institution of Professional Engineers New Zealand, "IPENZ: Engineers New Zealand." `https://www.ipenz.nz/`.

[22] N. Giacaman, "Illustrating concurrency and parallelism." `http://homepages.engineering.auckland.ac.nz/%7Engia003/house.html`.

[23] L. W. Anderson and D. R. Krathwohl, *Title: A Taxonomy for learning, teaching, and assessing: A revision of Bloom's Taxonomy of educational objectives*. Longman, 2001.

[24] TIOBE Software BV, "TIOBE programming community index." `http://www.tiobe.com/tiobe_index`, 2016.

[25] P. Carbonnelle, "PYPL PopularitY of Programming Language Index." `http://pypl.github.io/PYPL.html`, 2016.

[26] ACM and IEEE, *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*, 2013.

[27] K. J. Whittington, "Infusing active learning into introductory programming courses," *J. Comput. Sci. Coll.*, vol. 19, pp. 249–259, May 2004.

[28] N. Nagappan, L. Williams, M. Ferzli, E. Wiebe, K. Yang, C. Miller, and S. Balik, "Improving the CS1 experience with pair programming," *SIGCSE Bull.*, vol. 35, pp. 359–362, Jan. 2003.

[29] R. Fishman, "College decisions survey: Deciding to go to college," tech. rep., New America Education Policy, 2015.

[30] M. M. Robles, "Executive perceptions of the top 10 soft skills needed in today's workplace," *Business Communication Quarterly*, vol. 75, no. 4, pp. 453–465, 2012.

[31] R. Bridgstock, "The graduate attributes we've overlooked: enhancing graduate employability through career management skills," *Higher Education Research and Development*, vol. 28, no. 1, pp. 31–44, 2009.

[32] National Association of College and Employers, "NACE job outlook 2015," tech. rep., 2015.

[33] L. J. Shuman, M. Besterfield-Sacre, and J. McGourty, "The ABET professional skills – Can they be taught? Can they be assessed?," *Journal of engineering education*, vol. 94, no. 1, pp. 41–55, 2005.

[34] R. S. Hansen, "Benefits and problems with student teams: Suggestions for improving team projects," *Journal of Education for business*, vol. 82, no. 1, pp. 11–19, 2006.

[35] A. Mohan, D. Merle, C. Jackson, J. Lannin, and S. S. Nair, "Professional skills in the engineering curriculum," *IEEE Transactions on Education*, vol. 53, no. 4, pp. 562–571, 2010.

[36] J. Perrenet, P. Bouhuijs, and J. Smits, "The suitability of problem-based learning for engineering education: theory and practice," *Teaching in higher education*, vol. 5, no. 3, pp. 345–358, 2000.

[37] R. Griffiths, "Knowledge production and the research–teaching nexus: The case of the built environment disciplines," *Studies in Higher education*, vol. 29, no. 6, pp. 709–726, 2004.

[38] M. Healey, "Linking research and teaching exploring disciplinary spaces and the role of inquiry-based learning," *Reshaping the university: new relationships between research, scholarship and teaching*, pp. 67–78, 2005.

[39] D. F. Wood, "Problem based learning," *BMJ: British Medical Journal*, vol. 326, no. 7384, p. 328, 2003.