Re-imagining a Course in Software Project Management

Paul Ralph
University of Auckland[1] and University of British Columbia[2]
[1]38 Princes Street, Auckland, 1010, New Zealand
[2]3333 University Way, Kelowna, BC, V1V 1V7, Canada
paul@paulralph.name

ABSTRACT

It is now common for software engineering programs to include some project management education, typically in the final year of the undergraduate program, or at the postgraduate level. Such courses are challenging for several reasons: faculty are unfamiliar with management education and literature; the software project management literature is dominated by nonempirical, atheoretical, unscientific practitioner recommendations; common software engineering pedagogical approaches are inappropriate for teaching management; and many students seem unwilling or unable to take the course seriously (that is, by attending all classes and completing all assigned readings, activities and projects). This paper describes a postgraduate management course in software engineering, which was iteratively refined over seven years to address all of these problems. Its key features include: 1) scholarly, evidence-based readings; 2) quizzes on readings at the beginning of every class; 3) Brief, slide-free lectures; 4) copious in-class activities formalized into a gradeable workbook; 5) an ambitious, multistage project combining novel research, collaborative data collection and literature reviewing, peer review workshops and individual deliverables. The course embodies both research-led teaching and substantial, novel class-based research.

CCS CONCEPTS

• **Social and professional topics → Software engineering education**; *Model curricula*; *Computer science education*; Information systems education; Information technology education; • **Software and its engineering → Software development process management**;

KEYWORDS

Education, Training, Software Project Management, Scrum, XP, Lean, Kanban

ACM Reference format:
Paul Ralph. 2018. Re-imagining a Course in Software Project Management. In *Proceedings of 40th International Conference on Software Engineering: Software Engineering Education and Training Track, Gothenburg, Sweden, May 27-June 3, 2018 (ICSE-SEET'18),* 10 pages.
DOI: 10.1145/3183377.3183379

ICSE-SEET'18, Gothenburg, Sweden
 978-1-4503-5660-2/18/05…$15.00
DOI: 10.1145/3183377.3183379

1 INTRODUCTION

In an ambitious longitudinal study of 2300 undergraduate students, Arum and Roksa [2] found that:

(1) 36% of students "did not demonstrate any significant improvement in learning" over a four-year university degree.
(2) Among students who did improve, the average improvement was marginal.
(3) Learning was positively associated with faculty expectations and time spent reading, writing (including projects and problem sets, not just essays), and studying alone.
(4) Interaction with faculty outside of the classroom and measures of social integration are either unrelated or negatively related with learning.
(5) Group study and participation in fraternities and sororities are *negatively* related to learning.

The only plausible explanation for these findings is that popular pedagogy is broken.

When I was a student, completing a double-degree in computer science and commerce, pedagogy was obviously broken. The very good students read all the readings, completed all the problem sets, assignments, etc. and attended all the classes, where they were bored to tears. The computer science instructors painstakingly worked through problems we had just solved. The management instructors painstakingly described what we had just read. Meanwhile, both groups banged through hundreds of slides with thousands of bullet points, ignoring reasonable limits of human attention [12]. A faster pace or elevated discussion was impossible because most students did not read the readings, complete most of the problem sets or attend most of the classes. Most students crammed from the slides just before the tests and exams, and retained very little [18]. The instructors had to dumb-down assessments to maintain expected pass rates. This is bad pedagogy and we can do better.

My most successful attempts so far have been in a postgraduate course in software project management. I taught versions of the course from 2010 to 2016 inclusive at two different institutions. In one institution, it was a required course for postgraduate software engineering students and an elective for postgraduate computer science students. In the second institution, the course was tailored for students taking a masters degree in information technology management. Classes sizes ranged from 30 to 106. The course was taught as a single section with between 30 and 40 hours of in-class instruction.

The course aims to engender deep, detailed understanding of software project management theory and practice, from both manager and developer perspectives. This includes systemic interpersonal, organizational and managerial problems in software projects and the sociotechnical practices and techniques for overcoming them.

Its secondary aim is to contribute to research in software project management.

This article describes the course design and how its features address diverse challenges. The design of the course and my understanding of the problems co-evolved across its seven iterations. For simplicity, however, the article presents all of the problems together (Section 2), and then explains how the most recent design of the course (Section 3) addresses them. The next section reviews some indicators of success. Section 5 explores several lessons learned and outstanding issues. Section 6 concludes the paper with a summary of the course's key features and benefits.

2 PROBLEMS

This section reviews problems encountered in teaching software project management. The course design presented in the following section addresses each of these problems.

2.1 Content

Software project management research, textbooks and bodies of knowledge tend to present material through the lens of software development methods and project management frameworks (e.g. Extreme Programming, Lean, the Rational Unified Process, Scrum). This is problematic because methods oversimplify and over-rationalize the messy reality and diversity of software projects. They marginalize phenomena incongruous with their prescriptions, including amethodical development [28], illusory requirements and goal disagreement [5]. Meanwhile, focusing on methods confers undo credibility on the methods themselves—there is simply no compelling evidence that any particular method outperforms any other particular method on any important dependent variable.

Furthermore, methods crowd out theory and empirical literature from the curriculum. Genuine expertise is developed by incorporating experience with "sound relevant theoretical knowledge" [3, p. 222]. Not covering relevant theory therefore sabotages student's chances of integrating their future observations to develop genuine expertise. However, many software project management courses avoid relevant management theories including actor-network theory, behavioral decision theory, boundary objects, (Weber's) theory of bureaucracy, competitive strategy, complexity theory, design thinking, equity theory, feminism, flow, general systems theory, institutional theory, intellectual capital, prospect theory, soft systems theory, stakeholder theory, structuration theory, task-technology fit, team cohesion, transactive memory and work systems theory. While no single course would cover all of these, relevant theoretical and empirical discourse is supposed to be the primary basis of tertiary education. Indeed, management curricula predominately comprises relevant theories and their practical implications, as evidenced by bodies of empirical research. Specific practices are taught and applied in the context of these theories.

For example, a typical software engineering course might discuss "DevOps" as recommended by a professional software developer [10]. A typical management course, in contrast, might discuss the wide-ranging empirical literature on cross-functional teams [17] and its implications for different kinds of projects (e.g. combining the development and operations teams).

2.2 Reading

As explained in the introduction, more reading is associated with more learning. However, finding good readings and getting students to read them are both challenging. Popular textbooks embody the content problems discussed above; that is, favoring pseudoscientific practitioner recommendations over scientific theory and empirical findings. Students complain that the textbooks are expensive, boring, and unreasonably long (because they are wordy and padded). Many students do not consult assigned readings at all, let alone prior to class.

2.3 Slides, notes and note-taking

The busy instructor writes her course notes as bullet points in a slide deck, adorned with a few figures, comics, discussion questions and links to videos. In class, she talks through the bullet points. Sometimes she avoids student questions or rushes through complex ideas because 'we have to get through the material.' She draws the exam questions directly from the slides so students cannot complain that 'that wasn't covered in class.' Students ignore readings, skip classes and cram from the slides to pass the exam.

This narrative illustrates many problems:

(1) Reading and attending classes feels unnecessary because everything on the exam is in the slides.
(2) The slide deck discourages presenting complex or nuanced ideas and data. "Many true statements are too long to fit on a PowerPoint slide, but this does not mean we should abbreviate the truth to make the words fit. It means we should find a better way to make presentations" [29].
(3) The students do not take notes. Taking notes is not about copying information. Note-taking facilitates learning by forcing the student to pay attention and synthesize information [9]. Taking notes forces them to ask themselves (and sometimes instructors), 'what is important here, what is the key message, what should I write down?'
(4) After many such courses, some students *think of a course as a slide deck*. They cannot cope cope with any course where the instructor does not spoon feed them a set of course notes, preferably in .pptx format.

As controversial as this may seem, narrating from bullet points projected on a screen is ineffective pedagogy [29].

2.4 Evaluation

Four broad phenomena appear to undermine evaluation at the university level:

(1) One-on-one, face-to-face, line-by-line feedback is prohibitively resource-intensive. In a large class, one instructor cannot even get to all students during in-class activities.
(2) The percentage of students who do not take their education seriously greatly exceeds the failure rate most institutions will accept. A minority of students read most assigned readings, attend most classes, complete most assignments, and exhibit a reasonable grasp of the material. However, failing 50% of a class is untenable at most institutions. The grades simply would not be approved, and could be perceived as a failure of the instructor.

(3) Cheating is ubiquitous, even at the postgraduate level [16]. Many instructors seem delusional on this point. If half of students admit to cheating in anonymous surveys, but none of your students seem to cheat, it does not mean your students are honest; it means you are not good at catching cheaters. Onerous processes for prosecuting cheaters and limited penalties tacitly encourage cheating.
(4) Many institutions pressure faculty to employ formal, final exams. Large classes necessitate objective questions (e.g. multiple choice), which are pedagogically unsuited to complex material. For example, a multiple choice question cannot effectively test the ability to explain to stakeholders why fixed-price, fixed-schedule contracts are risky.

2.5 Research

Research and teaching are often conceptually separated. This false dichotomy has many negative effects, including:

(1) Missed opportunities for research in the classroom.
(2) Missed opportunities for research-led teaching and evidence-based educational materials.
(3) Disenfranchised academics including good researchers who resent teaching and good teachers who resent research expectations.

Rather, each course presents unique opportunities to bring relevant aspects of our research into the classroom. Such synthesis is more difficult when we teach well outside our research area, but even this can spur new collaborations with colleagues.

Unfortunately, most students lack the skills and experience to design a study, rigorously analyze qualitative or quantitative data, or write a manuscript at the level of quality demanded by good scientific outlets. This raises the question, *how can we build novel, useful research directly into class activities and evaluation, given students' lack of research experience?*

2.6 Summary

In summary, I was faced with nine interconnected problems:

(1) insufficiently evidence-based curricula
(2) poor-quality textbooks
(3) students not completing assigned readings before class
(4) tardiness; low attendance
(5) insufficient attention, note-taking, synthesis during classes
(6) ubiquitous cheating
(7) exams inappropriate for subject matter
(8) insufficient resources for detailed, rapid feedback
(9) students ill-equipped for many research activities

3 SOLUTIONS AND COURSE DESIGN

While the problems described above seem intractable, they can be overcome through good pedagogical design. This section describes the current design of the course, and explains how its features address each of the problems identified in Section 2.

3.1 Content

The course embraces research-led teaching [27]; that is, the content of the course is shaped by contemporaneous research in software engineering, including my own. The course is organized as shown in Table 1. Several aspects of its organization warrant explanation.

First, many of these topics depart from the content of bodies of knowledge documents, such as SWEBOK [4] and PMBOK [19], and may be unfamiliar to software engineering researchers. This is to be expected. When we begin with theory rather than software development methods and process models, different topics and structures emerge.

For example, empirical research suggests that conceptualizing software development as parallel processes of sensemaking, coevolution and implementation is more veracious and useful than conceptualizing it as iterating between analysis, design, programming and testing [23]. Here, *sensemaking* refers to making sense of the project context. *Coevolution* refers to the rapid cognitive oscillation between our understanding of the problematic context and ideas for the software product from which design concepts emerge. *Implementation* refers to programming and related activities including static code analysis and unit testing [22]. Much of the course is therefore organized into these categories, rather than categories based on an oversimplified software development lifecycle.

The large amount of preliminary material preceding project management proper is also noteworthy. Many of the topics covered under *theory*, *sensemaking* and so on, including stakeholder theory, goal modeling, multidisciplinary collaboration and team cohesion, are crucial for project management.

Other topics are necessary to rectify ubiquitous misunderstandings of software practice. For example, students are taught to "elicit requirements." Potential users do not "have" requirements. Often, they do not even have preferences. Most preferences are constructed spontaneously within a particular social situation (e.g. a user interview) [15]. Neither users nor analysts know whether these spontaneous preferences are required, irrelevant, or deleterious to success [21]. Rather, analysts typically face a complex, ambiguous problematic context where stakeholders disagree on problems, goals, desirable features and success conditions [5]. The analyst must not only imagine possible solutions but also build consensus around and enroll stakeholders in acceptable compromises.

Similarly, students conceptualize design as a rational procedure of navigating a virtual tree of decisions to find a satisfactory candidate. Designers do not navigate virtual decision trees [6, 31]—they generate solution concepts through coevolution and design thinking, a distinct cognitive process that involves but cannot be reduced to unstructured problem solving, creativity, analysis, searching, sketching and writing [7].

Students struggle with this philosophical shift, and need coaching to adopt better ways of thinking about software development. These realizations are vital for understanding project management (specifically the faulty assumptions of traditional, plan-driven methods) and applying contemporary management research.

The panel discussion in the first segment has been very helpful here. I invite a diverse group of software professionals for a panel discussion. I aim for two men and two women, varying in age, race, industry and role. The four panelists sit (facing the students) at a long table with hands-free microphones. Each panelist has five minutes to introduce themselves, followed by open questions and concluding with a few minutes for students to approach panelists,

Table 1: Course Outline

Segment	Topics	Hours
Introduction	Course philosophy and structure	2
	Panel discussion	2
Theory of Software Engineering	Systems: general systems theory; work systems theory; complex adaptive systems	2
	Sensemaking-coevolution-implementation theory	2
	Epistemology of requirements	2
Sensemaking	Strategic analysis	0.5
	Scenarios, personas and goal modeling	1.5
	Stakeholder theory	0.5
	Process modeling	0.5
	Interviewing	2
	Qualitative data analysis	2
Coevolution	Creativity	1
	Design thinking and designer behavior	1
	Sketching and rich pictures	1
	Storyboarding	1
	Multidisciplinary collaboration	1
	Video game design, guided emergence, playtesting	1
Implementation	Transactive memory and team cohesion	1
	Theory of misfits / technical testing / test-first vs. test-after	1
Project Management	Projectification of the firm	1
	Common project management challenges and practices	6
	Psychology of project management: motivation vs. control; cognitive biases	2
	Understanding and measuring software project success	
	Software contracts, estimation, deception, compliance environments	1
	Managing uncertainty	1
Term Project	Choosing topic, designing instruments, discussion, peer review	4

shake hands, get business cards, etc. The messy realities of real-world development tends to emerge as a theme in these discussions.

Another aspect of the course outline worth unpacking is the "common project management challenges and practices" item. This item represents myriad topics including artifacts (e.g. user stories, use cases), roles (e.g. product manager vs. product owner), practices (e.g. continuous integration, planning poker), concepts (e.g. release management, Agility), and theories (e.g. Weber's theory of bureaucracy). These topics are organized by the ubiquitous challenge they relate to, rather than any particular method, process model or other prescriptive framework. For example, I discuss the challenge of onboarding new team members, and explain how pair programming, stand-up meetings, peer code review, coding standards, prioritizing technical debt, team code ownership and simple design all assist enculturation.

A few other items of note:

- Despite being shown at the end, hours related to the term project are interspersed where needed.
- The implementation segment is brief because most students are already relatively skilled and unconfused about implementation.
- Some topics (e.g. including intellectual property, infrastructure, security) are briefly covered throughout the course, wherever they fit.
- Topics in various standards (e.g. SWEBOK, PMBOK, Essence) that lack construct validity or empirical support are intentionally omitted to make room for evidence-based curriculum.

3.2 Readings

Students are assigned 10 to 20 pages of reading per hour of in-class instruction, including:

(1) Literature reviews, especially shorter, more accessible reviews like those in *Communications of the ACM* and *IEEE Software* (e.g. Juristo [14] on unit testing). These constitute about half of the readings.
(2) Theory papers: detailed descriptions of a theory, its use and implications (e.g. Alter [1] on Work System Theory).
(3) Individual studies that exemplify a stream of research or a explain an important topic (e.g. Ralph [24] on software engineering success), especially where no good literature review is available.

(4) Instructional, textbook-like material on research methods including excerpts from Rubin and Rubin [25].
(5) Thought-provoking journalism, including Eichenwald [8] on Microsoft's lost decade.

This focus on literature reviews, including meta-analyses, meta-syntheses, systematic reviews and mapping studies, emphasizes and maintains the evidenced-based nature of the course. Most students grasp the findings if not the methodological intricacies.

3.3 Quizzes

Students complete a quiz at the beginning of (almost) every class. The quiz comprises five to ten multiple choice questions, which test recognition of concepts from the readings. I aim for questions that are straightforward if the student has read the material and incomprehensible otherwise. For example: *A project is a work system that is a) temporary, b) managed, c) cross-disciplinary, d) repeatable, e) well-defined.* The related reading explicitly defines projects as temporary work systems, but the distracters all seem plausible.

The quiz works as follows. The students arrive and pull a bubble answer sheet out of their workbook (see below). Multiple-choice questions are displayed via an auto-advancing slide deck. The instructor simply starts the presentation, and students have one minute to answer each question. The instructor then collects the bubble sheets and a teaching assistant later scans them. We used Remark OMR[1] to grade the quizzes automatically. While there are many other ways of conducting quizzes, this format seems most robust against cheating (see Section 5).

Quizzes are worth 20%. If there are *n* quizzes, only the top *n-2* grades are counted. This rewards the students who take all the quizzes, while allowing students to miss a couple of quizzes, for any reason, without substantial penalty.

These quizzes virtually eliminate four different problems problems. Attendance is consistently over 90%. Students are at their desks, ready to start *on time* so they do not miss the first question. There is no wasteful transition at the start of class—the quiz begins and everyone stops chatting. Students read materials. Not every student reads every word of every reading, but most of the students read significant portions. Moreover, they read the materials *before class*, which frees the instructor from rehashing the reading material. All of this improves learning, reduces cramming and allows more effective and interactive use of class time.

3.4 Classes Format and Workbook

Most recently, the course was taught as two, 110-minute lectures per week. A typical class begins with approximately 15 minutes of exposition, which explores, refines, extends and sometimes critiques the topic of the day and readings thereon. Exposition is given without slides and no notes are provided. Students are encouraged to take their own notes. As explained above, it is the process of taking notes, not the notes themselves, that helps students make sense of material and distill key messages.

Next, students explore a problem or practice a technique. Although most exercises are done in groups, each student has to fill out the relevant worksheet from a workbook I created for the course. Students complete exercises by writing directly in the workbook. Worksheets involve diverse tasks including answering discussion questions, drawing models and diagrams, writing scenarios, creating personas, matching things (e.g. agile practices to problems they address), analyzing text and brainstorming. Workbook grades make up 20% of students' final grades.

This is followed by a brief reflection. Reflections are usually implemented either as a class discussion or enthusiastic students sharing their work with the rest of the class. This exposition-practice-reflection cycle usually takes about 50 minutes, so a 110-minute class includes two such cycles.

The combination of the exposition-practice-reflection structure and workbook addresses four interconnected problems:

(1) Expecting anyone's undivided attention for a 110-minute lecture is unrealistic. Limiting lecturing, diverse exercises and encouraging note-taking help students pay attention.
(2) Students often complain that classes should be more interactive. Replacing lecturing with exercises and reflection increases interactivity.
(3) Between the abstract discussion, semi-structured exercises and the absence of course notes, many students worry about missing key points. Formalizing exercises in a workbook with the same structure as the course seems to alleviate these concerns.
(4) Many students do not complete in-class activities or take them seriously. Grading the workbooks addresses this, and it is much easier to grade the workbook holistically than to grade each individual exercise.

The current workbook has 27 exercises, varying in scope from about 10 minutes to about 90 minutes of work. It also contains the bubble-sheets used for the quizzes. By including the bubble-sheets in the workbook, and having students fill in their student numbers in advance, quizzes can begin on time without handing out the bubble-sheets. I keep extra bubble-sheets in my bag for students who forget their workbooks.

3.5 Project

Students complete a multistage, individual term project, which combines student choice, mass-collaboration, library research, primary data analysis, drafting, peer review and individual deliverables. This section describes each stage of the term project.

Directions for the project are given orally, in class, and not written down. This is admittedly controversial, but following oral directions is an important skill. How are students supposed to learn to take oral directions if we given all directions in writing?

3.5.1 Topic Choice. First, I brainstorm possible project topics (usually with colleagues). All of the students will use the same topic. Criteria for good topics include:

- The topic is appropriate for the course.
- Many students will find the topic interesting.
- Little or no good existing research addresses the topic.
- The topic is a good fit for the instructor's research area.
- The topic can be studied by collecting a set of artifacts or interviewing a set of professionals (see Section 3.5.2).

Students then select the topic by popular vote. Previous topics include:

[1] http://remarksoftware.com/products/office-omr/

- What are the dimensions of software engineering success?
- What kind of decisions are made during coding?
- What are the most common elements of real-life requirements documents?

3.5.2 Data Collection. Each student is responsible for collecting one data point—some years, a specific artifact (e.g. a real-life requirements document); other years, an interview with a software professional (e.g. about their perceptions of software engineering success). For the latter, students brainstorm potential interview questions. The instructor collects the suggestions and synthesizes a semi-structured interview guide. All of the students use the same interview guide, participant information sheet and consent form.

Students must submit unique data points—they are not permitted to interview the same people or submit duplicate artifacts. For interviews, they submit transcript. Students include metadata appropriate to the artifact or transcript (e.g. the source of the artifact; the name of the interviewee). The instructor compiles the metadata and artifacts or transcripts into a database. (The remainder of this paper will use "artifacts" for simplicity—this includes transcripts).

Since everyone needs to submit their data points before the project can continue, there is significant peer pressure to submit on time, and lateness has been rare at this stage.

3.5.3 Data Analysis. The database of artifacts or transcripts and (cleaned, de-identified) metadata are released to the students. Students review the artifacts and select some to study. They must include their own. How many others depends on the size of the artifacts. Most years I recommend four to six. Some students choose to analyze more.

The analysis depends on the type of artifact. For interviews, I recommend (and provide instruction on) open coding [26]. For artifacts, I have used different approaches. One option is open coding, where each student generates their own coding scheme from the data. The other option is closed coding, where the instructor generates a standard coding scheme a priori (possibly based on student suggestions). The latter is easier for the students but harder for the instructor.

Students typically produce a set of themes, patterns, categories or other findings, depending on the data. Since each student analyzes a different set of artifacts from a slightly different perspective (that is, their own), each student's findings are unique.

3.5.4 Library Research. Next, students review existing research related to both the topic in general and their findings in particular. Since the topics are always novel, no research directly addresses the topics, so students have to find indirectly-related work. Specific findings often relate to existing bodies of research (e.g. Stakeholder theory, conceptual modeling).

The project brief instructs students to aim for 20 academic references. In class, I explain how to tell the difference between academic and non-academic literature and give examples of literature reviews that synthesize rather than simply recount existing work. Students are encouraged to share useful references via a wiki. Students are encouraged to include references on the research approach just as in a normal academic paper.

3.5.5 Write-up. The students write a report describing their research and results. They are encouraged to use typical headings for academic manuscripts; namely, *Introduction, Literature Review, Methodology, Findings, Discussion* and *Conclusion*. The workbook contains an outlining exercise with recommendations for what to include under each heading.

Students are required to use the two-column IEEE conference template and encouraged to use LaTeX. The main body of their report is not to exceed three pages, exclusive of bibliography and appendices. Good students typically submit a large appendix, comprising numerous figures and tables, to reveal the chain of evidence from their data to their conclusions.

3.5.6 Peer Review. The course involves two peer review workshops. The first focuses on the analysis; that is, what will become the appendix. The second focuses on the body of the report. Students print their draft analysis or draft report and bring it to class. They are randomly divided into groups of three or four. Students pass their work around the group for review and feedback. Students write summaries of both the reviews they give and reviews they receive in their workbook. Therefore, if a student has not completed their analysis or draft, they cannot participate and lose marks for these workbook activities.

3.5.7 Benefits. The design of this project has many benefits. For one, it is extremely difficult (or irrational) to cheat. Faking a transcript or document is far more work than doing the interview or finding a document. Plagiarism has not been an issue because the topics are novel and used only once so there is typically nothing to plagiarize from. Students are encouraged to review each other's work and collaborate on most of the project. However, the instructor can make very clear that any two papers with near-identical results will be noticed and not tolerated—so far, I have done this with over 300 students and never detected identical findings. Even paying someone else to write the paper would be difficult: between the unorthodox subject matter and predominately oral directions, the expectations are difficult to explain to an outsider.

Furthermore, interviewing professionals is useful in several ways: the students get practice interviewing (a crucial analysis skill) and begin networking with software professionals. Reviewing actual artifacts is similarly eyeopening because they are generally more complex and messy than the toy artifacts they encounter at university. Moreover, both activities can produce substantial datasets for future research. While students' analysis rarely reaches scholarly standards, learning to analyzing text is crucial for sensemaking, but rarely taught in detail elsewhere.

The peer review workshops also have several benefits:

(1) The students receive immediate, detailed feedback.
(2) The students experience peer pressure to and receive grades for completing their analysis weeks before the deadline and their draft days before the deadline. This increases quality, reduces last-minute rushing, and provides opportunities to iterate on the analysis and write-up.
(3) Randomizing groups gives the students insight into the quality (or lack thereof) of their peers' work. Students often seem surprised at how good some reports are, and how bad others are.
(4) Students have an opportunity to ask clarifying questions about the expectations for the project.

3.6 Summary

Table 2 summarizes the key features of this approach. While none of these techniques is entirely new, the combination of evidence-based content, quizzes every class, slide-free lectures, a workbook of in-class activities, a research-intensive project and peer review workshops seems to address many common pedagogical problems.

4 EVALUATION

Evaluating a course is deeply challenging [11], not least because we typically cannot randomly assign students to two different versions of the same course or reliably measure dependent variables of interest. This article should be viewed as an experience report, not a study. It's conclusions are based on intuition and anecdotal evidence and therefore have a much lower epistemic status than rigorous empirical research. However, we can explore the relationships between the course's features and its aims (next), followed by various imperfect indicators of success.

4.1 Features and Aims

The first two problems mentioned in Section 2 were insufficiently evidence-based curricula and low-quality textbooks. Rather than a textbook, readings predominately comprise scholarly publications, especially literature reviews. The course content, moreover, focuses on theory and discrete, empirically tested practices, rather than software development methods and the gurus who propose them.

The next group of problems were lack of reading, low attendance and tardiness. These problems are all addressed by beginning classes with reading comprehension quizzes. The quizzes motivate preparedness, attendance and punctuality.

Another complex of problems involves students not paying sufficient attention in class, not taking notes and therefore not synthesizing in-class content. This is addressed by teaching without a slide deck, not providing course notes, encouraging note-taking, and focusing on experiential learning through in-class activities.

Section 2 further notes the tension between concern over cheating and exams (which ostensibly but do not really prevent cheating) being pedagogically inappropriate. This is addressed by the combination of the quizzes and project. Both are more pedagogically appropriate. Furthermore, the quiz protocol and project format discourage and impede cheating.

The next problem was insufficient resources for detailed, rapid feedback. This is addressed by the two peer-review workshops. These workshops provide a level of detailed, interactive feedback that would be otherwise impossible in a large class.

Finally, while the course aims to contribute to groundbreaking research, most students are ill-equipped for rigorous data analysis and lack the experience to write an acceptable scholarly paper. The course addresses this not only by encouraging collaboration on some of the project but also by focusing on generating a good dataset. While the students are not expert statisticians or qualitative data analysts, they are perfectly capable of locating examples of real-life requirements documents, design documents, user stories, UML models or other development artifacts. These datasets can be extremely useful for further research.

4.2 Size, Attendance and Punctuality

In 2015 and 2016, this was the largest postgraduate course in the Faculty of Science, with 105 and 106 students, respectively.

Before implementing the in-class quizzes, it was not unusual to have attendance under 50% in some classes, and I was often dismayed at the large number of empty seats. Students often turned up significantly late. After implementing in-class quizzes, I began taking head counts to make sure no one was filling out a quiz sheet for an absent student. Mean attendance in 2016 was 96%. Lateness was a non-issue.

4.3 Commendations and External Evaluations

I have received two teaching commendations related to this course, one for innovative use of teaching technologies, and one for exception levels of student satisfaction (more on this below).

External examiners have evaluated the course several times as part of routine university quality checks. Their evaluations have been very positive. For example, one examiner said:

"This is an ambitious course involving the development of skills that software engineers typically resist; kudos for managing so many students through the course with such high quality results! ... I note that the literature review asks for references to at least 20 academic sources, which seems high for a semester-long paper. The report peer review workshop is such a great idea, but notoriously difficult to manage; I'd love to hear your experiences with it."

4.4 Peer Evaluation of Teaching

Most years I have asked a colleague to audit a class and provide feedback. This feedback has been overwhelmingly positive. Here is an excerpt by a colleague who has won myriad teaching awards:

"Paul had carefully prepared several interactive activities for the students in this lecture. In one activity, Paul organised students into groups of three and distributed a worksheet to each group which was used for developing personas for a software project that he had envisioned. Immediately prior to the activity, he did a superb job of running an open class discussion in which students presented ideas for potential stakeholders in the project which then lead directly on to the persona development. At the end of the activity, each group presented their developed personas to the rest of the class. This was an excellent way of engaging students with the course material, and the scaffolding provided by the worksheet enabled students to develop incredibly detailed personas that varied greatly—precisely the desired outcome for this activity. It was clear that Paul had put a lot of thought into this activity, and it was equally clear that students gained a lot from participating as evidenced by the quality of their work... [he later] ended the lecture with an open discussion about the activities that students had undertaken, and he closed with a strong justification of why the process being taught was so valuable in practice."

4.5 Student Evaluation of Teaching

Notwithstanding the fact that student evaluation of teaching is uncorrelated with learning [30], student evaluations of the course have been overwhelmingly positive. In 2011, the course received a perfect score for teaching quality—every single student who completed the evaluation gave the course an overall score of five out of five. In 2013 and 2014 the course received the highest overall scores

Table 2: Course Features and Benefits

Feature	Details	Benefits
Content	Scholarly, evidence-based readings	Promotes expertise; develops critical readings skills
	Focus on theory	Inoculates students against pseudo-science
	Organized by challenge, not method	Promotes evidence-based practice; reduces reverence for methods
Quizzes	Given beginning of every class	Encourages preparation; improves attendance; reduces tardiness
	Auto-advancing slide show	Reduces cheating
Lectures	Brief lectures without slides	Improves attention, listening skills, hinders cheating
	No notes given	Promotes note-taking, synthesis
	In-class group activities with reflection	Improves collaboration skills; promotes retention
	Giving directions orally	Improves listening skills
	Panel discussion	Combats unrealistic expectations of industry
Workbook	Formalizes in-class activities	Reduces student anxiety from lack of course notes
	Graded twice per semester	Increases attendance and attention to in-class activities; reduces grading load
	Includes quiz sheets	Speeds up in-class quizzes
Term Project	Novel topic	Supports research, hinders cheating
	Collaborative data collection	Supports research, combats over-optimistic expectations of industry
	Two peer review workshops	Improves feedback, encourages good work habits, raises standards
	Large, collaborative literature review	Promotes reading, secondary research skills and researcher mindset
	Individual analysis and write-up	Promotes individual accountability, hinders cheating

in the masters program. In 2016, over 95% of students indicated that they were satisfied or very satisfied with the course and quality of teaching.

In a staff-student feedback meeting in 2011, several students vociferously argued that this course was the quintessential masters course—one other courses should emulate.

Students enjoy the succinct, slide-free lectures; for instance:

"Absolutely love your style of teaching, it is very engaging and informative. Way way wayyyyyyyyy better than reading off slides and falling asleep in class!"[2]

That said, in previous iterations, many students complained about the lack of course notes or slides. Some students clearly feel entitled to bullet point summaries provided by instructors. Some perceived the course as disorganized. Some students also expressed concern about vague exercises or lack of take-aways from the class. After creating the workbook, however, these comments all but stopped. As intended, the workbook replaces slides as a kind of security blanket for the students. Many students highlighted its usefulness; for example:

"The active learning through doing in class exercises really helped. It stimulated critical thinking and fostered cementing of what the themes of each class were."

The quizzes are controversial. Students do not *like* the quizzes, but grudgingly accept their effectiveness. In 2016, most negative student comments concerned the quizzes; however, *no one* suggested getting rid of them. Rather, students asked for small changes such as practice quizzes, more questions (to distribute risk) and clearer questions. One student summed up the class attitude:

"Forcing us to do the readings thoroughly by having quizzes on them was a tactic I really disliked, but was truly effective in making me learn the content."

Obviously, not every student appreciates every reading. One year, for example, I optimistically assigned Quine's *Two Dogmas of Empiricism* [20] to prepare for a discussion of the implications of epistemology for requirements engineering. The students universally condemned the paper as incomprehensible, so it was dropped. Some students, moreover, have complained about the volume of readings. *More* students, however, commented favorably on the readings, suggesting, for example, that they read much more for this course than their other courses and consequently learned much more. Students do not seem to mind most of the scholarly readings. One student explained:

"The quizzes made me meditate on the readings. The lectures go fast and deep into some concepts, but thanks to the readings everything was really clear."

The following additional student comments are included to give a sense of their reactions.

- "The lectures were the most genuinely engaging I have experienced at university. The content was interesting and was contextualized in a way that allowed us to understand its importance."
- "Without a doubt, best lecturer I've ever had. Made the course content interesting and I learnt heaps. I enjoyed the course being 100% coursework and think it was broken down excellently. It gives students the opportunity to get the best grade for putting in hard work."
- "Paul is seriously awesome. Such a passionate lecturer and genuinely interested in making his course as effective as possible. A different approach is taken here to any other

[2] All of the student comments reported in this paper were provided anonymously, and instructors do not receive them until well after grades are finalized, so there is no direct incentive for students to be overly positive.

course and I think it is significantly more effective. A+ course"

- "The teaching was amazing, making it easy to stay focused during the two hours lectures. The peer review was also a very good idea to improve my dissertation writing skills in general."
- "The quizzes were very helpful as they actually made me read more about the course before the lecture and gave me better understanding of what the course is all about. Plus the peer review is a great innovation."
- "The readings and quizzes were really helpful, and getting the essay peer marked. also not having lecture slides is a much better teaching method as it makes you really pay attention to what is being said and therefore take it in better—best course of the term without a doubt"
- "Topics were varied and interesting and Paul spoke very passionately and informatively about them."

5 LESSONS LEARNED AND OPEN ISSUES

Iterating on this course has generated several notable lessons, which are described in this section.

One bit of good news has been student appreciation for the more managerial and philosophical elements of the course. In 2016, students indicated that an article about human resources policies was the course's most interesting reading. Evidently, computer science and software engineering students are more interested in social and interpersonal issues than one might expect.

The delivery mechanism of quizzes matters. In previous iterations, I experimented with automated online quizzes. I accepted that small groups of students might take the quiz together. I did not expect students to post the answers in a private group so everyone else could copy them, which is by all accounts what happened. When I first tried in-class quizzes, I would advance the questions manually. Some students used this break in attention to cheat. An auto-advancing slide show while the instructor (and several teaching assistants if possible) watch the students is therefore preferable.

Similarly, the method of submitting data points and term papers matters. I switched to using questionnaire systems to enforce complete and correctly formatted metadata on data points. Students can still write "not applicable" for some fields but the questionnaire structure seems to have diminished missing metdata. A related problem is some students' tendency to exceed word limits. I experimented with having term paper text submitted through a structured field such that students simply could not submit more than the desired number of words. However, plain-text submission was too limited for this kind of project. Using the IEEE template is a good compromise. Some students still get creative with the formatting, but compliance seems higher than with no template and a word limit.

In most courses, instructions and guidance for term projects and activities are generally given in writing. In recent years, I have only provided instructions orally for several reasons:

(1) Most students do not seem to read extensive written guidance, especially the students who need it most.
(2) Taking oral instructions is a critical skill in which students get little practice.
(3) Giving oral instructions encumbers the use of essay writing services to cheat.

The panel discussion is a great addition. Both the students and panelists really enjoy it. Students ask insightful questions and panelists provide interesting details. When I ask the panelists about differences between their university courses and real-life experiences, their answers frequently debunk many of the same misconceptions discussed in Section 2.1. This primes the students for, and lends credibility to, some of the more avant-garde material. However, some students can get very worked up over panelist's comments, for example, when one panelist explained that employees join his start-up to work 60-80 hours per week trying to get rich. This too was a teachable moment—different people have different priorities and finding work that matches your priorities is important.

One perennial challenge with peer review is incomplete, unformatted or unedited drafts. Peer reviewers find this very frustrating, and much of their feedback comes down to 'finish the paper', rather than finding mistakes or suggesting improvements the student had not considered. Techniques that seem to have helped include: avoiding the term *draft*, emphasizing the importance of complete work for peer review, and formalizing the peer review process through the workbook. However, a more effective approach would involve a little deception: we could simply have the report "due" in the second last week of class, and then surprise the students with the peer review and opportunity to revise.

Another unresolved issue, and more fundamental question about tertiary education, is exactly what students are and should be entitled to. Some student comments suggest that they feel entitled to many things, including:

(1) The right to feel safe at university
(2) The right to free speech
(3) One-on-one communication with instructors, including email and face-to-face meetings
(4) Extensive (line-by-line) written feedback on assignments
(5) Course notes generated by the instructor
(6) Minimal penalties for cheating
(7) Passing all courses regardless of merit, attendance or completion of assignments

Most scholars would probably agree that items 1 and 2, above, should be guaranteed, while items 6 and 7 are absurd. Items 4-6, however, occupy more of a gray area, which this course has illuminated.

Some students have complained vociferously about being encouraged to ask questions in class or during office hours rather than use email. Some students clearly feel entitled to multiple one-on-one meetings verging on tutoring sessions outside of office hours. Given that time spent with faculty outside of class is very resource intensive and uncorrelated with learning [2], perhaps item 3 is fundamentally unreasonable. Some years I used Piazza(.com) to manage student communication. While this did reduce email, it backfired spectacularly when a student became publicly abusive (to me and several of his peers) and I discovered that Piazza has very limited mechanisms for instructors to shut down discussions.

Similarly, students often express dissatisfaction with feedback. Most years, when introducing the peer review workshops, I explained how the peer review is a compromise between the ideal but

impractical (real-time, line-by-line, one-one-one feedback from an instructor) and the practical but limited (slow, written, summary feedback from an instructor). The peer review workshops involve real-time, detailed feedback, but from several students instead of one instructor. While the peer review workshops have been well-received, many students still do not understand or accept that the level of instructor feedback they desire is impractical.

Finally, some students have expressed that it is somehow improper for instructors not to provide course notes. I find this utterly baffling and have no explanation for this attitude. As explained above, providing notes discourages reading, attendance and note-taking. Perhaps if more instructors abandoned slide-driven lecturing and encouraged note-taking, this attitude would subside.

6 CONCLUSION AND OPEN ISSUES

In summary, this paper describes a novel approach to a capstone or postgraduate course in software project management. It documents my experiences developing and teaching the course over seven years.

It asserts that evidence-based readings, quiz-every-class, note-taking, a workbook of in-class activities, ambitious term projects and peer review are obviously more effective than textbooks, students skipping classes and readings, slide-supported lecturing and superficial final exams.

This article further asserts that software project management curricula should be grounded in relevant software engineering and management theory, rather than software development methods, process models, project management frameworks and other practitioner-generated prescriptions. The course organization is therefore offered as an alternative to the knowledge areas in the ACM/IEEE model curriculum [13], and the reading list is available from the author.

As this is an experience report rather than a study, the evidence for these assertions is at best anecdotal. However, I submit that few of us really believe that reading bullet points off slides to students who neither read materials nor write notes is effective pedagogy. We do not do it because it is effective, we do it because it is easy and protects us from criticism. Similarly, few scientists really believe that basing curriculum on scientific research is a bad idea. It is just that using a textbook is easier.

While intended mainly for computer science and software engineering students, this course could also be valuable for management students or as part of a game development program. However, the recommended curriculum and pedagogy may not suit every instructor and context. To be blunt, this style of teaching requires a skilled, knowledgeable instructor.

I hope this exposition is thought-provoking and promotes more effective software engineering curricula and pedagogy.

REFERENCES

[1] Steven Alter. 2013. Work system theory: overview of core concepts, extensions, and challenges for the future. *Journal of the Association for Information Systems* 14, 2 (2013), 72–121.

[2] Richard Arum and Josipa Roksa. 2011. *Academically Adrift: Limited Learning on College Campuses.* University of Chicago Press, Chicago, USA.

[3] A Bogner, B Littig, and W Menz. 2009. *Interviewing Experts.* Palgrave Macmillan, London, U.K.

[4] Pierre Bourque and Richard E Fairley (Eds.). 2014. *Guide to the Software Engineering Body of Knowledge, Version 3.0.* IEEE Computer Society.

[5] P. Checkland. 1999. *Systems thinking, systems practice: includes a 30-year retrospective.* John Wiley.

[6] Nigel Cross. 2001. Design cognition: Results from protocol and other empirical studies of design activity. *Design knowing and learning: Cognition in design education* 7 (2001), 9–103.

[7] N Cross. 2011. *Design Thinking: Understanding How Designers Think and Work.* Berg, Oxford, UK.

[8] Kurt Eichenwald. 2012. Microsoft's Lost Decade. *Vanity Fair* (Aug. 2012), 1–12. http://www.vanityfair.com/business/2012/08/microsoft-lost-mojo-steve-ballmer

[9] Michael J A Howe. 1974. The Utility of Taking Notes as an Aid to Learning. *Educational Research* 16, 3 (June 1974), 222–227. DOI: http://dx.doi.org/10.1080/0013188740160310

[10] M Httermann. 2012. *DevOps for Developers.* Apress.

[11] B. Johnson and L. Christensen. 2008. *Educational Research: Quantitative, Qualitative, and Mixed Approaches.* SAGE Publications.

[12] A H Percival F Johnstone. 1976. Attention Breaks in Lectures. *Education in Chemistry* 13, 2 (March 1976), 49–50.

[13] Joint Task Force on Computing Curricula. 2015. *Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering.* IEEE, ACM.

[14] Natalia Juristo, Ana Moreno, Sira Vegas, and Martin Solari. 2006. In Search of What We Experimentally Know about Unit Testing. *IEEE Software* 23, 6 (2006), 72–80. DOI: http://dx.doi.org/10.1109/MS.2006.166

[15] Sarah Lichtenstein and Paul Slovic. 2006. *The Construction of Preference.* Cambridge University Press, Cambridge, UK.

[16] Donald L. McCabe, Kenneth D. Butterfield, and Linda Klebe Trevino. 2006. Academic Dishonesty in Graduate Business Programs: Prevalence, Causes, and Proposed Action. *The Academy of Management Learning and Education ARCHIVE* 5, 3 (2006), 294–305.

[17] Edward F McDonough. 2000. Investigation of Factors Contributing to the Success of Cross-Functional Teams. *Journal of Product Innovation Management* 17, 3 (May 2000), 221–235. DOI: http://dx.doi.org/10.1111/1540-5885.1730221

[18] Shelby H McIntyre and J Michael Munson. 2008. Exploring Cramming. *Journal of Marketing Education* 30, 3 (May 2008), 226–243. DOI: http://dx.doi.org/10.1177/0273475308321819

[19] PMI. 2013. *A Guide to the Project Management Body of Knowledge* (fifth ed.). Project Management Institute, Newton Square, PA, USA.

[20] W V Quine. 1951. Two Dogmas of Empiricism. *The Philosophical Review* 60, 1 (Jan. 1951), 20–43. http://www.jstor.org/stable/2181906

[21] Paul Ralph. 2013. The Illusion of Requirements in Software Development. *Requirements Engineering* 18, 3 (2013), 293–296. DOI: http://dx.doi.org/10.1007/s00766-012-0161-4

[22] Paul Ralph. 2015. The Sensemaking-coevolution-implementation theory of software design. *Science of Computer Programming* 101 (2015), 21–41. DOI: http://dx.doi.org/10.1016/j.scico.2014.11.007

[23] Paul Ralph. 2016. Software engineering process theory: A multi-method comparison of Sensemaking-Coevolution-Implementation Theory and Function-Behavior-Structure Theory. *Information and Software Technology* 70 (2016), 232–250. DOI: http://dx.doi.org/10.1016/j.infsof.2015.06.010

[24] Paul Ralph and Paul Kelly. 2014. The Dimensions of Software Engineering Success. In *Proceedings of the International Conference on Software Engineering.* ACM, Hyderabad, India, 24–35.

[25] Herbert J Rubin and Irene S Rubin. 2005. *Qualitative Interviewing (2nd ed.): The Art of Hearing Data.* SAGE Publications, Inc., Thousand Oaks, CA, USA . DOI: http://dx.doi.org/10.4135/9781452226651

[26] J Saldana. 2012. *The Coding Manual for Qualitative Researchers.* SAGE Publications, London, UK.

[27] Jan Schapper and Susan E Mayson. 2010. Research-led teaching: moving from a fractured engagement to a marriage of convenience. *Higher Education Research & Development* 29, 6 (Dec. 2010), 641–651. DOI: http://dx.doi.org/10.1080/07294360.2010.489236

[28] Duane P Truex, Richard Baskerville, and Julie Travis. 2000. Amethodical systems development: the deferred meaning of systems development methods. *Accounting, Management and Information Technologies* 10, 1 (2000), 53–79.

[29] Edward Tufte. 2003. *The Cognitive Style of PowerPoint.* Graphics Press, Cheshire, CT, USA.

[30] Bob Uttl, Carmela A White, and Daniela Wong Gonzalez. 2017. Meta-analysis of faculty's teaching effectiveness: Student evaluation of teaching ratings and student learning are not related. *Studies in Educational Evaluation* 54 (Sept. 2017), 22–42. DOI: http://dx.doi.org/10.1016/j.stueduc.2016.08.007

[31] Carmen Zannier, Mike Chiasson, and Frank Maurer. 2007. A model of design decision making based on empirical results of interviews with software designers. 49, 6 (June 2007), 637–653. DOI: http://dx.doi.org/10.1016/j.infsof.2007.02.010