Teaching Statement

Prof. Bill Tucker

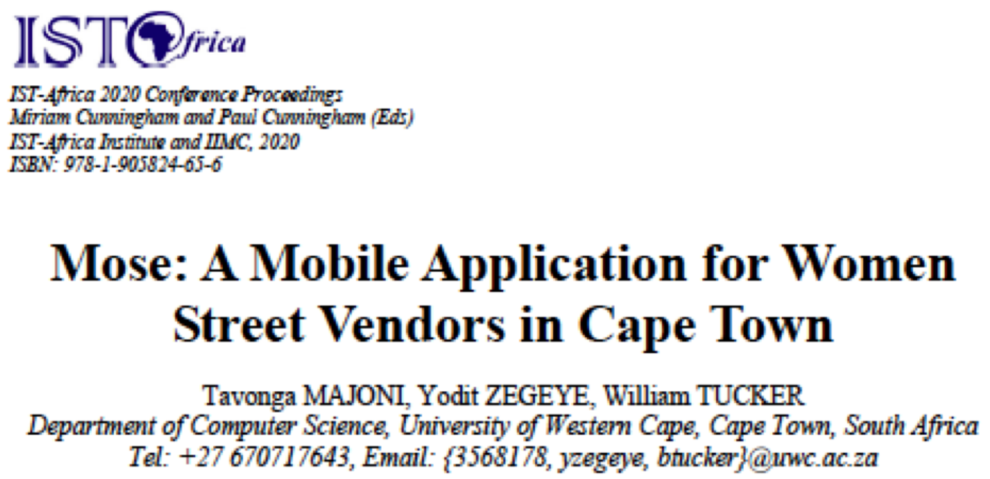
submission that earned Faculty Excellence in Teaching award to senior academic in Faculty of Natural Sciences, UWC, 2018.
last revised Sep 2020

Students' needs

During my years at UWC, I have seen a *technical* shift in the capabilities, and thus the needs, of the incoming Computer Science students from a standard bell curve distribution to an inverse bell curve that I call the 'Gini curve' that clearly separates the 'haves' from the 'have-nots'. Whereas most incoming students had likely not seen a computer 10-15 years ago, currently half of the 1st year students pitch up with their own laptop and the attendant computer literacy skills that accompany owning such a device. The other half displays paucity in computing background similar to the majority of the students 10-15 years ago. I usually enquire about the use of laptops and mobile phones with a simple show of hands. Due to my assessment techniques (see below), I also see this Gini-like spread of technical capabilities manifest in terms of written and programming performance.

However, no matter which side of the 'Gini curve', almost all of the students suffer from poor communication skills. As a native speaker of English, and as one who has more than 40 years of teaching experience, having begun teaching English as a foreign language as a teenager living in Taiwan, I am painfully aware of the dreadful English skills across the board: reading, writing and speaking. My assessment techniques (see below) are meant to simultaneously identify and address both technical and language skills because, based on my own experience in both postgraduate studies and in industry, the combination of the two comprise the key to any Computer Science student's future.

By the time students get to Honours level, the 'Gini curve' appears to return to a normal distribution (probably because only the best continue at Honours level, roughly 20%). I continue to emphasise both technical and language skills at postgraduate level where, for example, Honours students present a year-long project four times throughout the year to the department, in both a verbal presentation and a written report; each time also pre-presenting to the research group, and supervisor(s). Thus we are continuously able to assess and help improve written and spoken communication skills. We continue to do so at MSc and PhD levels, too, moving on to submitting work-in-progress papers, posters, and papers published in accredited national and international conference proceedings (see publications listed in my CV and Google Scholar as evidence that this is working well, and below shows the header of a paper published at IST-Africa as a result of my Honours ICT4D course). We often emulate the peer review process in house, in courses and in research seminars, because all Computer Science conference proceedings submissions are peer-reviewed, and thereby serve as preparation for subsequent submission to accredited conferences and journals (see Figure 1).


IST-Africa 2020 Conference Proceedings
Miriam Cunningham and Paul Cunningham (Eds)
IST-Africa Institute and IIMC, 2020
ISBN: 978-1-905824-65-6

Mose: A Mobile Application for Women Street Vendors in Cape Town

Tavonga MAJONI, Yodit ZEGEYE, William TUCKER
Department of Computer Science, University of Western Cape, Cape Town, South Africa
Tel: +27 670717643, Email: {3568178, yzegeye, btucker}@uwc.ac.za

Figure 1: Paper published from the 5th iteration of an ICT4D Honours course project output.

Moving students from a 3-year BSc degree to postgraduate studies, we lose 75-80% of our students to industry due to financial pressures; as understandably, most come from poor and disadvantaged backgrounds. These students leave varsity to get a job as soon as they can. I strongly encourage a 4th year to 'cement' skills in a professional context, to enable students to gainfully enter the workforce with a stronger skillset and/or to pursue advanced postgraduate studies. There are several ways to do this. Firstly, with funding (more on this later). In the classroom, though, I sometimes use programming exercises at 1st year to project income based on various degrees, entry-level salaries (based on communication with my graduates) and annual increases. I also show these results to third year students to remind them, at least in part, financially, why they should get an Honours and/or MSc degree. Secondly, I constantly integrate research activities, e.g. Zenzeleni and SignSupport, into undergraduate and Honours courses to expose students to applications of the basic concepts they are learning in class. Thirdly, due to my industry experience in the American dot com boom of the early 90's, and long term corporate research funding and collaboration in South Africa with Telkom, Cisco, and Aria Technologies, I am also able to integrate a contemporary South African corporate professional experience into my lectures, exercises and postgraduate activities. Finally, through my research group's software development with free and open source software (FOSS), I am also able to integrate FOSS mechanisms of software development into courses and research activities. We also publish all of our research as Open Access via the UWC Research Repository, making both the software and the outputs available to all.

I am grateful to have had long term THRIP funding which allowed bursary payments to Honours students. We had been paying Honours students R20k/annum since 1999 whereas the NRF only upped its Honours support from R8k/annum much later on. We were able to 'top up' NRF bursaries, allowing students a 50% increase of their NRF bursaries tax free, with no strings attached, as our students often quality for Scarce Skills bursaries. For the promising Honours students that we are able to retain after a BSc, we saw an almost 90% continuation to successful MSc studies for many years, also in part because we fund them amply with soft (restricted) funds, both South African and foreigners alike (note, this was written in 2018, and the situation, especially with covid, has changed and foreigners battle to get financial support, so we must be creative).

Teaching and learning

My approach, at both under and postgraduate levels, is based on learning by doing, aligned with the notion of authentic learning. This is a transmission view to teaching and learning, and is not passive at all. I have been employing situated 'learning by doing' techniques since my early days of teaching English in Taiwan: to community college students preparing for the TOEFL, to business people needing English to conduct business with foreigners and to children in a dual language (Chinese/English) kindergarten. In each case, my approach was based on situated conversation, reading and writing in English with an emphasis on two-way conversation.

Throughout my undergraduate studies, I did not once study or cram for a final exam because I attended class regularly and did assignments on time. At the University of Alberta and then at Arizona State University, I experienced a wide range of teaching and learning techniques as a student. Subsequently, I adopted the techniques from the only postgraduate course for which I received a B. I got all A's in Computer Science from the very start. In my opinion, I learned the most in the single course that caused me the most grief. It was a 'learning by doing' course, writing an operating system from scratch in assembler. When I came to UWC, I also brought with me the 'learning by doing' approaches gained from six years of industry experience in a professional software engineering environment that stressed design, documentation, source code control, rigorous debugging and testing, and performance profiling; exactly for what that tough operating systems course had prepared me. Now, after more than 20 years of university teaching, learning and conducting research, I still follow that same approach, and still keep in touch with Emeritus Professor David Pheanis who taught me this way. Needless to say, I have adopted his techniques for my own purposes, and the UWC environment (and kept in the loop the entire time).

My main approach to Computer Science education is programming, and more programming; alongside documentation and more documentation. My approach is distinguished in that I insist that a program must work perfectly, and be verified automatically by code I have designed, before it is marked for internal and external documentation, i.e. on written English in terms of content, structure, format, spelling and grammar. This sometimes perplexes students (and other lecturers) as students are most often awarded partial marks for partially working code. I do not. If code does not work as specified, e.g. only passes a total of 999 out of 1000 test cases, a student receives 0 marks, not 99%. Working code, e.g. passing all 1000 test cases, is a pre-requisite to marking the documentation of the code that describes what the code does and how it is done, in technical English. In the 'real' world, this is the norm, as is version control, testing and debugging, and performance profiling. These activities, in addition to authoring user guides and websites in English, are 'tools of the trade' that I insist that students learn by doing, and thereby award marks for these tasks above and beyond the task of programming. In the 'real world', if code does not work properly, no one will buy it; and in the academic world, if code does not produce results, it is difficult to get a degree and/or publish. I also only use continuous assessment (see below) and refrain from giving a final exam whenever possible. If I must give a final exam, e.g. for 1st years, it is a programming rather than a written exam. I am not a fan of supplementary nor special exams. I feel they waste so much time, in fact up to 62 academic days per year at UWC; time that in my opinion would be better spent having an additional semester to provide more authentic learning opportunities to students that come to university under-prepared, e.g. to retake 'killer' courses (so we avoid the cross-year clashes and extended enrolment) and also to take additional courses.

Figure 2: The flipped classroom: students must read material before coming to class.

Throughout my entire teaching career, I have employed the 'flipped classroom' (yes, the photo in Figure 2 is intentionally flipped) approach where students must prepare material before coming to class. There are two methods to accomplish this: with undergrad (and some postgrad) courses, I use random pop quizzes (described elsewhere). For the Honour ICT4D course, where students read 2 papers per week, they must submit a one page structured summary before that week's seminar on the topic. This builds an incrementally iterated approach to writing summaries, honing skills that can then be passed on to iterations of a class project, consisting of 4 cycles of presentation, demo and paper (see Figure 3 below). This is a form of scaffolded continuous assessment, where every assessment builds on the previous one. All assessments count equally so as not to put all the emphasis on the final product, but the process.

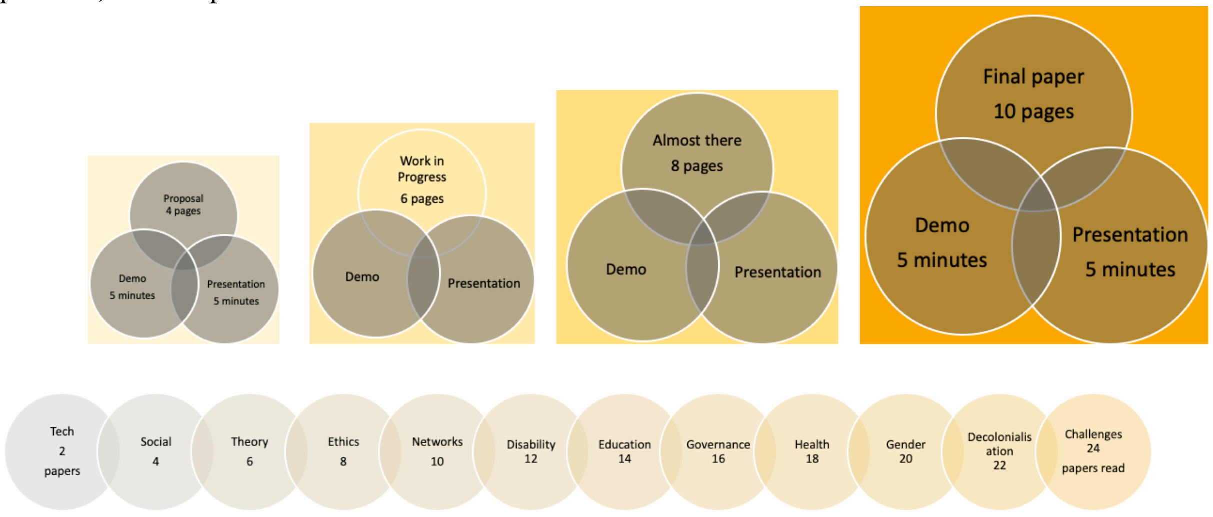


Figure 3: Scaffolded continuous assessment via incremental iteration of all submissions, in this case, presentation, demo, paper and weekly summaries. In this semester-length Honours ICT4D course, nly the 4th iteration of the prototype is coded.

The differences in T&L approaches within my department are often stark, and for that I believe the students should be grateful. Students must be exposed to different methods of instruction and assessment, for I have also benefited throughout my education and industry experience from a variety of approaches and methods. We should be teaching students how to learn to program, not how to program; and that requires a mixed bag of tools.

Curriculum development

In Computer Science, there is a curriculum recommended by a combined ACM/IEEE effort. The ACM and IEEE Computer Society are considered the two most prestigious professional computing bodies, both based in North America. Their computing curriculum recommendation puts us in a very difficult situation as it is intended for a 4-year degree, and we only offer a 3 year BSc in South Africa. As my colleagues can attest, I am very vocal at our annual curriculum development meetings where we consider adapting our curriculum to follow the ACM/IEEE recommendations while at the same time acknowledging the reality that most of our students leave with a 3-year degree that often takes longer than three years, not including those enrolled in the four year extended programme, which spreads the first year over two years (and note, there is no Computer Science in the first year of the extended programme). We do not give near the same number of courses as our peers in the North. Nor are our students allowed to take courses outside the Faculty, which is a requirement in the North.

I feel we should be reorganising existing coursework to ensure that our 3rd year BSc students leave with at least an introduction to the recommended topics within the constraints of the module system given to us. For the latest version of the ACM/IEEE recommendation, recently revised, we must include more coursework on security; distributed and parallel processing; and ethics and social issues of computing. I have already reorganised 3rd year operating systems and networking courses that were originally designed to 'hook' students into staying on for Honours to complete the courses (because most textbooks are designed for a semester course and we only offer term courses, so term 1 would be given, for example, at the end of the 3rd year with the follow up in the first term for Honours) so that the 3rd year course is more comprehensive (although more shallow). I also suggested that we move the industry-based Cisco CCNA networks course to the evening for revenue generation, and return to a more academic orientation with computer networks (we have done the latter, and only recently, the former). I have also suggested that we synthesise distributed computing into the operating systems modules, and security into the computer networks. We introduced an ICT4D course at Honours level in 2012, and in 2017 it was made a permanent addition to our curriculum, and it includes ethical and social issues of computing, as well as open source software engineering and an introduction to research methods (we do not presently have such a course). However, due to the size of our department (8-9 permanent academics), lecturing eats much into our research time. That's ok. In addition, I have also suggested that a combined networks and operating systems course could easily follow on the combined database/software engineering/human computer interface grouping in terms of an extended year-long programming project at third year level. These ideas are still under discussion within the department.

I was responsible for redesigning the Honours curriculum and project structure in 1999, to reflect the Software Development Life Cycle (SDLC) best practices *by doing it*, and we continue to successfully use it to this day (see www.cs.uwc.ac.za -> Honours). Actually, I feel it must be updated. It has not changed since 1999! For example, in 2020, after I had done this for several years, the department finally agreed to have Honours students iteratively write a 10 page paper on their project (see Figures 1 and 3 above; instead of a 60-100 page report. Yet, the remainder of the Honours programme remains untouched for more than 20 years!

I am also quite adamant that we further adapt our curriculum to train Computer Scientists *in* Africa *for* Africa, and not just for the West, as is typically done in Computer Science departments across the country, and the world. We are actually training our students to work in the USA, UK and Australia rather than in the so-called 'developing world' which we inhabit. Our textbooks, even the International Editions, come laden with Western (mostly American) preconceptions, terminology and tacit cultural assumptions that are not necessarily appropriate for our context. For example, the African notion of Ubuntu is very different from the individualist competitive paradigm that comes embedded in our textbooks and our application of them within our courses. As an immigrant to South Africa, I have bumped my head against these issues, in addition to the very different marking scale, e.g. in South Africa an 'A' is 75, whereas in America 75 is a 'C', or average. I think we shoot too low. Students just want to pass, i.e. get a 50, not excel. Regarding Ubuntu, I often take an extremely flexible approach to group work, allowing students to choose their own groups, and even change group membership at any time, and actively encourage students organised into groups to discuss assignments freely and openly with each other, even between groups. However, I also stress the ethical considerations of copying work, getting solutions from the Internet and obtaining pirate pdfs of textbooks; as far as I know, I am the only person in the department that asks students to sign a version of the UWC "plagiarism declaration" modified for Computer Science. However, in the 'real' software development world, whether corporate or open source, communication is 'king', especially with respect to communities in Africa. Thus, I feel we need to adapt the Western ACM/IEEE curriculum for African students to link their studies to community needs, and to solve tasks with a thoughtful combination of African and Western approaches, for I feel my students have a foot in each world, and can benefit from that, particularly for our context. Even when assignments or peer reviews are done individually, I allow and even encourage discussion between students. This is natural and beneficial for them to learn from one another.

To such ends, I frequently edit a given text's examples, both in presentation and in programming examples and exercises, with localisation, e.g. rands instead of dollars and "bond" instead of "mortgage". These are simple ways to situate learning and enhance understanding (of examples). I also take pains to craft exercises that are directly relevant to students' lives, e.g. to project bond interest with and without overpayment and other exercises to handle the vast amounts of money our graduates earn upon graduation because their parents often lack that experience and thus cannot pass it on to their children. We also have exercises that compare mobile phone packages, voice over IP vs. standard voice, SMS vs. WhatsApp and even implementing the South African tax code. I also attempt to weave the current political and ICT policy issues into lectures, e.g. before 2005 when voice over Internet Protocol (VoIP), e.g. Skype, was illegal; or handing out the recently gazetted ICT policy document and asking 1st year students to comment on it as a marked reading/writing exercise. These are yet other ways to situate learning, especially in a South African context.

To take this localisation even further, I have for several years now advocated that we start programming on mobile phones. I mean programming **for** and **on** mobile phones. Mobile phones are essentially tiny computers, and are ubiquitous in Africa, much more so than in America, for example, where a smart phone was once far and few between while ubiquitous in my classroom. Many of our 1st year students have advanced smartphones, and even if they do not, they can group themselves together with someone that has one, or at the very least, can use a simulator on a PC or server in the department. I have started on this path by introducing graphical user interface programming exercises at first year level. Before I started doing that, we only introduced students to GUIs at 2nd, and now 3rd, year level, which is in my opinion far too late; and flies in the face of ubiquitous GUIs on phones, tablets, laptops, PCs and servers. We need to move with the times, and also with our continent, to provide relevant examples for students to hone their computing skills, on mobile devices in particular.

Assessment

I use only continuous assessment, mainly in three forms: pop quizzes, design exercises (for pracs), and programming pracs (however, not done during prac time, but on their own time, see below) for undergraduate programming classes, and summaries, papers, presentations, demos and class participation for Honours classes (see Figure 4 below).

Pop quizzes are 10-minute individual assessment events that are handed out randomly, without notice (to achieve the flipped classroom). I usually give them at the beginning of a class to encourage timeliness, and make sure they come to class prepare to absorb and reflect upon the material. It still astounds me that even when pop quizzes become de rigor, students often still arrive 30 minutes late, or later, to sign an attendance roster and depart out the back of the hall, because the pop quiz also doubles as an attendance data collector, although by the 3rd year, the students 'get it'. I tend to drop the two (or more) lowest scoring pop quiz marks out of about 12-15 quizzes per semester. There are no make-ups. I always discuss the pop quiz answers immediately after collecting them from students, and use it as a vehicle to manage the pace and learn exactly where students are excelling and struggling. Because each pop quiz is short, we can mark them quickly and make them available to students for nearly same-day feedback. I also use the pop quizzes to deter cheating and copying: we often circulate several versions of the same pop quiz. All versions of the pop quiz look the same, and address the same topics in slightly different ways. That way the answers to the 'master' pop quiz are relevant to all versions, and when marking, we can easily tell who has copied from whom. This deters plagiarism quickly and effectively with only a small extra effort on our part. Note that I have exposed plagiarism in every single course I have ever given at UWC. We have an internal protocol for dealing with plagiarism before sending a student to the Proctor (which follows on the plagiarism declaration form mentioned above, and involves an intermediary step of admitting to plagiarism and only the 2nd instance is forwarded to the Proctor with all 3 documents as evidence).

- Drop the lowest scores for each category
- No final exam
- No tests of any kind

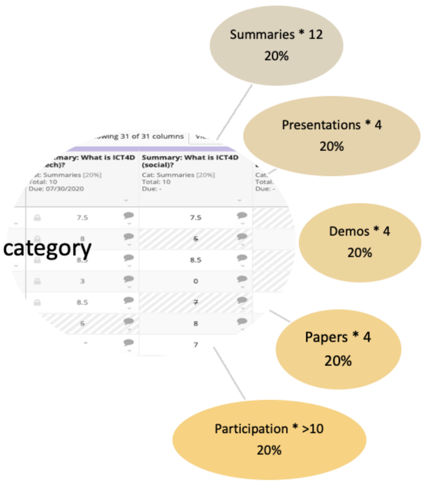
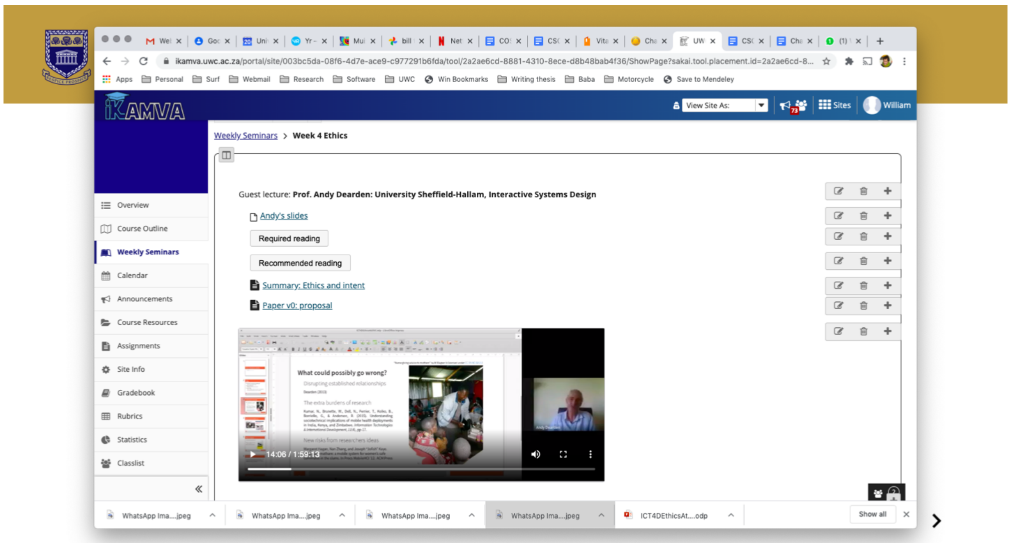


Figure 4: Marking is for continuous assessment, and lowest marks are dropped in all categories to allow for unforeseen circumstances, e.g. death in family, illness, missing a class, etc.

Programming (and non-programming) pracs/assignments are handed out on a weekly basis, and for programming, each with its own preliminary design exercise (also marked). For first year students, we expect the students to complete the assignment in the laboratory at set times (although they can work on it anytime – all assignments are given at the beginning of the week and pracs are scheduled Wed, Thu and Fri), and for 3rd year and Honours students, we expect students to complete the assignment on their own time and submit to an automated system on our servers (VPN access enables off campus access even when the campus is shut or locked down). In both instances, this builds valuable time management skills. Until 2015, all assignments were done in teams: 1st years program in teams of 2, 3rd years in teams of 3 or 4 and Honours in teams of 2 again. In 2015, I opted to go for individual submissions, even at 1st year, to deter relying on someone else to program. It turns out that one person per team does all the coding, and the others just coast. I found no changes in the marks (or in cheating prevalence); thus I surmise that more students learn to program on their own. Programming assignments and their rubrics (on the syllabus) are made available on the website of every course. I used Piazza for many years, and since 2017 have begun using iKamva which is based on the Sakai platform (see Figure 5 below). Now, especially with covid, online has become more than an online dumping ground for material. Yes, it still holds multiple forms of media; but it also contains a calendar, announcements, platform for online quizzes and assignment submission, marking and feedback; and much, much more.


ikamva.uwc.ac.za/portal/site/003bc5da-08f6-4d7e-ace9-c977291b6fda/tool/2a2ae6cd-8881-4310-8ece-d8b48bab4f36/ShowPage?sakai.tool.placement.id=2a2ae6cd-8...

Weekly Seminars > Week 4 Ethics

Guest lecture: **Prof. Andy Dearden: University Sheffield-Hallam, Interactive Systems Design**

Andy's slides

Required reading

Recommended reading

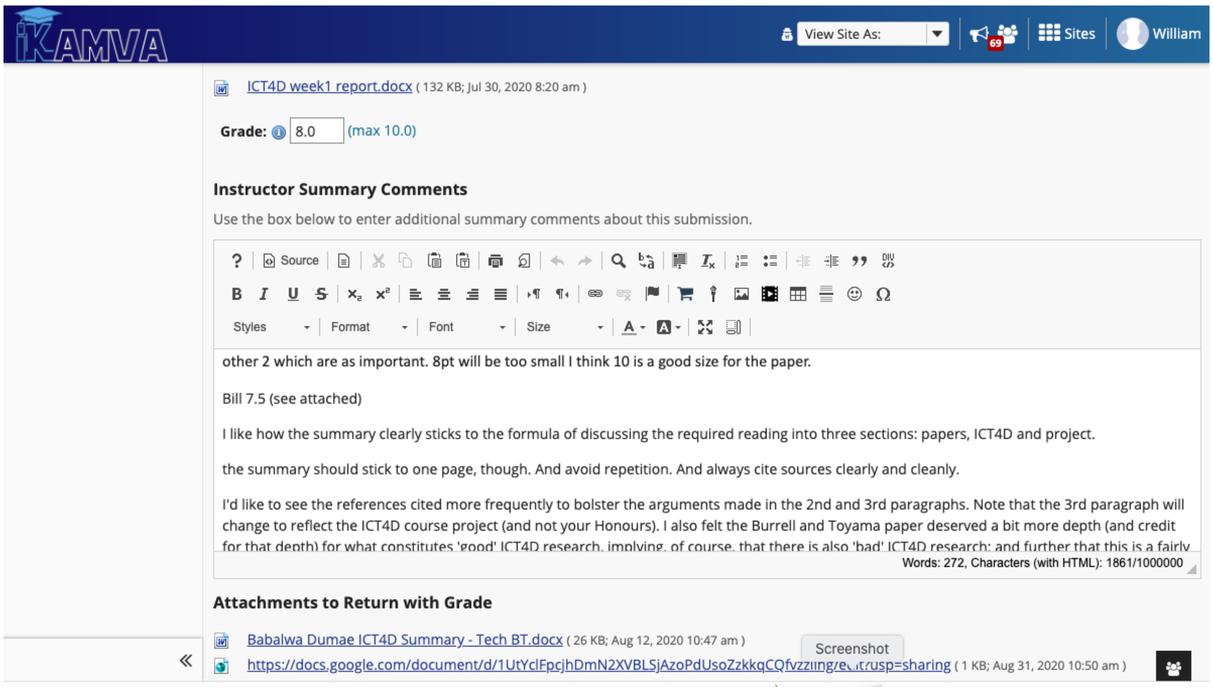
Summary: Ethics and intent

Paper v0: proposal

Figure 5: Online course management systems have become de rigour for SA universities. Note that my Honours ICT4D course had been online for 3 years already, and very little changed for covid. Other courses of mine had used Piazza before then, and since the late '90s, I only accepted programming assignment submissions digitally, and they were fully and automatically tested with scripting tools (first shellscript and later Python) before documentation was marked.

In brief, I code the assignment myself to exact specifications that I make available to students online, along with a server-based 'driver' that compares the output of my code to student code, given the same inputs. Course tutors validate my code to ensure that it is correct. The inputs are collected in the form of 'testfiles' that can contain up to 100's or 1000's of different forms of input, both good and bad, that the code must deal with. The principle is simple: there is no one way to code a given solution, as programming is as much an art as it is a science. However, the code must produce specific output given specific input (the science!). I only start to mark coding style and efficiency from 3rd year up. For all levels, however, we award marks based on various types of documentation: high level (module/class), low level (function/method) and code level; once a student solution ‘passes the driver’, i.e. produces exactly the same outputs as my solution, given the same inputs (although it can be implemented any way they like). At various times, I also award marks based on the use of source code control, debugging techniques (e.g. debug vs. trace), profiling (number of CPU ticks during execution) and even size of the executable (important for mobile and embedded applications); however, over the years I removed these requirements because students simply didn't do them – at UWC, the majority want 50% not 100%. This is a cultural mindset that I find extremely challenging to combat and change; and we are starting to win (there is anecdotal evidence of a culture shift happening in our department which I believe is mostly tied to the momentum of the postgrad programme, established in 2000).

At Honours level, I alternate programming exercises with written and verbal assignments in a ‘mini conference’ format. The process could be as follows: write a 2-page paper and give a 5 minute presentation to the class, then expand the paper and presentation based on feedback (from fellow students and myself) to 4 pages and a 10 minute talk with 5 minutes Q&A, respectively, and then a final version back to 2 pages and 5 minutes, respectively. I designed it this way to stress the integration of feedback into the process, and to contrast with the ‘single shot’ paper typically assigned in other courses. Because everyone reads everyone else’s paper, it also introduces the students to peer review. Note that for the 3rd year OS course, we also perform peer review between and within groups on program design, e.g. UML, and implementation (code review). These processes are prevalent in industry and the open source arena, and our students need to learn how it works; and get and give valuable feedback using the ‘sandwich’ method: 1) praise, 2) critique and 3) offer suggestions for improvement, i.e. a way forward. This process is iterated in order to factor feedback into successive efforts. And feedback is critical: that it be constructively critical, and timely (see Figure 6).



ICT4D week1 report.docx (132 KB; Jul 30, 2020 8:20 am)

Grade: 8.0 (max 10.0)

Instructor Summary Comments

Use the box below to enter additional summary comments about this submission.

other 2 which are as important. 8pt will be too small I think 10 is a good size for the paper.

Bill 7.5 (see attached)

I like how the summary clearly sticks to the formula of discussing the required reading into three sections: papers, ICT4D and project.

the summary should stick to one page, though. And avoid repetition. And always cite sources clearly and cleanly.

I'd like to see the references cited more frequently to bolster the arguments made in the 2nd and 3rd paragraphs. Note that the 3rd paragraph will change to reflect the ICT4D course project (and not your Honours). I also felt the Burrell and Toyama paper deserved a bit more depth (and credit for that depth) for what constitutes 'good' ICT4D research, implying, of course, that there is also 'bad' ICT4D research; and further that this is a fairly

Words: 272, Characters (with HTML): 1861/1000000

Attachments to Return with Grade

Babalwa Dumae ICT4D Summary - Tech BT.docx (26 KB; Aug 12, 2020 10:47 am)

https://docs.google.com/document/d/1UtYclFpcjhDmN2XVBLSjAzoPdUsoZzkkqCQfvzziing/e..t?usp=sharing (1 KB; Aug 31, 2020 10:50 am)

Figure 6: Feedback is crucial; not only for the student but for me to get a feel for the cadence of the course and its delivery and assessment. In the case of this feedback, it shows that multiple people review a student’s written submission for the Honours ICT4D course, thus emulating genuine peer review.

There is no final exam for any of my courses anymore. I must point out that continuous assessment as described above is much more time consuming than a single marking of a long final exam that does not, in my opinion, encourage the student to do anything at all except cram for a single assessment. That said, I am not opposed to combining continuous assessment with a final exam, and I did this from 1998-2002. However, after returning from sabbatical in 2005, I have avoided final exams as I feel that pure continuous assessment does a much better job with respect to situated and authentic teaching and learning, particularly in setting expectations for a course and its goals (to learn something by doing instead of just aiming to pass an exam) and also relieves me from the double burden of both continuous and final assessment. Unfortunately for me, this approach is somewhat non-traditional and not catered for by UWC's marks administration system, and I therefore must be 'creative' to fit pure continuous assessment into a system that does not actually allow for it (even today with 100% remote teaching and learning). However, many around me are now following in my footsteps since 2015-16 with #FMF, and since 2020 with covid. Note again, I started doing this, and mostly online, since the early 2000's.

Institutional, administrative and committee work on teaching and learning

Since 2010, I had volunteered every year to serve on the Faculty T&L committee. However, it was decided that it was more strategic for me to continue serving on the Faculty Higher Degrees committee to serve the department's needs and so as not to overload my administrative load so I may concentrate on maintaining the critical mass of research activities in our department. I finally managed to join the T&L committee in 2015, and really enjoyed the stimulation from other team members. That and attending T&L seminars forces me to reconsider and reshape my approach to T&L, and for that I am grateful. I was asked to step down from the T&L committee in 2019 after winning the Teaching Excellence award because that committee is considered a vehicle to win that award (which is not the case), and my methods challenge the plasticity of research over teaching (instead of them reinforcing one another) practised by most of my colleagues. Only recently, the T&L office asks me to give talks at their events, and has been well-received.

Scholarship of teaching and learning

As started above, my approach to T&L is authentic and situated; and iteratively evolves incrementally, aligned with experimental empirical Computer Science where we iteratively make small incremental improvements to algorithms and prototypes to achieve a given objective based on observable results (these are the version numbers you see associated with all types of software). I have co-authored an article based on 'learning by doing' in the African context, in terms of both courseware and research efforts, because our MSc system markedly differs from the course-based MSc prevalent in much of the world, several years ago with colleagues from UCT and the bridges.org NGO. The paper was accepted for a special issue in the Information Technology and International Development (ITID) journal, and then the special issue did not materialise. At some point, we should probably revise and submit the article locally to SACLA or SAICSIT.

My work with the Deaf community was featured as a case study in perhaps the world's most popular textbook on human computer interaction (see Interaction Design 3rd edition), the same textbook we use for our own HCI course. The case study, at least the extended online version, was essentially an adaptation of one of the case study appendices from my PhD.

I wish I had more time to contribute to the scholarship of T&L but I find myself unable to prioritise that given my heavy research load, e.g. an average of 12-15 postgrads per year, now more than half are PhD students. I have years and years of data, handwritten notes per course (in notebooks, one per course) of how processes have changed, to correlate with marks kept in spreadsheets since 1998. I tend to give a course for 5-7 years and then move on to something else. I insist on not teaching the same course for longer than that because I, too, need to grow, and the processes are easily transferable between courses.

Professionalism of teaching and learning

My engagement with T&L structures has mostly been through my peers in the department, and as from 2015, engagement with peers in the Science Faculty T&L committee. I was appointed Blended Learning Champion for the department by our DVC Academic. At one point, I asked to resign as e-learning champion because no one in my department other than myself was using any form of formal blended learning. It's only now, with a new HOD's edict that every single class should be on iKamva, that that has changed; especially with covid. And for that I am also grateful, and not just because it pushed me to move from Piazza to iKamva. I also use other forms of digital media to coordinate courses and tutors, e.g. WhatsApp. I am constantly consulting my colleagues about this or that assignment, lecture or demonstration technique. I also engage with my students formally by having them write a simple course evaluation at the end of each module; and more recently continuous evaluation to be able to react in agile fashion to ongoing and post-hoc course evaluations, e.g. having lecturers take an entire 1st year term class instead of alternating weeks, and in 2015 I even relented and gave in to student pressure (after 17 years!) for partial marks for a programming assignment that didn't work; albeit on my own terms: a program that works gives 50% and the documentation the remaining 50%, i.e. only 50% is possible if the code doesn't work; or quickly changing a scheduled lecture when a guest speaker can or cannot make it.

I have attended every Computer Science annual meeting since 1998 where we predominantly engage regarding the curriculum at both undergraduate and postgraduate levels. I am most involved with the postgraduate research, and even though I am only since 2012 in a senior position, I have been the most senior de facto researcher in my department with respect to volume of graduates, publications, funding and NRF rating. I also offered to take 1st year lectures from 2009-2015, which senior academics tend not to do. I enjoyed it immensely! I am an advocate of using technologies in learning, for hardware, e.g. multi-vendor equipment in the lab so that students learn principles rather than one vendor's approach; and for software, e.g. I use Facebook, LinkedIn, Google groups, WhatsApp and Piazza (and now iKamva) on a regular basis to conduct both course and research activities with students. I am a supporter of FOSS and encourage all of my postgraduate students to get involved with an active online FOSS community, e.g. the mesh potato project with www.villagetelco.org which has led to LibreRouter. To me, open source and open documentation, even open hardware, is the 'new' way for Computer Science, and prepares students for both advanced postgraduate studies and work in industrial, governmental and academic sectors. Our work with Zenzeleni was recognised by the Mozilla Foundation (we came 5th in the world in their Equal Rating Challenge!). In my opinion, we should also probably be moving toward open courseware, as well, e.g. the Open University programmes and Kahn Academy.

Evaluation of teaching and learning

All of my 3rd year and Honours level modules are externally evaluated, and I meet personally with examiners to discuss courses in detail, to measure up against myself annually, and also with respect to how courses are given and organised at examiners' respective universities, e.g. UCT, Stellenbosch and Georgia Tech. I also consistently conduct course evaluations with students by asking the same three questions: What did you like about the course? What did you dislike about the course? How would you change the course? (see the sandwich method?) I read these, and make corresponding small adjustments each year. I make an effort to read between the lines, e.g. when students complain about how hard assignments are, or how unfair a pop quiz is, I know I am doing the right thing because I receive email after several years thanking me for making my courses so difficult (sometimes I don't even have to wait that long), or for example when students complain about how I bring national politics into discussions about computing principles, e.g. socialist vs. despotic process scheduling in light of contemporary African politics, or even the use of social media in a politically responsible way, e.g. the Arab Spring or #FeesMustFall, and now with the challenges of remote learning covid. I encourage my students to engage with their world, in and out the classroom, and often talk and joke about politics during a lecture, yet more seriously engage them as to how computing can be used in the political and socio-economic realms of our lives.

If I could, I would only give a course for 3-5 years and then move on, and apply techniques to other source material. I would actually like to lecture courses I did not take as a student, to widen my foundation knowledge. Even when I do take a course for a long period of time, e.g. the operating systems courses, I try to avoid doing the same things in the same way every year. Yet instead of making big changes, I make little ones. This is the way of experimental and empirical Computer Science, to continually make small iterative changes to assess and improve.