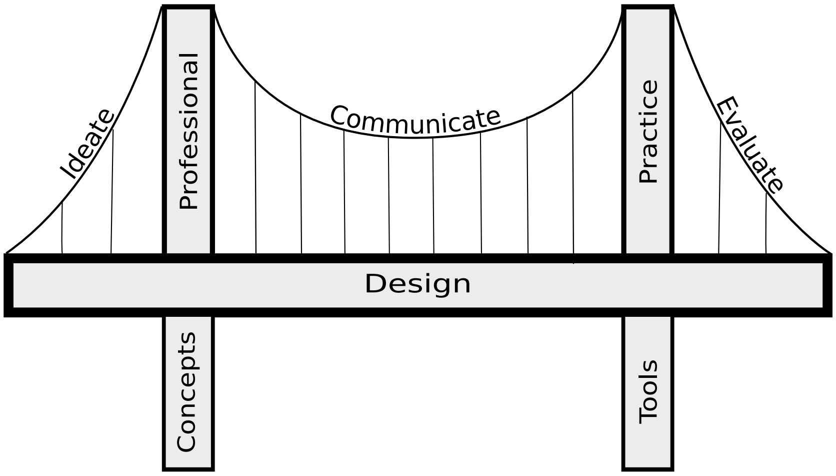
EDITED BY DEREK RAYSIDE

SE CAPSTONE DESIGN PROJECT HANDBOOK



UNIVERSITY OF WATERLOO

Compiled January 4, 2017

ACKNOWLEDGEMENTS:

- Prof Paul Ward initiated the SE Capstone Design Project, set up the overall structure, taught the first several offerings, and remains part of the instructional team.
- Profs Krzysztof Czarnecki, Charlie Clarke, Andrew Morton, and Patrick Lam are members of the instructional team.
- SE2014 student and class rep Michael Chang wrote an SE499 report on the SE Capstone Design Project and was involved in the creation of this handbook.
- SE2014 student and class rep Ming-Ho Yee has provided much valuable feedback and suggestions.
- Many students from recent graduating classes have shared their wisdom here for the benefit of future students.
- The chapter on Popular Project Ideas was developed in collaboration with Michael Kirkup from Velocity, where they also often see many of the same ideas repeated.

Contents

In which we do not arrive at a definitive answer but, instead, gain a better understanding of the question.

Design is what designers do. Who are designers? What do they do? What skills do they have?

Design is the output of the design process. What is that process? What general form does its output take?

A plan for arranging elements in such a way as to best accomplish a particular purpose. — Charles Eames

Don’t reinvent the wheel

List of Figures

Learning Objectives

The intended learning objectives of the Capstone courses are:

- To be proud of your project and team.
- To create positive change in the world. [§6.5]
- To think big thoughts (*i.e.*, to integrate and apply what you have learned in all of your other courses on a large team project).
- To make strategic decisions. [§4.2 + §4.3 + §4.4 + + §4.10]
- To apply professional practices. [§1.6]
- To select and apply appropriate abstractions in communicating complex projects. [§7]

Back where I come from we have universities — seats of great learning — where men go to become great thinkers. And when they come out, they think deep thoughts, and with no more brains than you have.
— The Wizard of Oz, 1939

Of course, in the twenty-first century, more women now go to (and graduate from) universities than men.

1.1 *Concept Map*

A *concept map* is an illustration that gives a holistic overview of a course (or a software system). It might be in the form of an abstract geometric representation, such as a mind map or software architecture diagram, or a visual metaphor (*e.g.*, Figure 1.1).

The concept map for the SE Capstone Design Project (Figure 1.1) uses the visual metaphor of a bridge as a classical engineering artifact for the public good. The foundations of the bridge are the *concepts* and *tools* that you have learned in your previous courses. But these compressive members alone are not enough to support good design: the superstructure of *professional practice*, and the tensile members of *ideation*, *communication*, and *evaluation* are also necessary. Your subjective experience of the compressive and tensile members will be different, which is why they have been depicted in distinct modes.

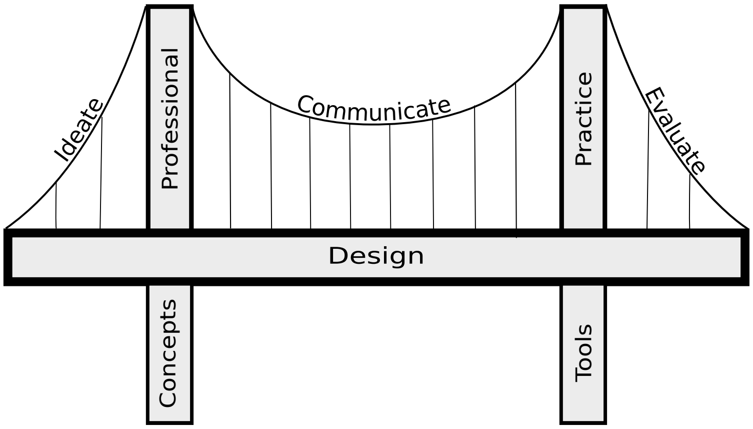


Figure 1.1: Suspension bridge concept map for SE Capstone Design Project. Design is supported by both compressive and tensile members.

1.2 *Formative and Summative Assessments*

Most engineering courses focus on *summative assessments* of linear thinking: that is, critical assessments of the student's learning and thinking, from which the grade is derived.

In the capstone courses we will also do a lot of *formative assessments:* exercises designed to give you feedback on your thinking, but with little or no grade associated. Often there might be a small nominal grade to encourage you to participate in the exercise. For example, in SE491 we will have a practice symposium day for you to give your presentation and receive feedback.

Some of the formative assessments are intended to encourage your lateral thinking: to help you examine your work from a different perspective.

1.3 *Grading Philosophy*

HOLISTIC EVALUATION. The exact mapping between project status indicators (*e.g.*, functionality, specification, productivity, plan, *etc.*) and the grade is evaluated holistically. In other words, there is no *a priori* marking scheme breakdown for these components. Capstone design projects differ significantly in their technical challenges, level of difficulty, domains, and objectives. Holistic evaluation considers all factors, and allows flexibility to reward focused excellence, new ideas, and new approaches.

GRADING CRITERIA. The expected criteria for an A grade at each demo is described on the timeline above. These criteria are to be interpreted as holistic guidelines: a project might appear to meet these criteria yet earn less than an A; similarly, a project that does not meet all of these criteria might earn an A. The difficulty of the project is a major factor moderating the criteria: the criteria might be relaxed for harder projects, strengthened for easier projects.

C is defined as meeting the criteria of the previous milestone. D is considered as less than the previous milestone. B is between A and C.

LINES OF CODE. It is important to not become overly focused on the lines of code metric of productivity. Many factors may moderate the interpretation of this metric, including difficulty of the problem, nature of the problem, choice of language, *etc.* Lines of code is just one indicator of a team's productivity.

The lines of code metric is intended to include not just the evolutionary prototype, but also all exploratory prototypes, testing and other infrastructure, any files written in a language designed by the team, *etc.* In other words, the metric is intended to include just about everything written by hand and processed by machine.

To paraphrase Churchill, it is the worst metric we have except for all the others.

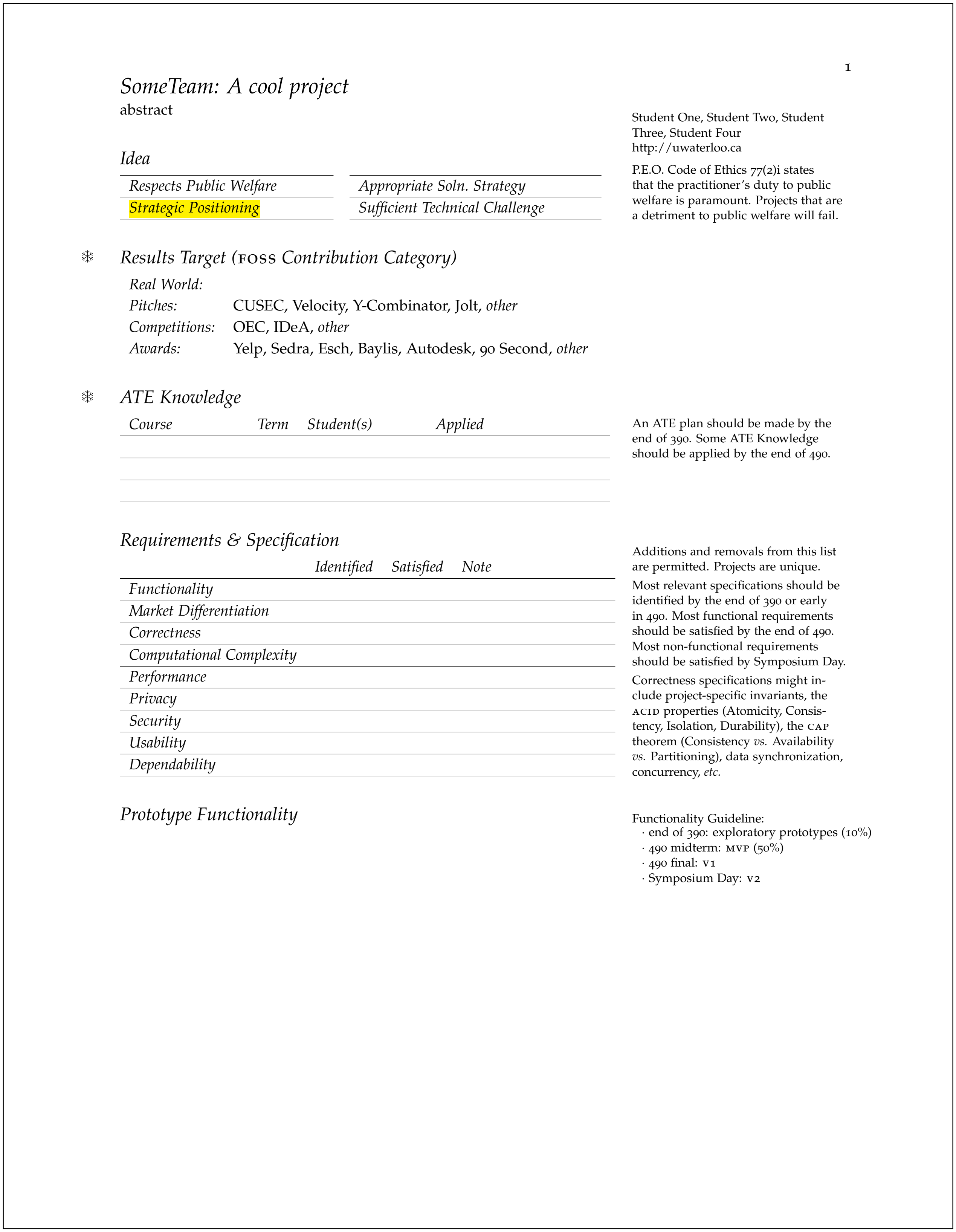
Lines of code should be measured with a standard tool for this purpose such as CLOC or SLOCCOUNT — not WC.

For example, team TSK from SE2015 had an award winning project with relatively few lines of code because their contribution was primarily in a series of user studies.

1.4 *Project Status Sheets*

From the end of SE390 through to Symposium Day the project status sheet (sample on next two pages) will be used to characterize the current status of the project. The status sheet characterizes the project at a point in time but does not include grading criteria, since those change over time. The criteria are given in the timeline table above.

The status sheet will include, at the end, the results rubric selected by the team (see §6.5).



1

SomeTeam: A cool project

abstract

Student One, Student Two, Student Three, Student Four
http://uwaterloo.ca

Idea

Respects Public Welfare	*Appropriate Soln. Strategy*
Strategic Positioning	*Sufficient Technical Challenge*

P.E.O. Code of Ethics 77(2)i states that the practitioner's duty to public welfare is paramount. Projects that are a detriment to public welfare will fail.

❄ *Results Target (FOSS Contribution Category)*

Real World:	
Pitches:	CUSEC, Velocity, Y-Combinator, Jolt, *other*
Competitions:	OEC, IDeA, *other*
Awards:	Yelp, Sedra, Esch, Baylis, Autodesk, 90 Second, *other*

❄ *ATE Knowledge*

Course	*Term*	*Student(s)*	*Applied*

An ATE plan should be made by the end of 390. Some ATE Knowledge should be applied by the end of 490.

Requirements & Specification

	Identified	*Satisfied*	*Note*
Functionality			
Market Differentiation			
Correctness			
Computational Complexity			
Performance			
Privacy			
Security			
Usability			
Dependability			

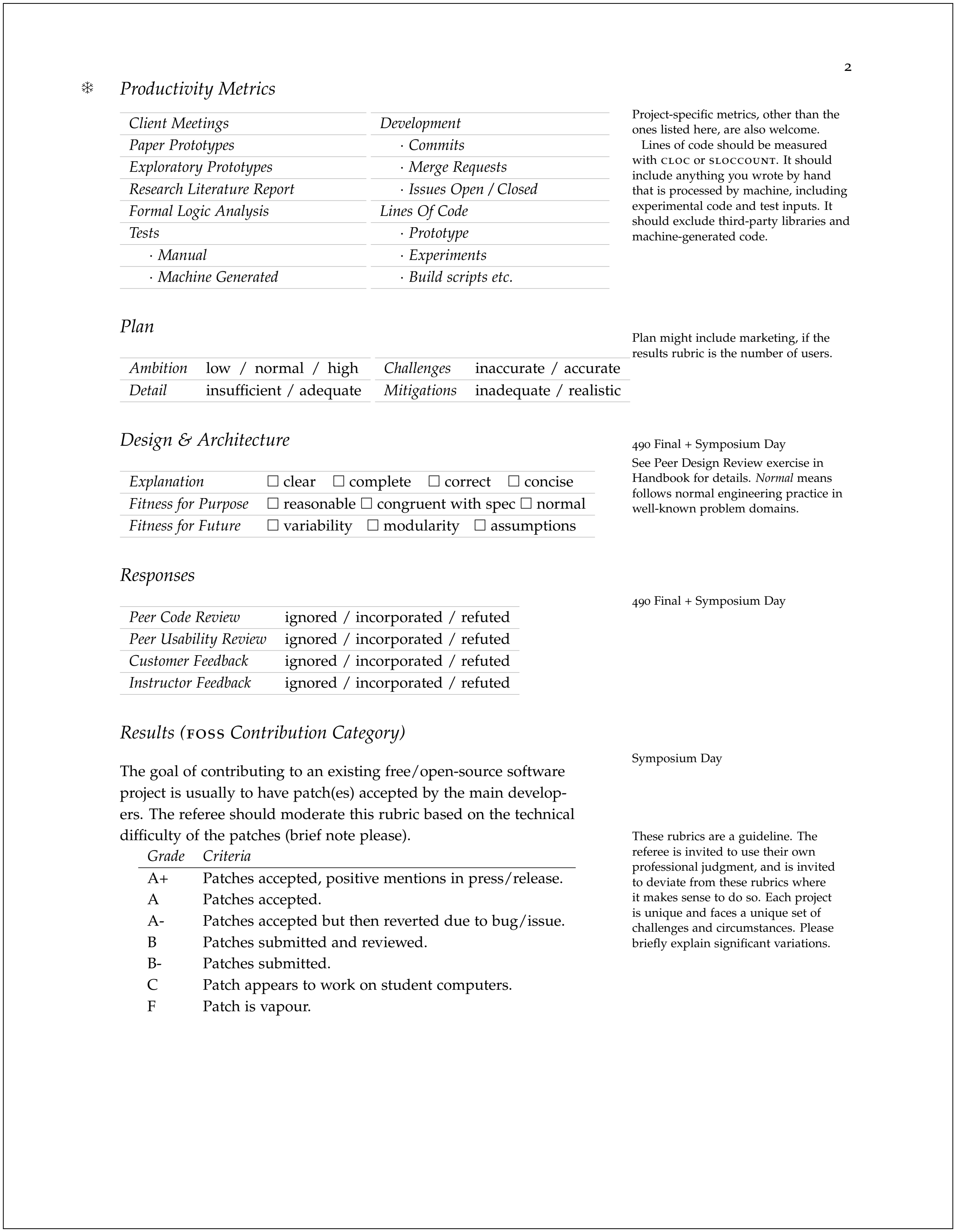
Additions and removals from this list are permitted. Projects are unique.

Most relevant specifications should be identified by the end of 390 or early in 490. Most functional requirements should be satisfied by the end of 490. Most non-functional requirements should be satisfied by Symposium Day.

Correctness specifications might include project-specific invariants, the ACID properties (Atomicity, Consistency, Isolation, Durability), the CAP theorem (Consistency *vs.* Availability *vs.* Partitioning), data synchronization, concurrency, *etc.*

Prototype Functionality

Functionality Guideline:
· end of 390: exploratory prototypes (10%)
· 490 midterm: MVP (50%)
· 490 final: v1
· Symposium Day: v2



❄ *Productivity Metrics*

Client Meetings	*Development*
Paper Prototypes	· *Commits*
Exploratory Prototypes	· *Merge Requests*
Research Literature Report	· *Issues Open / Closed*
Formal Logic Analysis	*Lines Of Code*
Tests	· *Prototype*
· *Manual*	· *Experiments*
· *Machine Generated*	· *Build scripts etc.*

Project-specific metrics, other than the ones listed here, are also welcome.

Lines of code should be measured with CLOC or SLOCCOUNT. It should include anything you wrote by hand that is processed by machine, including experimental code and test inputs. It should exclude third-party libraries and machine-generated code.

Plan

Plan might include marketing, if the results rubric is the number of users.

Ambition	low / normal / high	*Challenges*	inaccurate / accurate
Detail	insufficient / adequate	*Mitigations*	inadequate / realistic

Design & Architecture

490 Final + Symposium Day

See Peer Design Review exercise in Handbook for details. *Normal* means follows normal engineering practice in well-known problem domains.

Explanation	□ clear □ complete □ correct □ concise
Fitness for Purpose	□ reasonable □ congruent with spec □ normal
Fitness for Future	□ variability □ modularity □ assumptions

Responses

490 Final + Symposium Day

Peer Code Review	ignored / incorporated / refuted
Peer Usability Review	ignored / incorporated / refuted
Customer Feedback	ignored / incorporated / refuted
Instructor Feedback	ignored / incorporated / refuted

Results (FOSS Contribution Category)

Symposium Day

The goal of contributing to an existing free/open-source software project is usually to have patch(es) accepted by the main developers. The referee should moderate this rubric based on the technical difficulty of the patches (brief note please).

Grade	*Criteria*
A+	Patches accepted, positive mentions in press/release.
A	Patches accepted.
A-	Patches accepted but then reverted due to bug/issue.
B	Patches submitted and reviewed.
B-	Patches submitted.
C	Patch appears to work on student computers.
F	Patch is vapour.

These rubrics are a guideline. The referee is invited to use their own professional judgment, and is invited to deviate from these rubrics where it makes sense to do so. Each project is unique and faces a unique set of challenges and circumstances. Please briefly explain significant variations.

1.5 Course Calendar Descriptions

These courses have a few differences from the standard course format: all work is done in groups; there are no written tests; there is no textbook.

1.5.1 SE390

Students undertake a substantial customer-driven group project as part of the SE 390/490/491 design-project sequence covering all major phases of the software-engineering lifecycle. Lectures describe expectations and project-planning fundamentals. Students form groups, decide on a project concept, complete a project-approval process, develop high-level requirements for the project, perform a risk assessment, develop a test plan, and complete a first-iteration prototype. Social, legal, and economic factors are considered.

1.5.2 SE490

Continuing from SE390, students undertake a substantial customer-driven group project. Project groups establish and maintain project control processes, delivering a series of iterations on their SE390 prototype. Adaptive methods are encouraged and supported.

1.5.3 SE491

Final implementation, testing, and communication of the design project started in SE390. Technical presentations by groups. Analysis of social, legal, and economic impacts. Final release of the project. Project retrospective.

1.6 Professional Practice

The professional practices discussed here, that you are expected to follow, range from software-engineering discipline-specific practices to general professional engineering practices.

Organization of Work. As you do on your co-op work terms, your team is expected to use version control, issue tracking, automated builds and deployment, and automated test harnesses.

Evidence of Design's Fitness for Purpose. Professional engineers provide some evidence of their design's fitness for purpose. In traditional branches of engineering this evidence is usually in the form of a mathematical model. That form of evidence is becoming more common in software engineering,[1] but is not yet the norm.

Common practice in software engineering is to demonstrate that the design is *normal*[2] (*i.e.*, not *radical*), accompanied with functional testing and perhaps usability testing (if applicable). The use of mechanized program analysis tools is also becoming more common.

Giving and Receiving Constructive Criticism. The engineering profession progresses via honest criticism. By understanding the successes and failures of those who came before us we build a community of practice with established design norms and rules. Participating in this professional community means, in part, learning to both give and receive balanced constructive criticism.

Time Management. Part of being a professional is managing your own time and being able to plan for large deadlines that are far in the future. The Capstone project is likely the largest project over the longest time period of any that you have undertaken. There are not many intermediate deadlines in these courses. Many students find that challenging. Every year students ask instructors for more intermediate deadlines. One of the tough lessons through this project is learning to set your own intermediate deadlines and manage your own time on a large, long-term, open-ended project. If the courses had more intermediate deadlines, it would detract from this important professional learning objective.

Team Work. All project work completed for the SE Capstone courses is considered to have been completed by the group as a whole. All group members should attempt to contribute equally to the project, including participating on each deliverable. An appropriate division of labour can be discussed with the instructor. All group members will be assigned the same grades *unless* someone lodges a complaint with the instructor.

[1] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. How Amazon web services uses formal methods. *Communications of the ACM*, 58(4):66–73, 2015

[2] Michael A. Jackson. The Name and Nature of Software Engineering. In Egon Börger and Antonio Cisternino, editors, *Advances in Software Engineering: Revised Lectures of Lipari Summer School 2007*, volume 5316 of *Lecture Notes in Computer Science*, pages 1–38. Springer-Verlag, 2008; Edward W. Constant. *The Origins of the Turbojet Revolution*. The Johns Hopkins University Press, 1980; and Walter G. Vincenti. *What Engineers Know and How They Know It: Analytical Studies from Aeronautical History*. The Johns Hopkins University Press, 1993

1.6.1 *Culture of Trust and Honesty*

Some students — and indeed other instructors — have pointed out that it is possible to be dishonest in the capstone courses. Yes, it is. And it is similarly possible to be dishonest in other parts of your life.

SE students have demonstrated honour and integrity in the Capstone courses in a number of ways. At the extreme, one student even attempted to transfer out of SE to CS after 4A because he felt he had not contributed adequately to his team's success during SE490.

The findings in the educational literature suggest that no system of rules and enforcement can eliminate all dishonesty. You may be surprised to discover that the educational literature finds that fostering a culture of trust and honesty is the most effective way to reduce dishonest behaviour.

More importantly, these courses are part of your professional development. As a professional engineer, you must be trustworthy. Society has a higher expectation of honesty from professionals such as engineers, lawyers, accountants, and doctors, and that is part of why these groups are granted elevated status in law.

The Professional Engineers of Ontario Code of Ethics [3] contains at least two mentions of honesty:

- *(1):* 'It is the duty of a practitioner to the public, to the practitioner's employer, to the practitioner's clients, to other members of the practitioner's profession, and to the practitioner to act at all times with,'
  - *iii:* 'devotion to high ideals of personal honour and professional integrity'
- *(8):* 'A practitioner shall maintain the honour and integrity of the practitioner's profession'

We are confident in your professional conduct in this course and elsewhere in your career.

1.7 *Use of pre-existing tools, libraries, frameworks,* etc.

Students are permitted to use any pre-existing software, tools, libraries, frameworks, *etc.*, appropriate to their project and permitted by licensing. Students should make clear to the instructor which parts of the project are original, which are derivatives of pre-existing works, and which are configurations of pre-existing works. Grades are based solely on the work done by the students for the course, which may include configurations of and derivatives of pre-existing works, but does not include the pre-existing work itself.

[3] Revised Regulations of Ontario 1990, Regulation 941, section 77. http://peo.on.ca/index.php?ci_id=1815&la_id=1#Section77

1.8 SE Curriculum Committee Intended Learning Outcomes

The SE Curriculum Committee has defined a range of attributes that it hopes graduates of the program will have. The Committee has further mapped these attributes back to individual courses that are supposed to develop those skills, and then monitors the outcomes. This monitoring process is part of a high-level feedback-control loop, where The Committee periodically makes adjustments to the curriculum and program to improve outcomes.

Copy in the rest of the learning outcomes

SE390

- Investigation:
  - Conduct (Create) investigations of complex computing problems by methods that include: (a) problem identification, conceptualization, and abstraction; (b) background research; (c) appropriate experiments; (d) data analysis and interpretation; and (e) information synthesis, in order to reach substantiated and valid conclusions.

SE490

- Investigation: Apply independent research to complement course materials.
- Design: Create experimental and evolutionary prototypes.
- Implementation: Understand programming in the large.
- Individual Work:
  - Manage (Apply) own time.
  - Assume (Apply) responsibility for own work.
- Project Management: Make decisions (Evaluate) under uncertainty, including project risks.
- Tools:
  - Apply configuration management tools for non-trivial software projects.
  - Apply management tools to software project schedules and deliverables.
- Professionalism: Apply relevant software standards and best practices.

SE491

- Design: Evaluate designs for compliance with behavioural and non-behavioural requirements such as for health, safety, economic, environmental, ethical, legal, and social issues by applying tactics for dealing with non-functional requirements.

- Individual: Function (Evaluate) effectively as an individual in a team.
- Communication Skills: Make (Create) effective oral technical presentations.
- Professionalism:
  - Remember professional societies relevant to software engineering.
  - Understand intellectual property rights with regard to software, e.g., copyright, utility patents, design patents, trade marks, license agreements, trade secrets.
- Impact of Engineering on Society and the Environment:
  - Evaluate cultural, economic, health, safety, and social implications of software.
  - Understand privacy laws and their impact on software requirements and data collection.
  - Understand software warranties and liabilities.

1.9 *CEAB*

Add how the SE Curriculum Committee categorizes the Capstone courses for accreditation purposes.

The Canadian Engineering Accreditation Board (CEAB) is a standing committee of Engineers Canada[4] that accredits undergraduate engineering programs in Canada. A number of the requirements in CEAB's Accreditation Criteria and Procedures[5] may be fulfilled by a fourth-year design (capstone design) project.

Graduates of an engineering program should possess the "graduate attributes" described in Section 3.1 of CEAB's Accreditation Criteria:

3.1.1 **A knowledge base for engineering:** Demonstrated competence in university level mathematics, natural sciences, engineering fundamentals, and specialized engineering knowledge appropriate to the program.

3.1.2 **Problem analysis:** An ability to use appropriate knowledge and skills to identify, formulate, analyze, and solve complex engineering problems in order to reach substantiated conclusions.

3.1.3 **Investigation:** An ability to conduct investigations of complex problems by methods that include appropriate experiments, analysis and interpretation of data, and synthesis of information in order to reach valid conclusions.

[4] Engineers Canada is a national organization of the 12 provincial and territorial associations that regulate engineering and license engineers in Canada.

[5] Canadian Engineering Accreditation Board. Accreditation criteria and procedures, 2013. URL http://www.engineerscanada.ca/sites/default/files/sites/default/files/accreditation_criteria_procedures_2013.pdf. Retrieved winter 2014

3.1.4 **Design:** An ability to design solutions for complex, open-ended engineering problems and to design systems, components or processes that meet specified needs with appropriate attention to health and safety risks, applicable standards, and economic, environmental, cultural and societal considerations.

3.1.5 **Use of engineering tools:** An ability to create, select, apply, adapt, and extend appropriate techniques, resources, and modern engineering tools to a range of engineering activities, from simple to complex, with an understanding of the associated limitations.

3.1.6 **Individual and team work:** An ability to work effectively as a member and leader in teams, preferably in a multi-disciplinary setting.

3.1.7 **Communication skills:** An ability to communicate complex engineering concepts within the profession and with society at large. Such ability includes reading writing, speaking and listening, and the ability to comprehend and write effective reports and design documentation, and to give and effectively respond to clear instructions.

3.1.8 **Professionalism:** An understanding of the roles and responsibilities of the professional engineer in society, especially the primary role of protection of the public and the public interest.

3.1.9 **Impact of engineering on society and the environment:** An ability to analyze social and environmental aspects of engineering activities. Such ability includes an understanding of the interactions that engineering has with the economic, social, health, safety, legal, and cultural aspects of society, the uncertainties in the prediction of such interactions; and the concepts of sustainable design and development and environmental stewardship.

3.1.10 **Ethics and equity:** An ability to apply professional ethics, accountability, and equity.

3.1.11 **Economics and project management:** An ability to appropriately incorporate economics and business practices including project, risk, and change management into the practice of engineering and to understand their limitations.

3.1.12 **Life-long learning:** An ability to identify and to address their own educational needs in a changing world in ways sufficient to maintain their competence and to allow them to contribute to the advancement of knowledge.

Section 3.4.4.3 of CEAB's Accreditation Criteria requires "A minimum of 225 AU in engineering design", which is "creative, iterative, and open-ended process" that "integrates mathematics, natural sciences, engineering sciences, and complementary studies in order to develop elements, systems, and processes to meet specific needs." Solutions may be constrained by standards or legislation, relating to "economic, health, safety, environmental, societal or other interdisciplinary factors". In particular, Section 7 of the CEAB's Interpretive statement on licensure expectations and requirements requires 225

AU of engineering design be taught by faculty that are licensed to practice engineering in Canada.

Section 3.4.4.4 of CEAB's Accreditation Criteria requires that "the engineering curriculum must culminate in a significant design experience conducted under the professional responsibility of faculty licensed to practise engineering in Canada, preferably in the jurisdiction in which the institution is located. The significant design experience is based on the knowledge and skills acquired in earlier work and it preferably gives students an involvement in team work and project management."

Section 3.4.4.5 of CEAB's Accreditation Criteria requires that "Appropriate content requiring the application of modern engineering tools must be included in the engineering sciences and engineering design components of the curriculum."

Section 3.5.5 of CEAB's Accreditation Criteria requires that "Faculty delivering curriculum content that is engineering science and/or engineering design are expected to be licensed to practise engineering in Canada, preferably in the jurisdiction in which the institution is located. In those jurisdictions where the teaching of engineering is the practice of engineering, they are expected to be licensed in that jurisdiction." In particular, Section 4 (b.) of the CEAB's Interpretive statement on licensure expectations and requirements notes that faculty members who have applied for professional engineering licensure or engineer-in-training status are not compliant for the teaching of engineering design with respect to Sections 3.4.4.3 and 3.5.5.

Section 4.4 of CEAB's Accreditation Criteria notes that an accreditation visit provides the opportunity for activities including:

> e. a review of recent examination papers, laboratory instruction sheets, student transcripts (anonymous, if necessary), student reports and theses, models or equipment constructed by students and other evidence of student performance.

1.10 CIPS

The Computer Science Accreditation Council (CSAC) is a body established by the Canadian Information Processing Society (CIPS) that accredits undergraduate programs in computer science. A number of the requirements in CSAC's Accreditation Criteria[6] may be fulfilled by a fourth-year design (capstone design) project.

Section 4.1 of CSAC's Accreditation Criteria define "Graduate Attributes", which describe what graduate of a computer science or software engineering program should know and be able to do.

[6] Computer Science Accreditation Council. Accreditation criteria for computer science, software engineering and interdisciplinary programs, August 2011. URL http://www.cips.ca/sites/default/files/CSAC_Criteria_2011_v1.pdf. Retrieved winter 2014

A graduate of a computer science or software engineering program must be able to:

1. Demonstrate Knowledge: Competently apply knowledge in
   a) software engineering,
   b) algorithms and data structures,
   c) systems software,
   d) computer elements and architectures,
   e) theoretical foundations of computing,
   f) discrete mathematics, and,
   g) probability and statistics.
2. Analyse and Solve Problems: Use appropriate knowledge and skills, including background research and experimentation, to identify, investigate, abstract, conceptualize, analyse, and solve complex computing problems, in order to reach substantiated conclusions.
3. Design Software and Systems: Design and evaluate solutions for complex open-ended computing problems, and design and evaluate systems, components, or processes that meet specified needs with appropriate consideration for public health and safety, as well as economic, cultural, societal, and environmental considerations
4. Use Appropriate Resources: Create, select, adapt and apply appropriate techniques, resources, and modern computing tools to complex computing activities, with an understanding of their strengths and limitations.
5. Work Individually and in a Team: Function effectively as an individual and as a member or leader in diverse teams and in multidisciplinary settings
6. Communicate Effectively. Communicate with the computing community and with society at large about complex computing activities by being able to comprehend and write effective reports, design documentation, make effective presentations, and give and understand clear instructions
7. Act Professionally. Act appropriately with respect to ethical, societal, environmental, health, safety, legal, and cultural issues within local and global contexts, and with regard to the consequential responsibilities relevant to professional computing practice.
8. Be Prepared for Life-Long Learning: Learn new tools, computer languages, technologies, techniques, standards and practices, as well as be able to identify and address their own educational needs in a changing world in ways sufficient to maintain their competence and to allow them to contribute to the advancement of knowledge.
9. Demonstrate Breadth of Knowledge. Possess knowledge in areas other than computer science and mathematics so as to be able to communicate effectively with professionals in those fields.

Section 6.0 of CSAC's Accreditation Criteria requires "good students", as demonstrated by (among other indicators) "prizes and scholarships awarded", and "student's satisfaction with their program and progress as assessed through questionnaires and interviews".

Section 7.1 of CSAC's Accreditation Criteria requires "evidence that Graduate Attributes have been met", including "mappings from course-level objectives to graduate attributes" and "rubrics for assignments and tests indicating which graduate attributes are being assessed".

Section 7.3.6 of CSAC's Accreditation Criteria requires "Significant Design Experience":

> Students graduating from an accredited program should have had the chance to develop a complete significant system, or make a major modification to an existing system, at some point in their studies, whether it be in course projects, a final 4th-year project, or an internship or in some other manner. This design experience should be open-ended in the sense that there is "no right answer," and should enable the student to integrate their knowledge from most, if not all, of the areas of computer science listed in Section 7.3.1, as well as knowledge of mathematics, domain knowledge and, where appropriate, with consideration for economics, societal issues, safety, etc.

Section 7.6 of CSAC's Accreditation Criteria requires "Non-Trivial Problem Solving in Teams". Students are expected to identify objectives and criteria, create and analyse alternative solutions, select, implement, test, and evaluate a solution, and document and communicate their work.

Section 7.7 of CSAC's Accreditation Criteria requires "Written and Oral Communication Skills". Students should be taught to "practice collecting information through reading and listening", "assemble the information for various audiences", and "present information both verbally and in writing".

1.11 Course Design Rationale

An Agile Approach. The Software Engineering Capstone Design Project endeavours to be relatively agile:

- Allow students to focus on writing great software.
- Give students freedom to pivot their projects.
- Low administrative overhead. Keep written reports short — ideally so they can be completed during class time. Try to have no 'homework' except the core software development.
- Encourage agile development practices.

This kind of approach gives the best students the freedom to excel. The danger of this kind of approach is that the lower-end students might not get enough structure, and might get a higher grade than they deserve. We have been working towards getting more accurate grade assessment for the low-end, and providing greater structure without creating unnecessary impediments and friction.

Class Time & Activities. There is one class meeting/lecture per week, two hours long. Relatively little of this time is spent in didactic lecture. Class time is generally focused on group activities, such as providing peer feedback, or critiquing videos of past student presentations. We have found that a single long meeting/lecture is better for activities than having multiple short meetings/lectures.

Temporal Organization. Capstone design projects are organized in a variety of different ways. Typically they span two, and sometimes three, academic terms. At Waterloo, due to the co-op programme, these academic terms might be adjacent in time, or they might be separated by co-op work terms.

The Software Engineering Capstone project described in this handbook spans three academic terms interspersed by two co-op work terms, for a total of twenty calendar months. This is a very long elapsed time for an academic project — perhaps the longest Capstone Project at Waterloo.

In ECE, the capstone project spans two academic terms over twelve calendar months (*i.e.*, interspersed with one work term). In System Design, the capstone project spans two academic terms over eight calendar months (*i.e.*, two adjacent academic terms).

Reporting. The Software Engineering Capstone Design Project is perhaps unique amongst the Waterloo Engineering Capstone Design Projects in its focus on oral, rather than written, reporting.

Most, if not all, of the other Engineering Capstone Projects require extensive written reports (50+ pages). Much staff time is then devoted to reading and evaluating these reports. Some students perceive that they get little benefit from this massive investment of staff time: the feedback comes too late for them to take action on. Moreover, because of the time it takes to mark the reports, they often must be submitted 4–6 weeks in advance of Symposium Day, and some students feel that therefore the report does not adequately capture their final design.

Systems Design at Waterloo has an oral exam, in which the examination panel asks the students questions. In Software, we have the students give a presentation, with time for questions afterwards.

Expectations & Evaluation. Largely due to the extended timeline, but also to the nature of software, we expect students to have projects that work, and that are actually used by people outside of the classroom.[7] A shorter timeline or a different discipline would probably require adjusting expectations.

Staff/Student Ratio. A cohort of Software Engineering students has typically been around 80 students, resulting in about 20 groups, usually of four students each. One instructor can reasonably manage 20 groups and give meaningful feedback and coaching.

se will be growing to around 120 students in 30 groups.

Many of the other Engineering programmes have 15-20 groups per cohort; some have multiple cohorts. ece is significantly larger, with 50-70 groups per year.

Teaching Load. The way the teaching load is defined in Software is a bit odd: se390 counts as a full teaching credit, whereas se490 and se491 are counted as half-teaching credits. It has been said that the original intent was that one instructor would teach all three courses and therefore the average credit would be $\frac{2}{3}$. In practice, due to logistical concerns, the different courses are taught by different instructors. There is also a service task associated with being the capstone coordinator, which is often taken by one of the instructors.

Group Size. The ceab mandates that a 'group' is at least three students. In Software Engineering our most common team size is four, with some threes, some fives, and rarely a six.

In Mechanical Engineering at Waterloo it is not uncommon to have groups of 6–8 students. Groups in ece and Software rarely, if ever, get to this size.

Students are sometimes permitted to satisfy the group-work re-

[7] The concept of 'usage' here is broad. For example, students might parallelize an algorithm and show that it performs better than the previous sequential algorithms. In such a case, no person outside of the class would have necessarily actually interacted with their code, but they have created knowledge that could be useful to someone outside the classroom.

quirement through collaboration with outside parties. For example, in Mechanical Engineering, students have been been permitted to work on an individual Capstone project (from a course enrolment perspective) when that individual is working with one of the established extra-curricular teams such as Midnight Sun (solar car) or the Alternative Fuel Vehicle.

Although there are no equivalent established Software-related extra-curricular engineering design teams, we have on occaision permitted Software Engineering students to collaborate with non-engineering students (*e.g.*, core Computer Science students). Sometimes these core Computer Science students do want to be enrolled in the Software Engineering Capstone Courses, and sometimes they do not.[8]

Cross-Disciplinary Teams. The Faculty of Engineering explicitly encourages cross-disciplinary teams, and has a number of mechanisms to support this.

If the team is dominantly from one discipline but has a single student from another discipline, then that outlier student might enrol in the courses of the dominant discipline. For example, in the SE2016 cohort there is a team with one student from Civil Engineering — that student is enrolled in the Software Engineering Capstone courses, and for him they are counting in place of the Civil Engineering courses he would have normally taken.

If the team is a mixture of more than two disciplines, or there is no dominant discipline, or if for some logistical or other reason it does not make sense to enrol the outlier student in the dominant discipline's courses, then the entire team can be enrolled in the General Engineering Capstone courses. The Faculty of Engineering allocates an instructor to teach (and evaluate) these cross-disciplinary teams.

[8] A reason that they cite for not wanting to be enrolled in the course is to avoid paying extra tuition.

1.12 *Capstone Compatability with Exchange Study Programmes*

In each cohort there is usually at least one student who will be off campus for one of the capstone course terms, often on an international exchange study programme, but sometimes for health or other reasons. This can be accommodated in a number of ways.

Remote Participation. The capstone courses are all team-oriented. It is possible to participate in a team remotely. This strategy can be a bit more challenging for se390 because teams are usually formed in se390. So the student would need to be part of a cohesive team before se390 begins. For se490 and se491 this is the preferred option.

Course Substitution. For se390 we will accept ece390 as a substitute if the student is not able to take se390 in the usual way.

For se490 and se491 it is possible to take the analogous courses in another UW engineering programme, although this has historically occurred due to technical interests rather than scheduling reasons.

Your exchange university might also have a project course that could serve as a substitute.

se499. In some cases a student can take an independent study (se499) credit as a substitute for one of the se capstone courses. This is arranged on a case-by-case basis.

Design Theory

To design is to express one's intent about the way something should be — most commonly some kind of physical or conceptual artefact. Our exploration of design begins with a brief overview of the design disciplines: who are designers? what do they do? what skills do they have? First we will focus on what designers in all fields share in common. Subsequently, we will focus on the particularities of software design.

Within software engineering the word 'design' is used in a variety of ways. Sometimes it is a shorthand for user interface design. Sometimes it means low-level details in contrast to the big picture of 'architecture'. We use the word design in its most general sense, which encompasses architecture and user interface design and all of the other decisions that go into a software system.

2.1 Linear and Lateral Thinking

Most (if not all) of your engineering courses have focused on *linear* thinking skills: applying logic and mathematics in a step-by-step process to derive the answer. Linear thinking skills are essential for engineers. You are selected out of highschool for your demonstrated linear thinking ability, and then UW spends four years honing and training those skills. An engineer without linear thinking skills is not an engineer.

Good *design* requires good linear thinking skills. A designer must know what can be built and how to build it.

Great design also requires *lateral*[1] thinking. Lateral thinking involves making intuitive, creative, potentially non-obvious leaps; it involves re-imagining the problem from a different perspective. Lateral thinking will be discussed more in §2.

2.2 Who are Designers? The Design Disciplines

Moggridge[2] relates various design disciplines in the following two-

[1] Edward de Bono. *New Think: The Use of Lateral Thinking*. Basic Books, New York, 1967

[2] Bill Moggridge. *Designing Interactions*. The MIT Press, Cambridge, Mass., 2007

dimensional space, with one axis ranging from the human and subjective to the technical and objective, and the other axis ranging from physical artefacts to conceptual artefacts.

Physical Design

Human & Subjective

industrial		mech. eng.	
graphic	ergo	production eng.	physics
web	H.C.I.	hardware	CS
interaction		software	

Technical & Objective

Conceptual Design

Figure 2.1: Contextualization of design disciplines by Moggridge [33].

Moggridge uses the term 'digital', which I have switched to 'conceptual' at the suggestion of Daniel Jackson.

2.3 The Core Skills of a Designer

Moggridge[3] identifies the following five core skills for all designers:

a. To synthesize a solution from all of the relevant constraints.
b. To frame, or reframe, the problem and objective.
c. To create alternatives.
d. To select from those alternatives.
e. Prototyping.

The last two of these skills exercise primarily linear thinking. The middle two exercise primarily lateral thinking. The first skill requires both linear and lateral thinking.

2.4 Core Skill: Creating Alternatives

There are many ways to create alternatives, to have ideas. Many people have theorized or studied how creative thought happens. This section discusses some of that work, to stimulate you in your quest for new ideas.

2.4.1 Modes of Creative Thinking: Intense and Casual

Sanders & Thagard[4] identify two modes of creative thinking in computer science: *intense* and *casual*. The intense mode is actively working on the problem. The casual mode are those 'aha!' moments that happen in the shower[5] or when out for a walk[6] — when not actively engaged and focused on the problem, but letting one's mind wander while idling doing other things.

The casual mode of creativity sounds very appealing: go for a walk and the idea will just come to you without any explicit work. It's not quite like that. Sanders & Thagard identify some common elements in stories of successful casual mode creativity:

a. Immersion in problem domain
b. Absence of immediate pressure
c. Absence of distractions
d. Mental relaxation
e. Unstructured time
f. Solitude

Note the first step: immersion in the problem domain. That's the intense mode of creativity. It is unlikely that an idea will come to you when you're out for a walk if you have not already applied some elbow grease to the problem.

[3] Bill Moggridge. *Designing Interactions*. The MIT Press, Cambridge, Mass., 2007

[4] Daniel Sanders and Paul Thagard. Creativity in computer science. In James C. Kaufman and John Baer, editors, *Creativity Across Domains: Faces of the Muse*. Lawrence Erlbaum Associates, Publishers, 2002

[5] Alan Kay had his own shower in the basement of Xerox PARC to facilitate casual mode creativity. [39]

[6] Alan Turing found that many of his ideas came to him on long walks or runs. [39]

2.4.2 *Comparison is the basis of many new ideas*

Many authors, scientists, and researchers have remarked that comparison is one of the key ways that they arrive at new ideas. Comparisons are generally of similarity or of difference. Figure 2.2 identifies some kinds of comparisons. Comparisons such as symmetry and duality involve both similarity and difference, and so we categorize them separately.

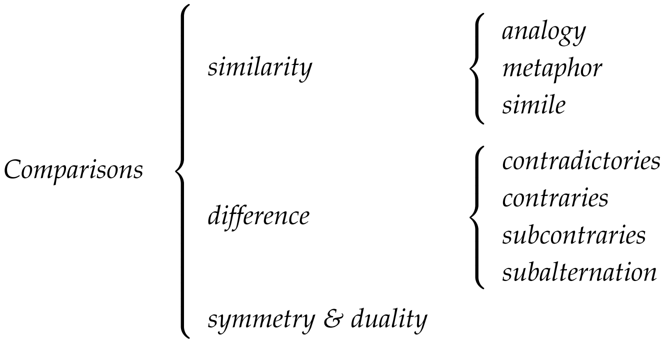


Figure 2.2: Some kinds of comparisons

The comparisons of difference in Figure 2.2 come from what is known as the *traditional square of opposition* in logic, depicted in Figure 2.3. For two contradictory propositions, one of them must be true and the other must be false. For two contrary propositions, at most one of them is true — but they could both be false. Similarly, for two subcontrary propositions, at most one of them may be false, but they may both be true.

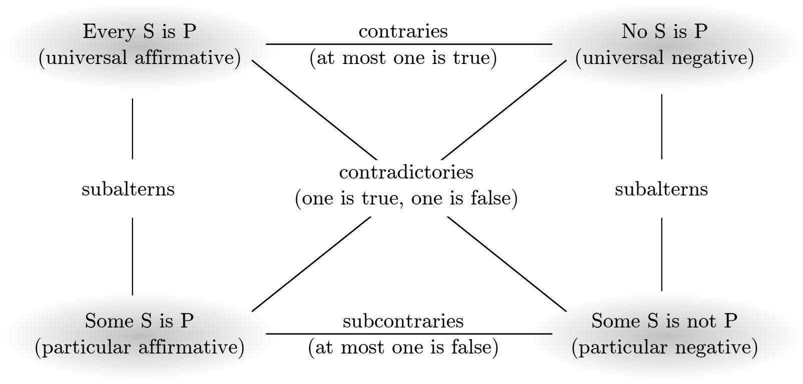


Figure 2.3: The traditional square of opposition. Originally described by Aristotle in *De Interpretatione*, and subsequently developed by many others. See, for example, the entry in the *Stanford Encyclopedia of Philosophy* by Parsons [37]: http://plato.stanford.edu/entries/square/

A STRATEGY FOR USING COMPARISONS to generate new ideas was proposed and tested by W.J.J. Gordon[7]: *make the familiar strange*, and *make the strange familiar*.

[7] William J.J. Gordon. *Synectics: The Development of Creative Capacity*. Harper & Row, New York, 1961

Local and Distant analogies. Sanders[8] & Thagard[9] studied interviews with famous software designers published in *ACM Queue* and found that comparisons of similarity were very often the basis of new ideas. They categorize these comparisons into *local* comparisons to other computing phenomena, and *distant* comparisons to phenomena in other areas. An example of a local analogy that they give is that the user interface of the original Macintosh computer was inspired by the user interface of the Xerox Alto. *Genetic algorithms* and *neural networks* are both examples of distant analogies to biological processes. They have a nice quotation from Ada Lovelace[10] in a letter she wrote to Charles Babbage[11] comparing Babbage's Analytical Engine[12] to the Jacquard Loom.[13]

$$\textit{Analogy} \begin{cases} \textit{local} & \text{Mac} \approx \text{Xerox Alto} \\ & \text{genetic algorithms} \\ \textit{distant} & \text{neural networks} \\ & \text{Analytical Engine} \approx \text{Jacquard Loom} \end{cases}$$

> We may say most aptly that the Analytical Engine weaves algebraic patterns just as the Jacquard loom weaves flowers and leaves.
>
> — Ada Lovelace[14]

Symmetry and Duality. *e.g.*, dynamic invariant detection[15] is the dual[16] of abstract interpretation[17]

Ideas may be combined to form new hybrids. Von Fange[18] describes using an *idea matrix* to systematically combine previously generated ideas into new hybrid ideas:

> Once a list of ideas has been gathered, we can quickly multiply its potential by listing the ideas both downward and across a sheet of paper (an Idea Matrix). By looking at each idea in combination with every other idea, we can produce still more ideas.

[8] Daniel Sanders and Paul Thagard. Creativity in computer science. In James C. Kaufman and John Baer, editors, *Creativity Across Domains: Faces of the Muse*. Lawrence Erlbaum Associates, Publishers, 2002

[9] Director of Waterloo's Cognitive Science Program

[10] The first software engineer.

[11] The first hardware engineer.

[12] The first computer.

[13] The Jacquard Loom was programmable in the sense that it used punched cards to describe the pattern to be woven.

[14] H. Goldstine. *The computer from Pascal to Von Neumann*. Princeton University Press, 1972

[15] daikon

[16] Michael D. Ernst. Static and dynamic analysis: synergy and duality. In Jonathan E. Cook and Michael D. Ernst, editors, *Proceedings of the Workshop on Dynamic Analysis (WODA)*, Portland, Oregon, 2003

[17] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the 4^{th} ACM Symposium on the Principles of Programming Languages (POPL)*, Las Angeles, CA, January 1977

[18] Eugene K. Von Fange. *Professional Creativity*. Prentice Hall, Englewood Cliffs, N.J., 1959

2.4.3 *Design Space Exploration*

There are a variety of techniques one can use to explore a design space and find new solutions.

MAKE A LOCAL ANALOGY TO THE NORMAL PROGRAMS. We have developed a very good understanding of how to write compilers, operating systems, and databases. Can the problem that you are trying to solve be reframed as one of these kinds of problems?

MAKE A DISTANT ANALOGY TO A BIOLOGICAL OR PHYSICAL PROCESS. Biology in particular has proven to be a useful source of inspiration in many areas of engineering.

TRY A DIFFERENT PATTERN OR STYLE. Parnas[19] presented two different designs for his toy example KWIC (Key Word In Context) program: flowchart and encapsulated. Garlan & Shaw[20] came up two more KWIC designs by trying different architectural styles. In an assignment we produced two designs for a simple calculator program, based on the Interpreter and Visitor design patterns, respectively. There are a number of catalogues of design patterns and architectural styles that one can turn to as a source of new ideas.[21,22,23,24]

AIM FOR ANOTHER PARETO POINT. The design(s) that you already have will make certain trade-offs in terms of computation time, space, effort to implement, modifiability, reusability, *etc.*. Identify the criteria that are important for the problem domain, see what trade-offs your current designs make, and then aim for a different trade-off. In an assignment we saw calculator designs that were modifiable but took some effort to implement; we also saw a design that sacrificed modifiability for a reduction in implementation effort.

CHANGE THE TECHNOLOGY. Different technology may require or facilitate different designs. The more different the technology, the more different the designs may be.

Changing programming paradigms can have a large impact on the possible designs. For example, when writing a web application in an imperative language (including imperative object-oriented languages such as Java), the control-flow is typically structured as a state machine with request/response. If writing a web application in a language that supports continuations, the control-flow can be structured more like a regular program, and the web application framework will use continuations to manage the state.[25] Try Prolog, Ruby, Scheme, or Haskell for a different way of thinking.

[19] David Lorge Parnas. On the Criteria to be Used in Decomposing Systems into Modules. *Communications of the ACM*, 15(12):1053–1058, December 1972

[20] David Garlan and Mary Shaw. An introduction to software architecture. Technical Report CMU-CS-94-166, Carnegie Mellon University, January 1994. URL http://www.scs.cmu.edu/afs/cs/project/able/ftp/intro_softarch/intro_softarch.pdf

[21] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995

[22] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. John Wiley & Sons, 1996

[23] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects*. Addison-Wesley, 2000

[24] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002

[25] Christian Queinnec. Inverting back the inversion of control or, continuations versus page-centric programming. Technical Report 7, LIP6, May 2001. URL http://www.lip6.fr/reports/lip6.2001.007.html

Relax a constraint or change an assumption. In an assignment we saw that we could simplify the design of a calculator program by changing the syntax of the input language it processes from infix operators to postfix operators (*i.e.*., reverse polish notation). In class we also discussed how the idea of operating system microkernels germinated in part from changing the assumption that the file system should be inside the kernel. The Seaside web application framework has an unconventional design:[26]

> Over the last few years, some best practices have come to be widely accepted in the web development world. Share as little state as possible. Use clean, carefully chosen, and meaningful URLs. Use templates to separate your model from your presentation.
>
> Seaside is a web application framework for Smalltalk that breaks all of these rules and then some. Think of it as an experiment in tradeoffs: if you reject the conventional wisdoms of web development, what benefits can you get in return? Quite a lot, it turns out, and this "experiment" has gained a large open source following, seen years of production use, and been heralded by some as the future of web applications.
>
> In this talk, you'll learn in-depth about Seaside's heretical design choices, and how it benefits from them. In particular, you'll see how closures and shared state let you ignore the details of URLs and query fields; how the right HTML generation API makes you less tied to your presentation layer, not more; how continuations free you from ever thinking about workflow as a state machine again; and how all of this combines to enable modularity and reuse like you've never seen before.

Morphological analysis.[27] Identify all of the design decisions and their possible alternatives, then systematically explore every possible combination.

What would Dijsktra do? There are many great software designers, many of whom have very distinctive styles. Ask yourself how they would approach your problem. Here are some names to familiarize yourself with: Edsgar Dijkstra, Simon Peyton-Jones, Butler Lampson, Tony Hoare, Rob Pike, Joshua Bloch, Michael Stonebreaker, Ted Codd, Linus Torvalds, Larry Wall, Donald Knuth, David Parnas, Fred Brooks, Michael Jackson.

[26] Avi Bryant. Web Heresies: The Seaside Framework. OSCON, 2006. URL http://conferences.oreillynet.com/cs/os2006/view/e_sess/8942

[27] Fritz Zwicky. *Discovery, Invention, Research — Through the Morphological Approach*. The Macmillian Company, Toronto, 1969

2.4.4 *Group Approaches to Creativity*

Showers may work for individuals to find ideas, but they are not known to be successful in this regard as a group technique.

Brainstorming is a commonly known technique originally developed by Alex Osborn[28]. The point of brainstorming is quantity, not quality. Setting a quota of the number of ideas to be generated can be helpful. Osborn's rules for successful brainstorming are:

> a. Judgement is ruled out. Criticism of ideas must be withheld until later.
>
> b. Free-wheeling is welcomed. The wilder the ideas, the better; it is easier to tame down than to think up.
>
> c. Quantity is wanted. The greater the number of ideas, the more the likelihood of winners.
>
> d. Combinations and improvement are sought. In addition to contributing ideas of their own, participants should suggest how ideas of others can be turned into better ideas; or how two or more ideas can be joined into still another idea.

Think / Pair / Share. Dym & Little[29] present three related techniques that I'll summarize with the pedagogical phrase *think / pair / share*.

a. *Think:* Each person in the group sketches k ideas, for some predetermined value of k usually in the range 1–3. A sketch may be a drawing or prose.

b. *Pair:* The sketches are passed around for written commentary. There is no talking in this step nor the previous step: all communication is written.

c. *Share:* The annotated sketches are posted on the wall and form the basis of a discussion.

Six Thinking Hats[30] Each hat represents a different mode of thinking. Everyone in the group is in the same mode at the same time, with the possible exception of the facillitator who may always wear the blue hat. Depending on what task the group is trying to accomplish they will put the hats on (metaphorically) in a different order.

🔵 *Process:* the group getting organized.

⚪ *Facts:* review the known facts of the issue being addressed.

🟢 *Creativity:* provocation; investigation; generation.

🟡 *Positive:* what is good? seeking harmony.

⚫ *Negative:* what are the limitations? seeking discord.

🔴 *Intuition:* straight from the gut.

[28] Alex F. Osborn. *Applied Imagination: Principles and Procedures of Creative Problem Solving*. Scribner & Sons, New York, 1953

[29] Clive L. Dym and Patrick Little. *Engineering Design: A Project Based Introduction*. John Wiley & Sons, 3 edition, 2008

[30] Edward de Bono. *Six Thinking Hats: An Essential Approach to Business Management*. Little, Brown, & Company, 1985

Task	*Hat Sequence*
Solving Problems	🟦⬜🟩🟥🟨⬛
Choosing	🟦⬜🟩🟨⬛🟥
Performance review	🟦🟥⬜🟨⬛🟩
Process improvement	🟦⬜🟨⬛🟩🟥

Solve an analogous problem.[31,32] Only one person in the group, the facilitator, knows what the real problem is. Before meeting the facilitator identifies the abstract problem domain, *e.g.*, storage or cutting, and an analogous problem in that domain. In the meeting the facilitator first introduces the abstract problem domain for discussion. After the group has warmed up to the abstract problem domain the facilitator introduces the analogous problem. The group works on the analogous problem using one of the techniques discussed above. As the discussion progresses the facilitator may reveal more and more details of the real problem, or the revelation of the real problem may wait until towards the end. Once the real problem is fully revealed then the group maps their solutions to the analogous problem to the real problem, perhaps generating new solutions in the process.

[31] This technique was developed by W.J.J. Gordon, who also wrote the *Synectics* book. Gordon's ideas are documented by Cros.[32]

[32] Pierre Cros. *Imagination, undeveloped resource : a critical study of techniques and programs for stimulating creative thinking in business*. Creative Training Associates, New York, 1955. URL http://hdl.handle.net/2027/uc1.l0050673813. Submitted in partial fulfillment of the requirements in Professor Georges F. Doriot's course in manufacturing at the Harvard Business School

2.5 Core Skill: Selecting from Alternatives

2.5.1 Conceptualizing the Design Space: Partial Orders and Pareto Fronts

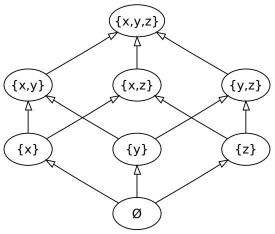
The term *design space* refers to a set of possible design alternatives. A design space may be characterized formally or informally. Some ways of characterizing a design space formally include the Pareto Front (which requires formalizing the metric or objective space) or a morphological analysis (which formalizes the design decisions).

In the words of Arthur C. Clarke: 'fast, cheap, or good: pick two.'

What is the relationship between designs in a design space? Are they totally ordered, like the integers? Usually not. It is more common that the designs in a design space are partially ordered, and that there is no distinguished supreme design in that ordering.

In a total order, any two elements can be compared and the result of that comparison is either $<$, $>$, or $=$. In a partial order a comparison may result in $<$, $>$, $=$, or ?: i.e., there may be pairs of unequal elements for which we cannot determine which is better than the other. In engineering terms, design is about making trade-offs.

Figure 2.4: Hasse diagram depicting a partial order [Wikipedia]



The following Hasse diagram shows the set of all subsets of the set $\{x, y, z\}$, ordered by inclusion (i.e., $\subseteq$, the subset relation). This is an example partial order [from the Wikipedia page on partial orders]. For example, we see that the set $\{x, y\}$ includes (is 'greater than') the set $\{x\}$. However, the sets $\{x, y\}$ and $\{x, z\}$ do not include each other and are not equal: they are 'incomparable'.

The partial order depicted in this Hasse diagram has a distinguished greatest element, namely the set $\{x, y, z\}$. It also has a distinguished least element, namely the empty set.

Design spaces rarely have distinguished top and bottom elements: there is usually no single best nor worst design. Imagine this Hasse diagram with the top and bottom removed: that's what most design spaces look like.

2.5.2 Making a Decision

Candidate designs from the design space should be subject to a rigourous engineering analysis. This analysis will likely eliminate certain proposals as infeasible or sub-optimal. It is likely that a number of proposals will remain, which represent different trade-offs. In this common situation, a decision must be made based on some external factor: the analytical criteria alone do not distinguish a single optimum (this is the very definition of the Pareto front).

Ultimately, a decision is an act of the will: a choice to make certain trade-offs, informed by the best available analysis.

2.6 Core Skill: Prototyping

There are different kinds of prototypes that can be built for different purposes. This section describes the ideas of *exploratory*, *evolutionary*, and *operational* prototypes described in the software engineering literature, as well as a novel approach for depicting a prototyping plan.

2.6.1 Experimental Prototypes

Plan to throw one away; you will, anyhow.
— Fred Brooks

An experimental prototype is intended to explore or demonstrate one aspect of a system, and to be discarded after the exploration or demonstration is complete. The purpose of an experimental prototype is to learn something. That knowledge is then used to build the real system.

Experimental prototypes are often built in a low cost way: in an inexpensive medium, intentionally excluding certain aspects of the system. If we were trying to build a house we might first make a number of sketches; then we might make cardboard mockups of a few of the designs. These sketches and mockups are experimental prototypes: our ideas about the system grow as we build and interact with them, but they are not the final product.

One might consider System R[33] as an experimental prototype for DB/2.[34] The developers of the seL4 microkernel[35] first built an experimental prototype in Haskell; the final version of the kernel was written in C. They compare their development time to that of other groups who have produced L4 microkernels and conclude that this experimental prototype actually saved them time overall.

Horizontal vs Vertical Experimental Prototypes. All of the experimental prototypes we've discussed so far have been *horizontal* prototypes: shallow prototypes of the entire system.

A *vertical* prototype is a detailed exploration of one facet or subsystem of the overall system. Since vertical prototypes have a specific focus they are often written in a programming language that is designed for that task. For example, an algorithm or transformation may be prototyped in a functional language; a user interface may be prototyped in Flash or Visual Basic.

Alternative programming paradigms can be useful for experimental prototypes. Functional programming languages are well suited to many kinds of algorithms and to symbolic transformations. Deductive logic languages such as Prolog and Datalog are used for many artificial intelligence and natural language processing tasks. There are languages specifically designed for constraint satisfaction problems such as scheduling. Alloy can also be used in this capacity.

Do five minutes of research to see if there is a programming language designed for your specific kind of problem.

[33] An early relational database engine from IBM Research.

[34] A commercial relational database engine from IBM.

[35] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. seL4: Formal verification of an OS kernel. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP)*, Big Sky, MT, USA, October 2009

Don't let your babies grow up to be cowboys. An experimental prototype is designed to learn something specific. An experimental prototype can reduce overall development time if this knowledge, but not the prototype itself, is used in building the final system.

An experimental prototype can increase overall development time if one succumbs to the temptation to treat it as an evolutionary prototype — *i.e.*, if one tries to make the experimental prototype part of the final system. Experimental prototypes are often made with tools that are not appropriate for the final system. Attempting to build the final system with unsuitable tools will usually end up costing, rather than saving, time.

One possible exception to this general rule is if the technology used for the experimental prototype can be integrated into the technology used for the final system as a whole. Proceed with caution.

Use experimental prototypes to learn about unfamiliar technology. If you are building a system in an unfamiliar technology, you can save yourself a lot of time by building vertical experimental prototypes to explore specific characteristics of that technology. Some students[36] learned this lesson the hard way:

> In order to code in iPhone, we must use Objective C. However, none of us was an expert in this language when we started the project. Consequently, there are some quirks that are specific in Objective C framework. One such example is exception handling. There are some methods that are run by the iOS framework on a separate thread (not main thread). Because our code runs on the main thread, when we call those methods, because those methods live in a different thread, we can't catch an exception on that thread. Unfortunately, we customized those methods to throw an exception upon connection failures. We didn't encounter any problem on the simulator because we had perfect connection on the wireless network all the time. But, when we actually tested it on the iPhone, we discovered that it crashed all the time upon connection failure.
>
> This caused us a lot of grief because in order to fix this problem, we either have to resort on changing the architecture of our program in a major way or resort on putting ugly hacks everywhere. Due to time limitation, we decided to use hacks.
>
> Had we actually tested our program on the device on the early prototyping stage, we would have caught this problem much sooner, and we would actually design the program to handle this quirkiness of the framework.

[36] The iLesion group from se2011. Their project was to make an iPhone app for radiologists diagnosing brain lesions (tumours).

2.6.2 Evolutionary Prototypes

An evolutionary prototype is one that eventually becomes the real system. An evolutionary prototype may start out like a horizontal experimental prototype as a shallow and incomplete version of the final system. However the evolutionary prototype is never thrown away: it evolves into the final system.

Evolutionary prototypes have been widely advocated in software engineering. Brooks[37] talks about 'organic growth'. Gabriel[38] (and others) have argued that evolutionary prototypes reduce time to market and that establishing a user-base and a growth plan is crucial for a project to stay alive. More recently, this evolutionary approach has been advocated by the agile programming community.

Evolutionary prototypes can be particularly good for exploring user requirements and preferences.

It is unwise to start an evolutionary prototype in an unfamiliar medium: vertical experimental prototypes should be used to learn the medium and tools before the evolutionary prototype is begun.

Evolutionary prototypes are not well-suited to exploring key algorithmic or computational concerns: experimental prototypes are better for this. Evolutionary prototypes need to handle all of the device and user interactions, and this tends to distract one from experimenting freely with computational dimensions of the problem.

This is more the Pokémon concept of evoluation than the biological concept of evolution. In biological evolution, children differ from parents, and the fittest children survive to reproduce. In Pokémon 'evolution', an individual grows into a new form, like a tadpole becoming a frog, or a Pikachu becoming a Raichu.

In Pokémon terms we might say that TeX is *legendary* because it does not evolve: it's version number becomes an ever closer approximation of π as it is perfected. In software, as in Pokémon, legendaries are rare: almost all software evolves or dies.

[37] Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, 1975

[38] Richard P. Gabriel. Lisp: Good news, bad news, how to win big. *AI Expert*, pages 31–39, June 1991. URL http://www.dreamsongs.com/NewFiles/LispGoodNewsBadNews.pdf. Presented as the keynote address at the European Conference on the Practical Applications of Lisp, Cambridge University, March 1990. Commonly known as 'Worse is Better'

2.6.3 Operational Prototyping

Add experimental prototypes to an evolutionary base, then throw the experimental bits away and re-implement them properly within the evolutionary base.[39] The modern manifestation of this idea is using branches in a version control system to explore new features.

[39] Alan M. Davis. Operational prototyping: A new development approach. *IEEE Software*, 9(5), September 1992

2.6.4 When to build which kind of prototype

Characteristics	*Experimental Prototyping*	*Evolutionary Prototyping*	*Regular Development*
Development approach	Quick and dirty; sloppy	Rigorous; not sloppy	Rigorous; not sloppy
What is built	Poorly understood parts	Well-understood parts first	Entire system
Design drivers	Development time	Ability to modify easily	Depends of project
Goal	Verify poorly understood requirements and then throw away	Uncover unknown requirements and then evolve	Satisfy all requirements

Figure 2.5: When to build which kind of prototype: experimental, evolutionary, operational

Alan M. Davis. Operational prototyping: A new development approach. *IEEE Software*, 9(5), September 1992

2.6.5 *Picturing a Prototype Plan*

There are different kinds of prototypes and all of them may be used while designing. When multiple prototypes of different kinds all lead towards a final design it can be useful to visualize how the relate to each other. First we introduce some iconography for the different kinds of prototypes discussed above:

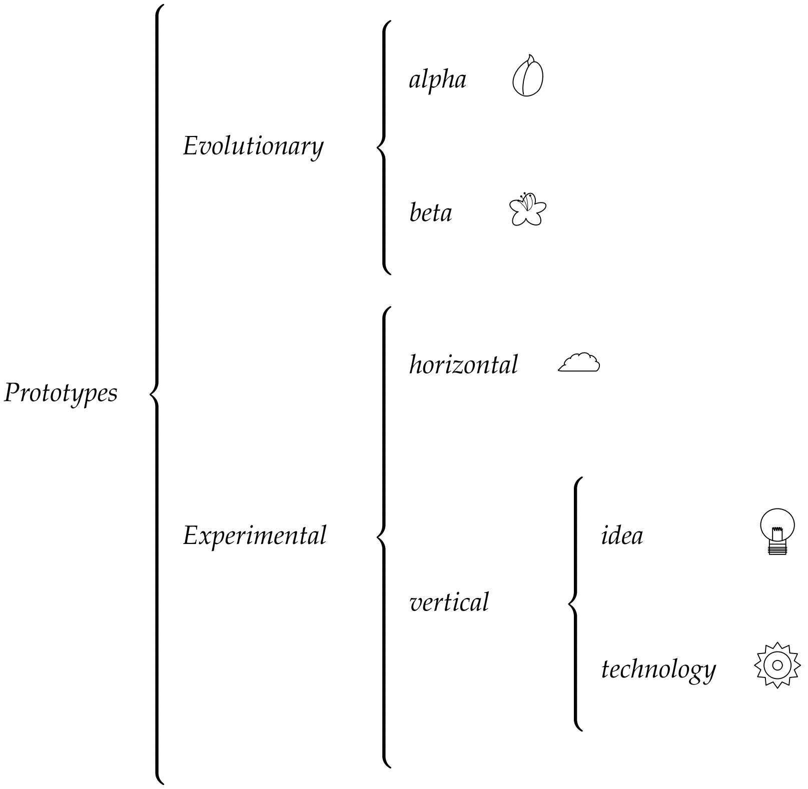
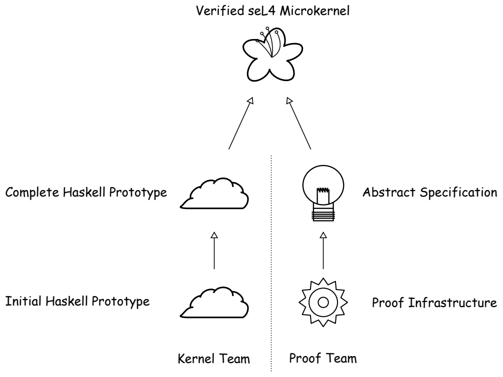


Figure 2.6: Icons for different kinds of prototypes. Icons drawn by Albert T. Yuen (se2012). Reuse by se students permitted.

With these icons we can tell the story of the seL4 verified microkernel[40] (or of your fourth year design project).


Verified seL4 Microkernel
Complete Haskell Prototype
Abstract Specification
Initial Haskell Prototype
Proof Infrastructure
Kernel Team
Proof Team

*

Figure 2.7: The story of the seL4 verified microkernel

[40] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. seL4: Formal verification of an OS kernel. In *Proceedings of the* 22*nd ACM Symposium on Operating Systems Principles (SOSP)*, Big Sky, MT, USA, October 2009

2.7 *The Design Process*

This section needs improvement

$$\text{Processes}\begin{cases}\textit{waterfall}\\ \textit{spiral}\\ \textit{funnel}\end{cases}$$

a. Ideation
b. Analysis
c. Selection
d. Elaboration
e. *Iteration*

Software design is often weak on ideation, analysis, and selection: all of the effort is spent in elaboration/refinement. I would like you to be able to perform all of these steps well.

We will not study ideation in detail. There are techniques for doing it well.

We may study analysis. What is commonly done under the rubric of software 'design' and 'architecture' does not do much analysis. This is part of why, for example, the agile approach taken in your fourth-year design project eschews design as a bunch of hot air: YAGNI.

Question: do we study analysis, or cover more modes of normal composition? Maybe we'll have a poll. There are some people who think that you don't have the skills for analysis.

We'll talk a little about the mathematics of complex decision making at some point, possibly later today.

2.8 *Definitions of Design*

The Software Engineering Institute at Carnegie Mellon University has catalogued over 20 different definitions for software architecture. I categorize these definitions into three groups: *structure-oriented*, *decision-oriented*, and *communication-oriented*.

$$\textit{Design Defns}\left\{\begin{array}{l}\textit{structure}\\ \textit{decision}\\ \textit{communication}\end{array}\right.$$

Structure-oriented: Some important structure-oriented definitions of software architecture include:

- *Garlan and Perry [20] 1995:* The structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time.
- *ANSI/IEEE 1471* [41] *2000*: The fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.
- *Bass et al. [1]:* The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

I think that the industrial designer and architect Charles Eames[42] made a clearer and more concise statement of this kind of definition several decades earlier:

- *Eames:* A plan for arranging elements in such a way as to best accomplish a particular purpose.

Structure-oriented definitions are the most common and have considerable influence over our design representations (which we will discuss in more detail below).

Decision-oriented: Decision-oriented definitions are gaining prominence in systems engineering (*e.g.*, Koo & Simmons[43]) and are occasionally found in software. For example, the Taylor *et alia* [44] software architecture textbook gives the following pleasingly concise definition:

- *Taylor et al. [42]:* A software system's *architecture* is the set of principal design decisions made about the system.

http://www.sei.cmu.edu/architecture/start/definitions.cfm

[41] IEEE. Recommended practice for architecture description of software-intensive systems. Technical Report ANSI/IEEE 1471-2000, 2000. URL http://www.iso-architecture.org/ieee-1471/

[42] Bill Moggridge. *Designing Interactions*. The MIT Press, Cambridge, Mass., 2007 p.648

[43] H.-Y. Benjamin Koo. *A Meta-language for Systems Architecting*. PhD thesis, Engineering Systems Design, Massachusetts Institute of Technology, 2005; and Willard Simmons. *A Framework for Decision Support in Systems Architecting*. PhD thesis, Aeronautics & Astronautics, Massachusetts Institute of Technology, 2008

[44] Richard N. Taylor, Nenad Medvidović, and Eric M. Dashofy. *Software Architecture: Foundations, Theory and Practice*. John Wiley & Sons, 2009

Communication-oriented: (Conway's Law)

- *Conway*[45] *1968:* organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations

Kitchen Sink:

- *Kruchten*[46] *1999:* An architecture is the set of significant decisions about the organization of a software system, the selection of the structural elements and their interfaces by which the system is composed, together with their behavior as specified in the collaborations among those elements, the composition of these structural and behavioral elements into progressively larger subsystems, and the architectural style that guides this organization—these elements and their interfaces, their collaborations, and their composition

Argument for Decision-oriented: Why is the decision-centric definition of design better for software? Structure-oriented definitions are rooted in an analogy to the physical world, whereas decision-oriented definitions are inherently conceptual and hence a better match for software design. For example, important decisions in the design of a software system include policies about naming, mutation, storage, computation, *etc.*. These decisions do not exist as components in the system but, rather, govern how the components behave and interact.

SE464 Student Definitions:

- The act of building something on paper as opposed to physically to meet the constraints of:
  - Customer
  - Environment
  - Mathematical reality
  - Existing components
  - Existing mathematics[47]
  - Existing science[48]
- Design is the output of the act of designing (See side note 3)

[45] Melvin E. Conway. How do committees invent? *Datamation*, 14(5):28–31, April 1968. URL http://www.melconway.com/research/committees.html

[46] Philippe Kruchten. *The Rational Unified Process*. Addison-Wesley, 1999

[47] Oliver Heaviside was an electrical engineer who invented new techniques for solving differential equations. He is also responsible for the formulation of Maxwell's equations that we use today.

[48] The steam engine famously predates the science of thermodynamics.

2.9 *Normal and Radical Design*

Don't reinvent the wheel

Engineers have been distinguishing between *normal* and *radical* design for some time:

> In designing a new machine, an engineer employs familiar components, often in rearranged configurations and occasionally in radically modified ones.[49]

Ferguson continues:

> An auto engine is an everyday machine whose existence 500 years ago was impossible to imagine. Yet except for the electrical components—the ignition coil and the spark plugs—nearly all of its elements were known when Leonardo was alive (1452–1519). The engine is composed of cylinders and pistons, a crankshaft, conical valves, cams, gears, bearings, chains, belts, and other mechanical components. The repertoire of mechanical elements was astonishingly close to completion when Leonardo was filling his notebooks with drawings of them. Some components, such as cylinders and pistons, date as far back as the first century A.D.[50]

Normal design involves incremental improvement in a product class that is well understood, with known problems and solutions, and customary user interactions. By contrast:[51]

> In radical design, how the device should be arranged or even how it works is largely unknown. The designer has never seen such a device before and has nor presumption of success. The problem is to design something that will function well enough to warrant further development.

How might we consider the space in between normal and radical design? Should we group the Toyota Prius with the Apollo space program (entirely radical) or the 2010 Honda Civic (completely normal)? Or is it somewhere in between? Figure 2.8 explores the space between normal and radical in terms of components and their composition. In the middle ground, a design might arrange normal components in a radical way, or it might involve radical new components arranged in a normal way.

In software engineering our most common kinds of components are data structures, algorithms, and protocols. We record normal arrangements of these components in *reference architectures*, *architectural styles*,[52] and *design patterns*.[53] A reference architecture describes a normal way of building a particular kind of thing, such as a web server, a compiler, or an operating system. We might say that *microkernel* and *monolithic kernel* are two different reference architectures for operating systems. Architectural styles and design patterns describe normal ways of arranging components, independent of the particular problem domain.

[49] Eugene S. Ferguson. *Engineering and the Mind's Eye*. The MIT Press, Cambridge, Mass., 1992

[50] Eugene S. Ferguson. *Engineering and the Mind's Eye*. The MIT Press, Cambridge, Mass., 1992

[51] Walter G. Vincenti. *What Engineers Know and How They Know It: Analytical Studies from Aeronautical History*. The Johns Hopkins University Press, 1993

[52] David Garlan and Mary Shaw. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, Inc., 1996

[53] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995

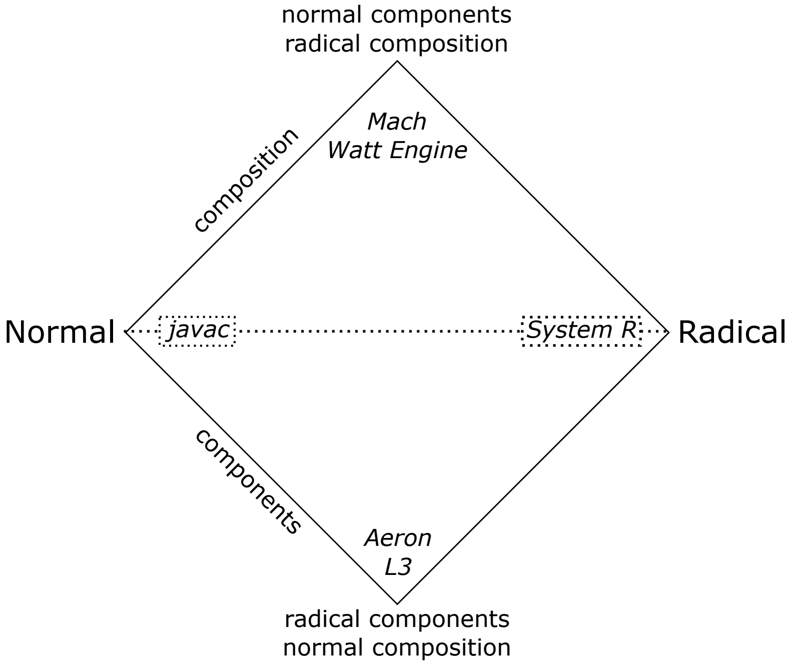


Figure 2.8: The space between Normal and Radical design in terms of components and composition.

For example, we might consider the Watt steam engine a radical composition of normal components: James Watt had the idea to separate the condenser from the cylinder while repairing a Newcommen steam engine (a design that was already fifty years old when Watt had his idea). Similarly, we might consider early microkernel designs (such as Mach) to be a radical composition of normal components: an operating system still has a file system, memory management, scheduling, *etc.*; it's just that a microkernel arranges these things differently than a monolithic kernel.

The Aeron chair substitutes mesh over a frame for fabric and upholstery: a radical component in what is otherwise a normal chair. L3 showed that by re-engineering the components of a microkernel performance could be improved considerably.

System R was an early relational database engine (from IBM). javac is the standard Java compiler (from Sun).

Don't reinvent the wheel. Most engineering practice involves normal design, and there are many practical benefits to working within the bounds of normality, including: reduced risk, reduced cost, easier maintenance, easier communication, and faster development time. Innovation generally increases risk, and so the benefits of the proposed innovation should outweigh its costs and risks.

Professional designers find normal solutions to radical problems: they find a way to minimize the innovation required. Some good strategies for doing this include:

- Re-imagine the new radical problem as a known normal problem through an analogy.
- Arrange normal components in a radical way.
- Use radical components in a normal arrangement.

Amateurs, by contrast, find radical solutions to normal problems: they reinvent the wheel. Amateurs create unnecessary difficulty not only for themselves, but also for those who must pay for, use, and maintain the systems they create.

Terminology. The Taylor[54] software architecture textbook uses the term *unprecedented* for what we are calling *radical design*. Our terminology originates with Edward Constant[55], and comes to us by way of Vincenti[56] and Jackson[57].

[54] Richard N. Taylor, Nenad Medvidović, and Eric M. Dashofy. *Software Architecture: Foundations, Theory and Practice*. John Wiley & Sons, 2009

[55] Edward W. Constant. *The Origins of the Turbojet Revolution*. The Johns Hopkins University Press, 1980

[56] Walter G. Vincenti. *What Engineers Know and How They Know It: Analytical Studies from Aeronautical History*. The Johns Hopkins University Press, 1993

[57] Michael A. Jackson. The Name and Nature of Software Engineering. In Egon Börger and Antonio Cisternino, editors, *Advances in Software Engineering: Revised Lectures of Lipari Summer School 2007*, volume 5316 of *Lecture Notes in Computer Science*, pages 1–38. Springer-Verlag, 2008

2.10 Laws of Software Design

You should take these seriously, but not too seriously. They are not all applicable in every situation. Consider them to be rules of thumb. Part of developing your professional judgment is knowing when they apply.

AMDAHL'S LAW (1967) Used to calculate the maximum possible overall speedup of a program if part of it is paralellized.

https://en.wikipedia.org/wiki/Amdahl's_law

BROOKS'S LAW[58] (1975) Adding manpower to a late software project makes it later.

BROOKS ON PROTOTYPING[59] (1975) Plan to throw one away; you will anyhow.

CAP THEOREM[60] It is impossible for a distributed system to provide all three of *consistency*, *availability*, and *partition tolerance*.

CONWAY'S LAW[61] (1968) Any piece of software reflects the organizational structure that produced it.

GREENSPUN'S TENTH RULE (1993) Any sufficiently complicated C or Fortran program contains an ad hoc, informally-specified, bug-ridden, slow implementation of half of Common Lisp.

https://en.wikipedia.org/wiki/Greenspun's_tenth_rule

HOARE'S RAZOR[62] (1980) There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.

SAINT-EXUPÉRY'S RAZOR (1939) Perfection is finally attained not when there is no longer anything to add, but when there is no longer anything to take away.

Wind, Sand, and Stars (Terre des hommes). Translated to English by Lewis Galantière. A memoir of adventures in the early days of aviation. The only non-software author in this section. This quote is widely cited by all kinds of designers, including software designers (*e.g.*, Lampson's *Hints for Systems Design*).

EINSTEIN'S RAZOR Everything should be made as simple as possible, but no simpler.

METCALFE'S LAW (1980) The value of a network is proportional to the square of the number of connected users/devices.

https://en.wikipedia.org/wiki/Metcalfe's_law

MOORE'S LAW (1965) Transistor density doubles every two years.

https://en.wikipedia.org/wiki/Moore's_law

[58] Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, 1975

[59] Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, 1975

[60] Wikipedia. CAP Theorem (Brewer's Theorem). URL https://en.wikipedia.org/wiki/CAP_theorem. Retrieved 2016-11-15

[61] Melvin E. Conway. How do committees invent? *Datamation*, 14 (5):28–31, April 1968. URL http://www.melconway.com/research/committees.html

[62] C. A. R. Hoare. The emperor's old clothes. *Communications of the ACM*, 24 (2):75–83, February 1981. Acceptance speech for 1980 Turing Award

Lehman's Law's of Software Evolution (1972–1996) Lehman first divides programs into three kinds:

https://en.wikipedia.org/wiki/Lehman's_laws_of_software_evolution

- *S-type:* Has an exact specification.
- *P-type:* In an externally defined domain, such as playing chess.
- *E-type:* Used by people for some socially constructed task. Must adapt to changes in the social/business environment.

The laws only apply to E-type programs.

1. *Continuing Change:* an E-type system must be continually adapted or it becomes progressively less satisfactory.
2. *Increasing Complexity:* as an E-type system evolves, its complexity increases unless work is done to maintain or reduce it.
3. *Self Regulation:* E-type system evolution processes are self-regulating with the distribution of product and process measures close to normal.
4. *Conservation of Organisational Stability (invariant work rate):* the average effective global activity rate in an evolving E-type system is invariant over the product's lifetime.
5. *Conservation of Familiarity:* as an E-type system evolves, all associated with it, developers, sales personnel and users, for example, must maintain mastery of its content and behaviour to achieve satisfactory evolution. Excessive growth diminishes that mastery. Hence the average incremental growth remains invariant as the system evolves.
6. *Continuing Growth:* the functional content of an E-type system must be continually increased to maintain user satisfaction over its lifetime.
7. *Declining Quality:* the quality of an E-type system will appear to be declining unless it is rigorously maintained and adapted to operational environment changes.
8. *Feedback System:* E-type evolution processes constitute multi-level, multi-loop, multi-agent feedback systems and must be treated as such to achieve significant improvement over any reasonable base.

Parnas's Criteria[63] (1972) A program should be decomposed into modules in a way that encapsulates (hides) things that are likely to change in the future. When those things do change, adapting the program is a relatively simple and localized edit. The program's module structure should not be derived from a flowchart describing the computation.

This idea is the basis of design patterns,[64] which mostly describe how to organize object-oriented software to manage certain kinds of anticipated future change.

[63] David Lorge Parnas. On the Criteria to be Used in Decomposing Systems into Modules. *Communications of the ACM*, 15(12):1053–1058, December 1972
https://en.wikipedia.org/wiki/David_Parnas

[64] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995

Parnas's Path As a rule, software systems do not work well until they have been used, and have failed repeatedly, in real applications.

Lampson's Hints on System Design[65] (1983) Recommended reading for anyone doing systems programming. Lampson won the 1992 Turing Award for his work in computer systems design. His long career spans Xerox PARC, MIT, and Microsoft Research.

Hoare's Hints on Language Design[66] (1973) A classic set of guidelines. Lampson recommends this as complementary reading to his hints. Lampson [67] points out that API design *is* language design — just without the concrete syntax.

Perlis's Epigrams (1982) Selected excerpts:

5. If a program manipulates a large amount of data, it does so in a small number of ways.
6. Symmetry is a complexity-reducing concept; seek it everywhere.
7. It is easier to write an incorrect program than understand a correct one.
9. It is better to have 100 functions operate on one data structure than 10 functions on 10 data structures.
15. Everything should be built top-down, except the first time.
16. Every program has (at least) two purposes: the one for which it was written, and another for which it wasn't.
19. A language that doesn't affect the way you think about programming, is not worth knowing.
20. Wherever there is modularity there is the potential for misunderstanding: Hiding information implies a need to check communication.
21. Optimization hinders evolution.
31. Simplicity does not precede complexity, but follows it.
54. Beware of the Turing tar-pit in which everything is possible but nothing of interest is easy.
58. Fools ignore complexity. Pragmatists suffer it. Some can avoid it. Geniuses remove it.
102. One can't proceed from the informal to the formal by formal means.

Epigram: a pithy saying or remark expressing an idea in a clever and amusing way. Perlis's article *Epigrams on Programming* was published in ACM SIGPLAN Notices 17(9) 1982. https://en.wikipedia.org/wiki/Epigrams_on_Programming

Alan Perlis: (1922–1990) Recipient of the first Turing Award (1966): *for his influence in the area of advanced programming techniques and compiler construction* (this official citation refers to his work on the ALGOL language). Prof at Yale. https://en.wikipedia.org/wiki/Alan_Perlis

https://en.wikipedia.org/wiki/Turing_tarpit

[65] Butler W. Lampson. Hints for computer system design. *ACM Operating Systems Review*, 15(5):33–48, October 1983. URL http://research.microsoft.com/en-us/um/people/blampson/33-hints/webpage.html. The online version is slightly revised

[66] C.A.R. Hoare. Hints on programming language design. Technical Report STAN-CS-73-403, Stanford, December 1973. URL http://web.eecs.umich.edu/~bchandra/courses/papers/Hoare_Hints.pdf. Keynote talk at POPL'73

[67] Butler W. Lampson. Hints for computer system design. *ACM Operating Systems Review*, 15(5):33–48, October 1983. URL http://research.microsoft.com/en-us/um/people/blampson/33-hints/webpage.html. The online version is slightly revised

Overall Outline

This chapter describes the overall timeline for the capstone project, as well as other aspects that are common across the three courses. Each course has its own subsequent chapter that describes the activities of that course in more detail, as well as giving the grading breakdown for the course.

Italicized time points indicate when project grading occurs. Exercises might be graded at other times.

Criteria given are for an A grade. C is meeting criteria of previous milestone. D is for being below criteria of previous milestone. B is between A and C.

3.1 *Timeline*

Course	*Activity*		*Accomplishment*
SE390	Internal Mini-Project	W1	Make a group, pick a project
			· brainstorming: an idea for each Result Rubric
			· pick a project and start doing it
	Symposium Videos	W2	Grade videos using Status Sheets & Rubrics
	Strategy & Scrum	W3	Corporate strategy exercise §4.10
			Discuss progress, challenges, goals (each team)
			List three skills on class roster (each student)
			· classmates might ask you for help on these topics
	Presentations	W4	Submit three slides to common slide deck
			· speak to Status Sheet
			· show functionality
	External Mini-Project	W5	Pick an external project, possibly change groups
			· choose a free/open-source (FOSS) system
			· *or* choose a third-party project proposal
	Understanding Risk	W6	Discuss past projects that changed and why
			Report whether you can build your FOSS project
	Symposium Videos	W7	Grade videos using Status Sheets & Rubrics
	Presentations	W8	Submit three slides to common slide deck
			· explanation · progress · challenges
	Project Ideation	W9	Generate at least one idea in each category:
			· ATE-related · CRUD · FOSS · research
			· new product · consultant · other
			· third-party proposal (your favourite)
			Work individually, then in pairs, then in groups
	Project Selection	W10	Announce groups and candidate ideas
			· submit three ideas to common slide deck (3 + 1 summary slide)
			· class vote (non-binding) on which project to do
	Symposium Videos	W11	Propose a video that is relevant to your project
			Grade videos using Status Sheets & Rubrics
	Scrum	W12	Discuss progress, challenges, goals
			Give feedback to other teams
	Final Demo	*exam*	Exploratory prototype(s)
			· rough spec: major categories identified
			· paper prototypes
			· technology experiments + initial selection
			· ATE plan
			· roughly 1,000 LOC (see notes)

Course	*Activity*		*Accomplishment*
Work Term	January		Gathering Data (if relevant). Students are not expected to do significant software development over the work term.
SE490	*Set up*	W1 ♥	Infrastructure · version control · automated test harness · build system · issue tracker *Old stories told in new ways* §7.1
	Risk Identification	W2	Plan for exploratory prototypes
	Peer Design Exploration	W3	Suggest alternatives to others
	Yelp! Earlybird Prizes	W4	Present to sponsor for prizes (optional)
	Midterm Demo	W5-7 ♥	MVP · spec fully identified, partially satisfied · basic functionality present · plan reified in issue-tracking system · roughly 2,000 LOC (see notes)
	Peer Usability Review	W8-9	Pilot user studies
	Peer Code Review	W10-11	Apply SE464 knowledge
	Final Demo	*exam* ♥	V1 · core functionality present · spec mostly satisfied · ATE knowledge applied · design & architecture articulated · responses to feedback · clear vision of how results will be gathered · roughly 5,000 LOC (see notes)
Work Term	September		Gathering Results · *New Product:* marketing · *Research:* experiments · *Consultant:* preliminary user experience · FOSS: patch submitted Students are not expected to do significant software development over the work term. It is a time for the outside world to use, interact with, or evaluate the software.

♥ Indicates an anonymous survey on how you are feeling about your project and team. We will also fill out (non-anonymous) project status sheets at these times.

Course	*Activity*		*Accomplishment*
SE491	Course Intro	W1	Course objective: develop V2; gather results
			Initial meetings
	Symposium Videos	W2	Grade videos using Status Sheets & Rubrics
	Referee Selection	W2	Nominate (to the Instructor) potential referees
	Legal report I	W3	First draft of legal/ethical/social report
	Legal report II	W4	Peer feedback on legal/ethical/social report
	Peer Review Exercise	W5	Peer review/usability exercise with students from another department. Might be a user study.
	$n+1$ Cohort Feedback	W6	Provide feedback to the $n+1$ cohort on their abstracts
	Reading Week	W7	*Reading Week*
	Effective Presentations I	W8	Learn how to make an effective presentation
	Practice Symposium	W9	Two days of practice talks.
			Each team gets 20 min talk + 25 min discussion
	Effective Presentations II	W10	Learn how to make an effective presentation
	Symposium Day!	W11	Presentation + Poster + Demo + *Results!*
			· full functionality
			· roughly 6,000 LOC (see notes)

SE390 *Outline*

The calendar description of SE390 is:

> Students undertake a substantial customer-driven group project as part of the SE 390/490/491 design-project sequence covering all major phases of the software-engineering lifecycle. Lectures describe expectations and project-planning fundamentals. Students form groups, decide on a project concept, complete a project-approval process, develop high-level requirements for the project, perform a risk assessment, develop a test plan, and complete a first-iteration prototype. Social, legal, and economic factors are considered.

This course has a few differences from the standard course format: all work is done in groups (although some is marked individually); there is no midterm test; there is no textbook.

4.1 Objectives

The goal of SE390 is to select and begin your Capstone Design Project. This is challenging. You have, until now, received relatively little training to prepare you for this kind of open-ended opportunity. In most of your previous courses, you have worked on small, focused problems that were selected for you. If there was a course project, it was of limited duration and in a prescribed domain (that of the course). Now you are faced with the daunting challenge of selecting a project in any domain that is worthy of more sustained effort, that will create positive change in the world (§1), and that you will be proud of (§1). How to do it?

In SE390, we will practice *making strategic decisions* (§1) and *thinking big thoughts* (§1). These are two of the key skills you need to develop for selecting your project and becoming a future intellectual leader. We will develop these skills through in-class activities and through two mini-projects.

4.2 *How to Choose a Project*

PASSION. Projects that students are excited about tend to succeed. Capstone is a unique opportunity for your to pursue your interests.

LEARNING. Pick a project where you will learn something. Acquire some new skills or ideas. Perhaps choose a project where you will be able to explore some of your ATE knowledge in greater depth.

CONTRIBUTION. Engineering is about applying technology in the service of society, including through society's commercial interests. Presumably you chose to be in engineering because you find satisfaction in making useful things. The Symposium Day Results Rubrics [§6.5] are some of the common ways that students have chosen to make contributions in the past. Your project might make a contribution in a new way that requires the creation of new rubrics.

Symposium Day Results Rubrics [§6.5]

Positioning can be important. For example, SE2016 Team Sleekbyte made a linter for Apple's Swift language at a time when Swift was new and none of the existing linters did proper parsing. Similarly, SE2016 Team Kaze was one of the first to use Riot Games API for League of Legends. Both of these teams added value to a previously defined user community.

TEAM. You may pick your own team for the final project, with no restrictions other than a minimum size of three (mandated by CEAB), and a maximum size of five (or maybe six; by historical convention in SE). One approach is to choose people you have worked with in the past and then select an idea that you all (more or less) agree on. Another approach is to seek out people who are excited about the same idea as you are.

You may continue either of the mini-projects on as the full project. You may form a new team or keep one of the mini-project teams for the full project.

IDEATION. Consider many ideas. The more ideas you consider, the better the chances of finding a good one that you are satisfied with.

Typically the most difficult thing for engineering students in generating ideas is to temporarily turn off the critical analysis part of their brains. Generation and evaluation of ideas are completely separate mental activities that should be done on separate days. In engineering education we often focus on analysis and evaluation rather than generation. Other technical disciplines, such as industrial design and architecture, spend more time training their students to generate ideas. This part of capstone is one of your opportunities to develop your lateral thinking skills, to think of ideas without consequences, to laugh while doing your school work. Embrace that experience. Something good will come of it.

4.3 Strategic Project Positioning

One of the key factors in choosing your project [§4.2] and understanding project risk [§4.4] is *strategic positioning*. Making good strategic decisions is one of the intended learning outcomes [§1].

On Symposium Day your grade will be determined in part by your results [§6.5]. At a high-level, your software must have done something for someone. There are currently four different categories of results to aim for [§6.5]: New Product, Custom Software, Research, and Free/Open Source Software.

You may propose a new category if the existing ones do not fit your project. One potential new category that has been discussed many times is *framework*.

That is, contributing to a large, pre-existing FOSS system. Not merely releasing your code as FOSS.

Strategic positioning is fairly easy in the Custom Software, Research, and FOSS categories: typically you have an external collaborator who has already identified a good opportunity, and will provide strong support for data acquisition and marketing (if necessary). But these are the least popular categories. Most students want to make new products — that category requires more strategic thinking. Some common dimensions to consider include:

No team has really succeeded with web scraping. It sucks a huge amount of time and doesn't produce good results.

a. *Data acquisition:* Does the project need data? How are you going to get it? Both UW Flow SE2014 and Kaze SE2016 were enabled by the release of new third party data APIS: UW Open Data and Riot Games, respectively.

b. *Marketing:* How are you going to acquire users? UW Flow SE2014 collaborated with students outside SE and student activities, such as orientation. Kaze SE2016 used Reddit to reach League of Legends players. Sleekbyte SE2016 reached out to prominent bloggers and thought leaders to reach Swift programmers. Parallax SE2016 similarly reached out to thought leaders in the WebGL community.

c. *New Technology:* Is there some new technology that gives your project good leverage and timing? For example: Sleekbyte SE2016 made a linter for Apple's new programming language, Swift. People have been making linters for forty years. Nothing new there. Every mature language needs one. Swift didn't have a good one yet. Parallax SE2016 made a debugger for WebGL, which was relatively new and lacking debugger support.

d. *Novelty* in the real world is usually about shades of gray. Nevertheless, working in a very crowded space, especially one with entrenched interests, makes it harder to find a niche.

e. *Background knowledge* in an area can help you identify a good opportunity. For example, Mixbox SE2014 won a $10k Esch Award for a music app that presented a new trade-off between features and usability as compared to existing products.

f. *Awesomeness:* Some teams are determined to be better. For example, a number of teams had done the course critique project before UW Flow SE2014, but they were determined to be better. Many of the teams who do game-related projects put in astonishing amounts of effort to produce great results: *e.g.*, unLit SE2016, HiveMind SE2013, Kinectitude SE2013.

The instructor's opinion of whether your project is likely to produce results, based on your project blurb in SE390, can be wrong. That's why it isn't part of the grading, and why we don't tell you that you cannot do a particular project (assuming it is legal and ethical).

There are two main reasons the instructor's opinion might be wrong: your blurb does not adequately communicate your strategic advantages, and they are not familiar with the domain.

4.3.1 *Some Strategic Assessments of Student Proposals*

Restaurant Feedback System. Small family restaurants often depend on building good relationships with their local communities in order to stay afloat. Building this rapport is necessary for success but can often be challenging because traditionally it requires face-to-face communication between customers and restaurant owners. Our proposed Restaurant Feedback System aims to facilitate these communications by making it easy for customers to anonymously provide feedback to restaurant owners.

What makes this system different from its existing counterparts such as Yelp is that all feedback is private and meant only to help restaurant owners improve their services. Local restaurants are likely to want to use this system because it can help facilitate more honest feedback than public review systems, where customers are likely to exaggerate their comments and be excessively biased for the sake of helping or harming the public image of the restaurant.

Since this system is set up to help restaurant owners succeed, we will set up an incentive system in order to get restaurant customers on board so that they have a reason to make reviews outside of helping local businesses succeed. Owners will be able to create coupons that they can offer to people who make reviews for their restaurant anonymously.

Assessment: Your proposal for a Restaurant Feedback System makes a good effort to lay out a strategy to get results: why it's different than Yelp/ChowHound/etc., and how you would entice restaurant owners and patrons to use the system. That's good.

I wonder if you actually know anyone who owns a restaurant? Do they think this a good idea? I suspect that if they see any value in it at all, they will see it as a way to get repeat business from existing customers, and not really care about the feedback at all. Asking the customers for the feedback is just a ruse to get them to think that their opinion matters: the point is to get them to return and spend money. Maybe some of the feedback will be meaningful and they will act on it, but I doubt that they will be motivated by the idea of feedback.

Also, have you consulted the literature on what motivates contributors to review sites? My (admittedly limited) knowledge of this area is that they are motivated by one of the following three things:

a. Social status. Influencing others, being acknowledged as an expert.
b. Altruism. Contributing to human knowledge.

c. Revenge. Complaining about something that went wrong at one meal.

Your strategy involves offering them money (a discount on their next meal). That will motivate some people to respond. But I think most people who are motivated to respond this way will write the shortest, most banal text possible in order to get the discount. Most will probably write something like "good".

We have seen students do projects like this (not exactly the same) several times in the past, and they have never been able to get traction. So, even though you have described a strategy for getting results, this historical experience makes me skeptical that the strategy will succeed.

You are, of course, free to do what you want, and you should do something that you are interested in. This strategic feedback is not part of your grade for SE390. You will be evaluated on your results on Symposium Day in March 2018 though.

My perception is that it would be much less risky, in terms of your grades on Symposium Day, to pursue one of the other project proposals: e.g., mapping underground rivers, watershed simulation game, medical teaching app, Sana, etc. Project proposals such as these describe software that people actually want. And many of these proposals involve greater technical depth than your Restaurant Feedback System proposal.

I'm happy to chat in class today.

4.4 Understanding Project Risk

Many students perceive risk on capstone projects differently than the instructors do. Here are few dimensions of project risk, along with some generalizations about whether students under-estimate or over-estimate risk in each dimension.

DATA. Many historical SE capstone project ideas did not reach their full potential because they required commercially valuable data that the students did not have access to. One way to succeed on such a project is to collaborate with a third-party who has the data.

Students often under-estimate project risk due to data availability.

For example, in SE2013 Team SeaSalt wanted to do a project with airline flight information: they abandoned the project (after hundreds of hours) when they discovered that the data they needed costs upwards of one million dollars. [§5.12.2]

SKILLS. A project idea might require learning a new programming language or technology or concepts. That is ok. You are arguably the best undergraduate software engineering students in the world. You have the intellectual capacity to learn. You have the time to learn.

Students often over-estimate risk due to new skill acquisition.

DIFFICULTY. Some projects present greater technical challenges than others. The instructors know this. Your project grades will be moderated by the difficulty of the challenge your are attempting.

Students often over-estimate risk due to technical challenge.

POPULARITY. Students tend to lean towards 'popular' project ideas (this is, by definition, what makes them popular). We, in collaboration with Velocity, have compiled a list of some such projects [§13]. It is often difficult to produce results with these project ideas — although there are notable exceptions, such as UWFlow [§11.14.2].

Students often under-estimate risk for popular project ideas.

Popular Project Ideas §13

SOCIAL SKILLS. Some project ideas involve software substituting for social skills. For example, a popular project idea is an app to help a group of people decide where to eat lunch. These people do not need software, they need social skills. After third year of university, most people have developed at least some of these social skills, and so their need for such an app declines.

Students often under-estimate risk for projects that substitute software for social skills.

For example, listening, speaking, leadership, compromise, *etc.*

MARKETING. Projects that aim to be evaluated by their user base will probably need to have a marketing plan to build that user base.

Students often under-estimate risk associated with marketing.

UWFlow (SE2014 [§11.14.2] collaborated with students in other faculties, as well as orientation week organizers, to build their user base.

SCOPE. Some student project proposals are too big and some are too small. For example, the general problem of video image search (proposed in SE2016) is too big: this is an active area of research that requires a more focused project definition. Conversely, address book synchronization (proposed in SE2014) is too small.

Students usually propose projects of reasonable scope.

4.5 Professional Communication

Professional communication is essential to your success in the capstone projects — and your future career. Moreover, it is required by the Professional Engineers of Ontario Code of Ethics, specifically:

> *A practitioner must co-operate in working with other professionals engaged on a project.* — Revised Regulations of Ontario 1990, Reg. 941, §77.6

> *A practitioner shall act towards other practitioners with courtesy and good faith.* — Revised Regulations of Ontario 1990, Reg. 941, §77.7(i)

PEO Code of Ethics: http://peo.on.ca/index.php?ci_id=1815&la_id=1

COMMUNICATION IS ESSENTIAL TO AGILE. Professional communication, with both colleagues and clients, is essential to Agile methodologies of software development. Historically, in industry, many software projects have failed due to poor communication. On the one hand, different sub-teams would develop software that could not integrate with code written by other sub-teams — due to poor communication between sub-teams. On the other hand, software engineers would deliver software to clients that never got used — due to poor communication with the client. These kinds of failures are very expensive, in both money and time.

Agile practices, such as *Continuous Integration*, along with improved version control systems, aim to mitigate the possibility of the first kind of failure: code integration.

Agile practices, such as *scrum*, aim to mitigate the possibility of the second kind of failure, by maintaining tight communication with the client and an understanding that the client's requirements might shift as the project progresses. An old saying in software engineering is 'we built what the customer initially asked for, but not what they actually wanted.'

Agile methodologies typically have very limited paperwork, especially as compared to traditional engineering (including traditional software engineering) development practices. Sometimes people miss the positive emphasis in Agile amidst this negative view towards written documentation: *Agile approaches advocate physical co-location, talking, whiteboards, and modern messaging technologies.* If you simply drop the paperwork but do not adopt the other communication practices, then the outcome will likely be worse than if you followed traditional practices of extensive written reports.

Some specific things you need to do:

- respond to meeting requests
- show up to meetings
- participate in the SE390 project selection processes
- log your time regularly (§4.9.2)
- keep external parties in the loop

Additionally, when programming:

- commit your code frequently
- write meaningful commit messages
- write meaningful comments on pull requests
- write meaningful feedback on pull requests

None of this is subtle inter-personal communication. None of this requires empathy, compassion, or understanding of other people. It's not complex. It's just factual communication about work.

Here are some issues that we have observed in the past:

- The team is not communicating with the external party. The external party wonders if anything is happening. It turns out that stuff is happening, but they aren't being kept in the loop.
- The team is not communicating with each other. Some team members simply do not respond to messages or meeting requests.
- Part of the team is communicating with the external party, but the rest of the team is not communicating with them. The rest of the team is off building something else, without knowledge of what the external party actually wants (since the part of the team that knows the requirements cannot get the coders to respond to meeting requests).

4.6 Internal Mini-Project: New Product

For the internal mini-project you are encouraged to pursue a project idea that would fall under the New Product rubric [§6.5.1]: *i.e.*, where the evaluation of results on Symposium Day is based on user engagement (typically number of users).

The mini-projects are evaluated only on effort. Results are evaluated on Symposium Day. This is a low-risk way for you to experiment with new ideas, new technologies, and new people.

4.7 External Mini-Project: FOSS, Research, or Custom Software

For the external mini-project, you are encouraged to pursue a project idea that would fall under one of: contributing to an existing *Free/Open-Source Software (FOSS)* system [§6.5.4]; or a developing some *Custom Software* for a particular client [§6.5.3]; or getting involved with some *research* [§6.5.2].

These mini-projects give you a chance to try out ideas in different directions. You may continue either of them for your capstone project. Ideally, at the end of the two mini-projects, you have the seeds of two viable capstone projects to choose between. Maybe, even then, you'll choose something different: that's ok, because you will have developed skills, wisdom, and perspective from the mini-projects that will help you with whatever project you pursue.

The mini-projects are evaluated only on effort. Results are evaluated on Symposium Day. This is a low-risk way for you to experiment with new ideas, new technologies, and new people.

The research might be in SE/CS or in some other discipline.

4.8 Schedule

Week	*Theory*	*Activity*	*Progress*
w1	Learning Objectives §1 Lateral thinking §2.1	Strategy §4.10 Mini-project brainstorming	Internal mini-project team formation
w2	Normal *vs.* Radical Design §2.9	Symposium Videos	Team formation
w3	The Design Disciplines §2.2 Designer's Core Skills §2.3	Name teams in slides	External Ideas
w4	*Internal Mini-Project Presentations*		
w5	*Thanksgiving Break* *External Mini-Project Team Formation*		
w6	Creating Alternatives §2.4	Mini-project brainstorming	Team Formation
w7	Prototyping §2.6	Symposium Videos	Scrum
w8	*CASCON* (The largest SE/CS conference in Canada.) (Hosted and paid for by IBM Toronto. No registration fee. Attendance encouraged.)		
w9	*External Mini-Project Presentations*		
w10	Creating Alternatives §2.4	Ideation	Team formation
w11	Understanding Project Risk §4.4 How To Choose a Project §4.2	Voting	Scrum
w12	Design Process §2.7	Symposium Videos	Scrum
w13	Definitions of Design §2.8	Symposium Videos	Scrum

4.9 *Marking Scheme*

See §3 for schedule

Individual			
Internal mini-project effort (§4.11)		25%	3 weeks of effort
External mini-project effort (§4.12)		25%	3 weeks of effort over 5 calendar weeks
		50%	
Team			
Capstone Prototype Demo (§4.13)	40%		
		40%	
Class			
Attendance (§4.9.3)	10%		
		10%	
		100%	

4.9.1 *Making an Honest Effort*

In the marking scheme above, the mini-projects are graded purely on effort. Does this mean that you do not have to do the presentations for them? The presentations do not appear in the marking scheme. No, obviously not: you must do the presentations, and you must do certain things by certain deadlines. This is part of an honest effort.

To make it clear that pathological behaviour is discouraged, we will have a system of demerits for unacceptable behaviours. It is the instructors hope and expectation that no demerits will be awarded. The point of having these is to communicate that certain things need to be done at certain times as part of your honest effort.

Note that these are guidelines, not firm rules. There might be good reasons why particular situations occurred, and you can explain them.

In the words of NHL coach John Tortorella: 'I see the starting lineup. I know the guy across the bench. With the lineup that he had; I can't put our players at risk that way. And that's what ensues. It shouldn't be in the game; that stuff. I don't want it in the game. But I have to protect my team. I'm not proud of it. I don't feel great about it at all. But I'd do the same thing again if it came that way.'
https://www.youtube.com/watch?v=LprzxFAoH9w

Individual		
More than 10 days between log entries (in a project)	-1%	
More than 10 hours logged in a single day	-1%	
Broken log file not fixed within 3 days of notice	-1%	
Read external proposals & install browser plugin	-2%	Oct 12
Participate in external ideation	-2%	Oct 16
Not associated with mini-project team in SVN	-3%	Oct 3 + Oct 19
Team		
Mini-project slides break the LaTeX build	-1%	Oct 3 + Nov 7
Mini-project presentation inadequate	-5–10%	

4.9.2 *Time Logging*

Every student will have their own individual log file for each of the three segments of this course (internal mini-project, external mini-project, capstone project prototype). For example, your log for the internal mini-project will have a path of the form:

https://ecesvn.uwaterloo.ca/courses/se_capstone/2018/se390/internal-mini-project/logs/userid.log

You should make a local checkout of the folder for your year, *e.g.*:

https://ecesvn.uwaterloo.ca/courses/se_capstone/2018/

Then you can edit your log file locally, and commit it to the server.

Subversion is better for these log files than Git, because Subversion records the actual userid and server time stamp for each commit. Git, by contrast, records the userid and time stamp as configured on the user's local machine, which might be incorrect.

Each entry in a log file will be one line, starting with a prefix indicating the entry type. There are two kinds of entries: time entries and notes. Time entries start with the prefix '*x*h:', where *x* is an integer indicating number of hours. Note entries start with 'NB:'. Figure 4.1 lists an example log file:

'NB' is a Latin acronym, like '*i.e.*'. It stands for *nota bene*, which literally means 'note well', or figuratively 'pay attention'.

```
1h: interviewing with potential teammates
NB: decided to work with Jessica and Calvin
2h: reading about GPU programming
1h: 2016-09-19 installing GPU libraries yesterday
3h: learning GPU matrix multiplication case study
```

Figure 4.1: Example time log file

Follow these guidelines when editing your log file:

These guidelines are checked by the check-log.sh script in the log directory.

- Each student may write only to his or her own log file.
- Write a line in the log file each time you work on the project.
- If you commit the log entry on the day the work occurred, then you do not need to record the date: it can be extracted from the commit metadata.
- If you commit the log entry after the day the work occurred, then please record the date explicitly, as shown in the example above.
- Do not edit past log entries (*i.e.*, only add to the log file).
- Do not insert log entries in the past. The property that the script checks is that the commits associated with the log entries increase monotonically. It looks at the output of svn blame: *e.g.*, in the following listing the student has inserted a line on commit 2025:

```
$ svn blame student.log
  1849 drayside NB: See the Handbook for formatting and usage guidelines.
  2021 student 1h: Form groups for a common interest
  2022 student 3h: Brainstromideas and agree on the general topic
  2025 student 3h: Debate for the function details
  2023 student 4h: Research on existing projects and re—design to ensure the project is new
  2024 student 2h: Design and draw necessary UI componets
```

The purpose of these rules is to preserve the good properties of an old-fashioned pen+paper based logging system: *i.e.*, that it can be reasonably relied upon as a truthful record.

In the words of the late Mayor of Toronto Rob Ford: 'the past is the past; you can't change the past; I'll take a urine test right now.' https://www.youtube.com/watch?v=JF9jsI8id7E&t=0s

4.9.3 Class Attendance

This is not a traditional lecture-oriented course. This is a design project course. The skills and intellectual growth that you will acquire in this course cannot (easily) be learned just by reading books: they are learned through activity and through social interaction. Moreover, these skills and growth cannot be easily tested in a traditional written exam. Therefore, attendance is crucial to the educational value of this course.

In each meeting we will learn some design *theory* (which you could, in principle, learn from a book), do an *activity*, and discuss your *progress* on your current project.

Attendance will be evaluated for the class as a whole. Each day will be graded according to the percentage of students in attendance, using the standard uw percentage to letter grade conversion scheme. So an A+ day is 95%+ attending.

4.9.4 Synergy with se464

se390 occurs in the same term as se464: Software Design & Architecture. se464 often has assignments or a project that can have some positive synergy with se390. This is intentional: the se Curriculum Committee re-positioned se463 Requirements, se464 Design & Architecture, and se465 Testing to have better synergy with the Capstone courses.

Synergy: (noun) the interaction of elements that when combined produce a total effect that is greater than the sum of the individual elements, contributions, *etc.*

The intended synergy with se464 is that you may use either (or both) of the se390 mini-projects as the se464 project. The objective of this synergy is to enable you to develop better software by focusing your efforts on one codebase. The objective is not to reduce your student workload by giving you double-credit for the same work. So if you pursue this synergy, you need to be forthright and honest in your disclosure with both instructors, and in your presentations/reports discuss how the synergy enabled you to build better software.

4.9.5 Synergy Between Mini-Projects

The three mini-projects (including the initial prototype for the capstone project) in se390 are described in this Handbook as completely independent. The point is to encourage you to explore new ideas, new technologies, and new people. Gaining a sense of perspective and wisdom about project selection and risk is best learned from some experience stepping outside your comfort zone.

Sometimes a team thinks they already know what they want to do for their capstone project before se390. Sometimes that thought even survives to the end of se390 without them changing their minds. If you think that you are in this august group, then the way to navigate the se390 outline is as follows: do you internal mini-project on your proposed capstone project; do your external mini-project on fixing a bug or adding a feature to some library that you will need to use for your proposed capstone project; return to your proposed capstone project for your intial capstone prototype. Easy.

UWFlow from se2014 was in this situation: they had already started on development of UWFlow.com before se390 began. By the end of se390 they had a preliminary version that other students could use.

4.10 Strategy Exercise

One of the intended learning objectives [§1] of the capstone courses is for you to make *strategic decisions*. In this exercise, imagine that you are the CTO or CEO of some major technology corporation, and you must decide on the future direction of the company.

This is a *formative* exercise, not a *summative* exercise. Most of your previous engineering courses had only summative assessments: that is, exercises intended to evaluate your skills and knowledge. The purpose of a formative exercise is to develop your thinking and provide you with feedback.

This is also a *lateral thinking* exercise, whereas most of your previous courses have involved *linear thinking* only.

Edward de Bono. *New Think: The Use of Lateral Thinking*. Basic Books, New York, 1967

4.11 Internal Mini-Project

Make a team, pick a project, get coding. Learn what you can do in a few weeks. Perhaps some wireframes or other experimental prototypes. Perhaps start on an evolutionary prototype. You are expected to pick an idea that would make a good capstone project, but you are not expected to produce a Symposium Day quality implementation.

This might be a good opportunity to learn a new programming language.

This is called the *internal* mini-project because the idea / problem domain originates with you. By contrast, in the *external* mini-project, you will find some direction from the outside world.

Grades for this mini-project will be based purely on effort (§4.9.2).

Grade	*Criteria*
A+	≥ 8 hrs/week, per student
A	≥ 6 hrs/week, per student
B	≥ 5 hrs/week, per student
C	≥ 3 hrs/week, per student
D	< C

Grades for the mini-projects will be assessed individually, in order to encourage you to work with new people, even though it is a group project.

On Symposium Day you will graded on real-world results.

SE390 is a full (0.5) credit course. Like all full-credit courses, it has a nominal budget of 10 student hours per week, including class meetings. The two hours of class meetings per week leave you with eight project hours.

4.11.1 Teams

Pick your teams as you please. Normal size is four. Minimum size is three (mandated by CEAB). Maximum size is six. You are strongly encouraged to work with new people, whom you have not worked with before, but who have common or complementary technical interests. In this beginning phase of the Capstone project, you need to open your mind to new ideas and new perspectives, and working with new people is an excellent way to do that.

You will identify your team in the slides §4.11.2.

4.11.2 Presentation

Your team will have five minutes to present your internal mini-project to the class. This is not much time. It will take some work for you to figure out what is important to say in such a limited time.

To keep the class on track, we will need to have very efficient transitions from one team to the next — otherwise we'll never get to hear from every team. To be efficient and stay on track we will

For your capstone prototype demo at the end of SE390 you will have a private audience with the instructor for twenty minutes. It will take the instructor two full days to meet with every group — time that we do not have in class during the term.

compile every team's slides into a single PDF using LaTeX. The slide files are in this directory of the repo:
https://ecesvn.uwaterloo.ca/courses/se_capstone/2018/se390/internal-mini-project/slides

To CREATE the slides for your team:

a. cd slides

b. svn up

c. cp team-example.tex team-*YourTeamName*.tex

Just the name for this team for this mini-project.

d. edit team-*YourTeamName*.tex

You have the option for separate team and project names, but not required.

- \subsection{Team Name / Project Title}
- \author{userid1, userid2, userid3, userid4}

This is how the instructor knows what team you are on, §4.9.1.

e. edit main.tex

- Find or create an appropriate theme heading
- \input{team−YourTeamName.tex}

f. svn add team−YourTeamName.tex

g. svn commit −m 'creating slides for YourTeamName' main.tex team−YourTeamName.tex

To EDIT your slides you have two main strategies:

- *Edit the LaTeX directly.* We are using the LaTeX package Beamer, which is well documented online.
- *Make slides in another program and export to* PDF. If you do this, you can include them in the LaTeX slides with a command like this:

  ```
  \includegraphics[page=1,width=\textwidth]{team−MyTeam.pdf}
  ```

4.12 *External Mini-Project*

The *external* mini-project is called 'external' because you must interact with the world outside the classroom in some way. There are several ways in which you might do this, including:

- Improve an existing FOSS system. Fix a bug. Add a feature.
- Collaborate with a prof on a project in their resesarch area. Profs outside of CS/ECE strongly encouraged.
- Collaborate with a non-profit organization on their software development needs.
- Participate in a hackathon. The hackathon must have some kind of theme or direction. The hackathon by itself might not give you enough hours, so you might want to develop the idea further outside of the hackathon.

You may choose a FOSS system that you might use on your capstone project.

There must be an element of engineering design in your project. Doing a flat website, or a Drupal install, *etc.*, does not have adequate technical depth.

The instructor will also post some possible external project ideas.

The point of this mini-project is to give you an opportunity to explore an external collaboration without making a long-term commitment. If this mini-project goes well then you might choose to continue it for your capstone project.

FOSS: Fix a bug or add a feature to an established FOSS system. Waterloo's reputation is in your hands: Communication with the FOSS project maintainers is discouraged but not forbidden. If you do reach out to the maintainers, be clear that you are working on this for a third-year class project and describe the approaches you have taken. Please get instructor approval before submitting a patch.

THIRD-PARTY: Third-party project proposals that have been vetted by the instructor will be presented in class. These will likely include ideas from academic, industrial, non-profit, and FOSS sectors. These third-parties understand the capstone project and are actively interested in collaborating with you.

For example, in the past capstone groups have collaborated with the Centre for Theoretical Neuroscience at UW and the Sana mobile health records project at MIT.

TEAMS: It is recommended that you change teams for this mini-project — that the teams be assembled by common interest in the project rather than by pre-existing social connections.

GRADING: Grades will be based on your time, as with the internal mini-project. Keep your time log according to the above instructions, but in the external mini-project log directory.

4.12.1 Idea Wiki

Ideas will be posted by you, your classmates, and the instructor, in a 'wiki' in the course Subversion repo:

https://ecesvn.uwaterloo.ca/courses/se_capstone/2018/se390/external-mini-project/ideas

The files will be in the Markdown text format. It is highly recommended that you install a browser plugin to render Markdown in pretty HTML. Then you can view the files with your web browser.

To ask a question about a project, or to indicate that you want to work on a project, just edit the Markdown file in the appropriate place.

A script will check that you have edited at least one line of one idea Markdown file.

4.13 Capstone Project Demo

4.13.1 Project Selection & Team Formation

We will use a wiki again, and follow a similar process for team formation. Note that this directory is not in the SE390 subdirectory: the intention is that this is the project that you will carry forward to Symposium Day.

https://ecesvn.uwaterloo.ca/courses/se_capstone/2018/projects

An external mini-project team is working to improve our wiki software. You can download their software here: https://github.com/arajeev/se-capstone-db

You may continue either of the mini-projects, or something new.

You may keep the same team as you had on either of the mini-projects. You are encouraged to allow new people to join your team if they are interested in the project.

Why did we do the mini-projects? A few reasons:

- To expose you to new ideas and people.
- To develop your judgement and wisdom through experience.
- We know from past experience that change is going to happen. What you think your team or project is going to be at the start of the term is likely to not last all the way to Symposium Day. The purpose of the mini-projects is to force change to happen early in the process, during SE390, in a safe and supportive environment.
- We know from past experience that student engagement is driven in large part by deadlines. In the past we have sometimes had all of the deadlines for SE390 at the end of the term, and the result was that students did not think seriously about their projects during the term.

4.13.2 *Schedule*

The Capstone projet demo occurs during the exam period. It is considered a group oral exam. It is not scheduled by the registrar's office, since each team will need their own time. Each team needs to schedule a time with the instructor.

4.13.3 *Evaluation*

Team Effort		
Time logs		3 weeks of effort
Lines of code		measured by CLOC
Heartbeat survey		participation in
Reporting		
application of design theory (§4.13.4)		
software architecture (§4.13.5)		
project status sheet (§1.4)		
Software		
Current status of the prototype(s)		this is the historical criteria that SE has always used
	40%	

4.13.4 *Application of Design Theory*

You are expected to apply the design theory covered in §2 to your chosen project. Additionally, you must identify and (briefly!) discuss the risks facing your chosen project (see §4.4). A 1-2 page report on:

- *Design Disciplines* §2.2: What disciplines are required for your project? Do you have the appropriate training? What are your plans to acquire it? Do you have appropriate collaborators?

- *Creating Alternatives* §2.4: Apply at least one of the described techniques to some important aspect of your project.

- *Selecting From Alternatives* §2.5: What decision did you make? Why?

- *Prototyping* §2.6: What prototypes did you build? Why? What kind were they? Display some knowledge of prototyping theory.

- *Definitions of Design* §2.8: Does your architecture report [§4.13.5] talk about the design from the perspective of the software structure, design decisions, and team organization/communication? If some perspective is missing, fill it in here (briefly).

- *Normal and Radical Design* §2.9: Is your design normal or radical? In what respects? Component selection? Component arrangement? Are you following an established architectural style, reference architecture, design pattern, *etc.*?

- *Laws of Software Design* §2.10: Identify and discuss any laws that apply to your project. Classify your project with respect to Lehman's categories (S-type, P-type, E-type).
- *Understanding Project Risk* §4.4: What are the risks facing your project? What are you doing to mitigate and manage them?

4.13.5 Software Architecture

You will, at some point before the end of SE490, need to document the architecture of your system. Is that time now, in SE390? It depends on two things:

- *What kind(s) of prototype(s) are you building? §2.6*
  If you are building exploratory/experimental prototypes that you are planning to throw away, then you do not need to write anything about software architecture yet.
- *How Normal or Radical is your design? §2.9*
  If your design is extremely Normal, then there might not be much to say: the architecture might be totally dictated by your choice of framework.

Eventually you will begin work on your evolutionary prototype: the code base that you will continue to improve and build on through to Symposium Day. That is the time when you need to write a brief (1–2 page) report on the architecture, because that is the time when you are making some commitment to these decisions.

The most important thing for SE390 is that you accurately describe the situation that your project is in, provide adequate justification, and take appropriate action. For example, if you are building a set of experimental prototypes in SE390, you should describe what your team intends to learn from each one, and what safeguards you have in place to prevent them from accidentally morphing into an evolutionary prototype (*e.g.*, using a non-executable wireframe UI mockup tool). If your architecture choices are totally constrained by your technology selection, then you should provide documentation from that technology that describes the standard reference architecture.

Synergy with SE464*:* If your capstone project is your SE464 project, then just submit the report your wrote for SE464 unaltered. In this case, your SE390 time log should also include all of the time you spent on your SE464 project — but you will be expected to have double the number of hours of a SE390 project that is not also used for SE464. The point of cross-course synergy on the project is to give you the opportunity to build better software, not to give you double credit for academic work (which is against university policy).

se490 *Outline*

The calendar description of se490 is:

> Continuing from se390, students undertake a substantial customer-driven group project. Project groups establish and maintain project control processes, delivering a series of iterations on their se390 prototype. Adaptive methods are encouraged and supported.

This course has a few differences from the standard course format: all work is done in groups; there is no midterm test; there is no textbook.

5.1 Objectives

The goal of se490 is to develop version 1 of your project — software that can be used to do something in September. That usage might mean actual end-users for a new product, or it might mean you taking performance measurements on an algorithms project. When you return to campus in January for 4b you should be working on version 2, and at Symposium Day in March you will be reporting on your successes and lessons learned.

The first two classes are devoted to planning exercises. These exercises are intended to give you fresh perspectives on planning and on your project.

The first half of the term may, at the team's discretion, be devoted to experimental and exploratory prototypes — prototypes where the value is the knowledge gained rather than the code written.

Once the team moves into evolutionary prototyping mode they are expected to put proper professional infrastructure into place: a specification, a build system, test harnesses, a bug-tracking database, *etc.*. Teams are expected to have this infrastructure established and be in evolutionary prototyping mode by the midterm demo.

There will be three rounds of formal peer interaction through the term. In the first half you will propose design alternatives for other teams. In the second half you will attempt to use another team's evolutionary prototype and you will critique their code.

5.2 *Marking Scheme*

See §3 for schedule

Classroom		
Initial status sheet completed (§1.4)	1%	
Old stories told in new ways (§7.1)	2%	
Heartbeat surveys completed	2%	
Risk Identification & Prototype Plan (§2.6)	5%	
Peer Design Exploration	5%	
Peer Usability Review	5%	
Peer Code Review	5%	
		25%
Demos		
Midterm Demo	25%	
Final Demo	50%	
		75%
		100%

A few times in the term we will do a simple anonymous *heartbeat survey* to see how you are feeling about your project and team. It is expected that you feel some uncertainty at the beginning of the term. The purpose of the survey is to give you a way to express that, and to see that the feeling is common.

This grade will be assessed on the class collectively according to the rate of participation in the surveys.

5.3 *Initial Status Sheet*

Status Sheets described in §1.4

□ Project selected
□ Abstract
□ Initial selection of results rubric
□ Initial results targets
□ ATE Plan

5.4 *Midterm Demo*

Requires a Status Sheet (§1.4)

□ Revised choices for results rubric, results targets, *etc.*
□ Requirements identified (see Status Sheet)
□ Iteration/Prototype Plan
□ Proposed design & architecture
□ Infrastructure: version control, build system, test harness, issue tracker
□ Functionality: MVP (possibly in exploratory prototypes)
□ Productivity: roughly 2,000 LOC (see Grading Philosophy §1.3)

5.5 *Final Demo*

Requires a Status Sheets (§1.4)

A *response* could be a refutation: you do not need to follow all advice that you are given, but you do need to explain why you are not following it.

□ Responses to feedback from:

  □ Peer code review
  □ Peer usability review
  □ Customer feedback
  □ Instructor feedback

□ Testing strategy/tools/coverage/*etc.*
□ Revised Iteration/Prototype Plan
□ Requirements mostly satisfied (see Status Sheet §1.4)
□ Functionality: V1 (ready for beta testing)
□ Productivity: roughly 5,000 LOC (see Grading Philosophy §1.3)

5.6 *Risk Identification & Prototype Planning*

Files:
prototype-plan/*

a. Learn about different kinds of prototypes in §2.6
b. Identify key technical and non-technical risks for your project
c. Determine which kind of prototype to use to mitigate each risk
d. Think of any other prototypes that might be helpful
e. Organize the order of building the prototypes into a plan

A key principle of agile project management is that the plan will change as the project progresses and new things are learnt. At the midterm demo we might have a retrospective discussion on how and why things changed, but you will not be required to stick to this plan.

Figure 2.7 tells the story of the seL4 microkernel as a prototype plan

5.7 *Peer Design Exploration*

Files:
peer-design.[txt | pdf]
peer-design-feedback.[txt | pdf]

In this in-class exercise every team will discuss a design challenge in their project with other teams. Every team is expected to provide, in advance of class, a brief (one page maximum) description of at least one (possibly two) design challenge(s). This description may be diagrammatic or textual or some combination thereof. The challenge might concern *fitness for purpose* or *fitness for future*. When in consulting role, each team is expected to provide at least two alternative solutions to the other team's presented design challenge.

Class time will be divided into two 50 minute halves, which will in turn be sub-divided into two 25 minute quarters. In the first quarter team *A* will describe their design challenge to team *B*, who will provide at least two solutions. In the second quarter they will switch roles and team *A* will provide solutions to team *B*'s design challenge. In the second half team pairings will be rotated.

5.8 *Peer Usability Review*

Files:
peer-usability.[txt | pdf]
peer-usability-feedback.[txt | pdf]

For this in-class exercise you will have the opportunity to get usability feedback from your classmates. If your project does not involve a usability component then your grade will be determined by the quality of the feedback you give to other teams.

Class time will be divided into 15-20 minute segments, and teams will rotate to new partners for each segment.

Teams gathering usability feedback are expected to provide, before class starts, a brief write-up of what they hope to learn about, what methods they intend to use, and what questions or activities the will be requested of the users. It is expected that most teams will be focused on *formative evaluation* (*i.e.*, getting feedback on how to improve an incomplete design) rather than *summative evaluation* (*i.e.*, measuring a completed design). Methods might include *think-aloud*, *participatory design*, *interviews*, *focus groups*, *A/B testing*, *concept testing*, *usability benchmarking*, *etc.* Teams with usability concerns in their projects are expected to enroll in a usability course or otherwise educate themselves in this area: teaching this material is beyond the scope of the SE Capstone courses.

http://www.measuringu.com/blog/formative-summative.php

http://www.nngroup.com/articles/which-ux-research-methods/

http://www.nngroup.com/articles/thinking-aloud-the-1-usability-tool/

https://en.wikipedia.org/wiki/Think_aloud_protocol

Teams are expected to respond to this peer usability feedback by the SE490 final demo. As indicated on the status sheet, this response could be either to incorporate or refute the feedback.

5.9 *Peer Design Review*

Files:
./ReviewedTeam/design.pdf
./ReviewedTeam/design-ReviewingTeam.[pdf|txt]

In this in-class exercise you will apply ideas you have learned in SE464 to reviewing project design from two other teams. These ideas from SE464 will include design patterns, architectural styles, code smells, and refactoring. Your team will receive reviews from two other teams. Your report should be a one page diagram and one page of text covering the following three issues:

This exercise will span two class meetings. In each meeting you will review a team and be reviewed.

- *Explanation of the Design.* What are the compenents? What libraries and tools are used? How are they arranged? Any noteworthy uses of architectural styles or design patterns? *etc.* This exercise is focsed on the internal design and architecture of the software — not the user experience (that is covered in the Peer Usability exercise).

$\frac{1}{3}$ page text + 1 page diagram

- *Fitness for Purpose.* A rationale for why this design meets the specification. Why is this design better than reasonable alternatives? If the project is in a known domain with a known solution strategy, is it following normal design conventions? If the project is in a novel domain or has a novel solution strategy, why is the proposed design a good match?

$\frac{1}{3}$ page text

For example, UW Flow (SE 2014) made a design error by using MongoDB for data that would have been better stored in a relational database.

- *Fitness for Future.* What are the anticipated kinds of change, growth, or variability in the domain? Does the design facilitate managing that change in a modular fashion? What are core assumptions that cannot be changed?

$\frac{1}{3}$ page text

For example, using the Visitor design pattern in a compiler enables new analyses and transformations to be added to in a modular fashion.

Design Review. As the reviewing team, your job is to think critically about the other team's design:

up to 1 page text

- Does the design report address the questions above?
- Is the design fit for purpose? Does it meet the specification?
- Is the design fit for the future? Can it grow modularly?
- Does the code implement the design? *(look at the code!)*
- Is the design solving the right problem?
- Are there design alternatives that should be considered?
- Are there coding issues that should be addressed?

5.10 *Research Literature Report (optional)*

Files:
research.pdf

Identify an important facet of your project and explore its prior art in the research literature, patent literature, and commercial practice. For each relevant and important instance of prior art that you find, include an abstract and an analysis in your report. For research papers or patents the abstract you include could be the abstract from

For example, movie rental websites and dating websites both use recommendation systems. What are some of the ways in which these differ? For most people, the number of movies they watch is orders of magnitude larger than the number of people they date. These kinds of differences in the empirical data might (or might not) lead to differences in the kinds of algorithms that are appropriate.

the paper (appropriately quoted and cited). Your analysis should discuss the similarities and differences between the prior work and your project. Some common topics from the past have included data synchronization, recommendation systems, version control, and distributed algorithms.

This (optional) report counts towards your team's productivity.

5.11 *Formal Logic Analysis (optional)*

Construct a formal logic model of some aspect of your design and run mechanical analyses on it. Potential tools for this task include Alloy, Spin, and TLA+. This approach can be particularly important for distributed algorithms or NP-complete problems.

This (optional) activity counts towards your team's productivity.

Amazon's web services group now makes regular use of these techniques. http://dl.acm.org/citation.cfm?id=2699417

http://alloy.mit.edu

http://spinroot.com

https://en.wikipedia.org/wiki/TLA%2B

5.12 *Changing Projects*

You are allowed to change your project at any time. However, your grade is generally based on the project you present at each time point, and the standards applied at each time point remain fixed. The later you change your project, the more difficult it will be to get back on track, and the greater risk you put your grade at. Hence, it is important to pick a good project to start with.

On the other hand, you should not stick with a project that you no longer believe in. Pivot quickly.

It is possible to receive some credit for a project that has been abandoned if substantial effort was invested in it. This is especially true if the previous project had to be abandoned due to external factors that were beyond your control and which you could not have reasonably predicated. To receive credit for an abandoned project, submit a post-mortem report to the instructor. This report should include a project status sheet completed for the time of abandonment as well as accopanying text to elaborate on the status sheet and explain why the project was abandoned.

A good example report was written by Team SeaSalt in SE2013: https://ecesvn.uwaterloo.ca/courses/se_capstone/handbook/docs/project-change-se2013-seasalt.pdf This example report pre-dates the current project status sheets, but speaks to the important issues.

The remainder of this chapter discusses some specific stories of teams who changed their projects and why.

5.12.1 *Entrepreneurial Success*

SE2013 Team JSTD switched projects because their original project was 'too successful'. One of the team members took a leave of absence to work on the project full time. They got investors, lawyers, users, *etc.* At the beginning of 4B they were looking for a second round of investors, and became concerned that if they spoke about their project publicly on Symposium Day it would spook the potential investors they were courting. So they invented a new project at the beginning of 4B to present on Symposium Day.

The original project, which is still an ongoing business, was Tunezy.com. As described in their SE390 proposal:

> This project addresses the problem of fans discovering new musical talent that is not mainstream, as well as a place for musicians to be discovered.
>
> Many online social networks, or musician tools are saturated with very popular, signed, and mainstream artists. If a musician is talented, it will be hard for people to discover them as they are hidden under all the popular artists. As well, some sites such as YouTube are also filled with other music/videos that push musicians further into undiscoverable space.
>
> This solution generates random environments that can be slightly altered by parameters such as size, location and terrain, or modelled after certain templates like a metropolis, town, megalopolis, etc... which can follow trends of different parts of the world. These environments are generated procedurally, so they require no artists, 3D modelers or software developers to generate an environment, this application simply just needs to be run. There will be the option of allowing the application of your own skins and textures for the environment, but this would be optional. Everything will be generated on the fly, and will thus not need mass storage to store this environment.
>
> This solution is a site mainly for the purpose of musicians and fans interacting. Fans will want to discover new music, and musicians will want to be discovered. This allows fans to interact with the musicians more directly, and for vice versa. As well, this will only allow unsigned musicians, so mainstream artists will not saturate the user base.

On Symposium Day they presented the following alternative project:

> Schoolax is a one-stop campus events hub, initially targeting students at the University of Waterloo. Schoolax's goal is to get students to be more involved with campus life by giving them a central location to search for campus events; our database of events range from career info sessions to Krav Maga club meets, from Eng Soc pub crawls to CIF open gyms. Students earn tokens by attending various events. These tokens can be redeemed on Schoolax for rewards.
>
> Schoolax gathers information from various University of Waterloo websites and clubs. A majority of events are scraped from various official sources such as UW Athletics as well as the Centre for Co-op Education, among many others. In addition, we are vigourously pursuing partnership opportunities with clubs and organizations to use Schoolax as their official hub for communicating with members about upcoming events.

5.12.2 Difficulty Acquiring Data

SE2013 Team SeaSalt began with the following project:

Text provided by James Rossy, member of Team SeaSalt.

> A system that predicts how flight prices fluctuate over time for popular routes and automatically alerts consumers when good prices are available for routes they're interested in.

After investing over 250 hours in the project they decided to abandon it. Accurate price predictions require lots of historical data about flight prices and a reliable stream of new data. Only a handful of organizations control access to that flight data and bidding for it starts in the hundreds of thousands of dollars. As a cheaper alternative, the team built a scraper to collect data from online sources but it became evident that such a solution wasn't reliable and couldn't scale. In 4a they started a new project to build a multi-platform dish review service, somewhat analogous to Yelp or Foodspotting (see Too-Salty.com).

5.12.3 Problem Too Hard

2014 Team Satisfaction started with the following project:

> SAT Solver on GPUs

This project turned out to be too hard. Multiple research groups have attempted to solve this problem without making significant headway. The team knew this heading in to the project, but bravely tackled something really hard. In 4a they pivoted to a related but more tractable problem (still with the same customer):

> Team Satisfaction is engineering a competitive open-sourced parallel side-processing framework for satisfiability solvers. The team has completed a prototype of an architecture that extends a serial conflict-driven solver with modular formula simplification algorithms that execute asynchronously.

On way to look at this pivot is that they switched from attempting a data-parallel approach (GPU) to SAT to a task-parallel approach (CPU). The knowledge and skills they had gained pursuing the first approach prepared them to tackle the second approach. Little effort was wasted.

Risk in project selection is sometime perceived differently by students and by faculty. Team Satisfaction selected a project with high intellectual risk: it was a really hard problem (still unsolved as of this writing). Students sometimes perceive high intellectual risk to correlate to high academic risk: *i.e.*, if the problem is too hard, then you might get a bad grade. Faculty assess risk differently. Faculty would consider this a low academic risk project because it is an intellectually rich area with a committed customer. The chances of getting a good grade are very high; even if the original proposal doesn't pan out, it will be easy to pivot to a related project.

From the faculty perspective, a project proposal that lacks intellectual depth and has no committed customer is at greater risk of getting a poor grade. However, students sometimes consider projects like this to be low academic risk, because they are intellectually easy and have reduced external unknowns (*i.e.*, customer). These are the kinds of conditions that lead to low ambition and low accomplishment and low assessment.

5.12.4 Couldn't Agree with Customer on IP Terms

se2012 Team unSchool started with the following project that had been proposed by an external customer associated with the Young Social Entrepreneurs of Canada:

> Online communication system for financial education. The system is to provide means for the instructors and students, and students amongst themselves to effectively communicate with each other. Quests (assignments), submissions and verifications (markings) are also done within the system. Virtual realities will be constructed in the system to motivate the students, and to provide them with a variety of experiences. Class-against-class or region-against-region competition could be held through the system.

Towards the end of 3B the students had difficulty coming to an IP agreement with the customer. Working with a potential startup presents different IP concerns than working with an established business with an established revenue stream. Both sides had reasonable positions, but were unable to come to an agreement. The student team presented the additional challenge that they did not speak to the customer with a unified voice: different team members would say/email contradicting things.

Over the 3B work term they switched to the following project:

> *Problem:*
>
> *Objective:* Our client is seeking an automated system that notifies members of a design team of any relevant change in the design documents.
>
> *Elaboration:* The company our client works at usually has teams of size 10 20 working on pipeline designs. Each team member, after making changes to his/her design document, is supposed to notify the rest of the team of this change, so the others could modify their files accordingly. But the ball is often dropped, resulting in conflicting designs. Our client tasks us in creating an automated system which would ensure these notification would get sent out.
>
> *Our Solution:*
>
> *Summary:* We build a notification system, with document dependency, on top of a version control system.
>
> *Structure:* Users are members of a number of projects, and can upload documents under these projects (teams). The user becomes the owner of the documents he/she uploads. An uploaded document may have dependencies on other documents under the same project,

> regardless who the owners of the depended documents are. Only the owner may modify the document and its dependencies. If an owner modifies a document and commits, a notification is sent to the owners of documents which depend on the modified document.

5.12.5 Couldn't Make Up Their Mind

2013 Team Mastodon started with this project:

> The idea for our team is to create a website (and possible mobile app) for "Must see, hear, read" media – like movies, music, and books. The idea is to keep a list of media for each category that users can up-vote or down-vote as media that is a "must". A user can also keep track of personal lists of media that they themselves want to see/hear/read sometime soon. There would also be the ability to recommend an item to a friend so that they have a list of "friend recommended media". There could also be the ability for users to post on facebook/twitter what media they've read/watched/listened to and recommend it to friends on facebook, or suggest friends look at their recommendation lists. Other possible extensions are having different filters for recommended categories – i.e. Top Movies of 2010 or Top Books of the 1950s.

By the beginning of 4a they had lost enthusiasm for their original project. Eventually they switched to:

> We're creating a piano learning game/tool which allows piano players to learn musical theory and practice playing by ear, sight reading, and learning pitch. We'll be using a midi adapter to convert music played by the user into a file to compare with the original file to give the user real-time feedback of what they play. We'll need a database of midi files representing music that we can query by difficulty, type, etc.

5.12.6 Too Many Good Ideas

se2016 Team Parallax did a different project each term. All of them were excellent. Their final project, a debugger for WebGL, won an honourable mention (§11.16.6) even though they had only started working on it in December. In se490 they did this instead:

> The Minecraft popularity explosion has made making simple structures out of blocks into an exciting and enviable pastime, beyond the likes that LEGO has enjoyed. While creating abstract structures out of simple blocks remains a highly creative process, the opposite presents an interesting mathematics problem - the deconstruction of 3D models into building blocks, and the reconstruction of the model with instructions may be deterministic. The *Parallax Project* explores block construction using a specified set of LEGO blocks in order to generate building instructions that optimize structural and aesthetic constraints in the input model.

They had the system fully implemented: conversion of a 3D image to Lego voxels, and then assembling those voxels into legal Lego bricks in a connected structure. This latter task was accomplished by using a SAT solver. They found this SAT solver approach worked well conceptually but didn't scale, and so decided to change projects over the work term.

Previously, in SE390, they had planned to do a project on an AI system that could answer elementary school physics questions.

5.12.7 Strategic Re-positioning

Team Kaze from SE2016 started with the following project:

> Avizu is a web platform designed to make creating, sharing, and finding detailed calenders easy. The primary principle behind Avizu is the ability to share calenders with others or find calenders that you feel would interest you, and later be able to recieve notifications and changes as these calenders grow and evolve.

During the discussion at their SE490 midterm demo they realized the strategic weaknesses of this project:

- lots of well-funded competition (*e.g.*, Google Calendar, Microsoft Outlook, *etc.*)
- user base so large that it lacks cohesion
- no obvious way to market the software

After the SE490 midterm demo they came up with a project that addressed all of these strategic limitations. They came to Symposium Day with almost 5000 returning users and won an award (§11.16.2).

> *LolPredict, League of Legends Game Analytics:* An analysis tool for the popular online game League of Legends. The system is designed to help players analyze previous games and, using personalized trends, generate suggestions on the optimal way to approach a current game or how best to improve in the future.

What were the strategic advantages of this new project?

- well-defined user-base
- large, relatively cohesive, user base (in the millions)
- obvious value proposition for users
- easy and obvious marketing channels: Reddit, *etc.*
- low competition: Riot Games had just released the data API for League of Legends, and not many people were using it yet

Kaze was brave to make this change. They had already written over 6 KLOC on the old project, which they completely abandoned as a sunk cost. Their courage and strategic re-positioning really paid off.

5.13 Changing Teams

It is rare that students change teams after the first few months of SE390. Usually if this happens it is not a good situation. There were three teams in SE2016 that disbanded: that is, each member went to join a different team. This was an anomalously high number. This does not happen in most cohorts. Each had their own reasons:

- Team X lost a team member to an overseas exchange or some other academic arrangement. That team member, however, was the main proponent of the project idea. The others were not that interested in the idea, so they disbanded and joined other teams.
- Team Y had a hard time deciding on a project idea. Whatever they had selected at the end of SE390, they wanted to do something else by the beginning of SE490. They floundered for a month or so in SE490 looking for a project, and eventually finding what appeared to the instructor to be a good research project with good faculty support. More than halfway through SE490 they decided that the open-ended nature of research was not for them and disbanded.
- Team Z disbanded during the last work term, in between SE490 and SE491. Some team members felt that others were not pulling their weight, or other irreconcilable views working together.

The later this kind of team change or disbanding happens, the more stressful it is for the students. It is important to use SE390 to find an idea that everyone is interested in and comfortable with. It is important to use SE390 to find a team that can work together effectively.

Being timid in SE390 and making what you perceive to be the most conservative or safe choices is not an effective strategy for avoiding these problems. In SE390 you should be adventurous: try different ideas and different teammates. Go through a rigorous exploration process. Then, from a position of wisdom informed by experience, make solid decisions that will carry you through the capstone project.

SE491 *Outline*

The calendar description of SE491 is:

> Final implementation, testing, and communication of the design project started in SE390. Technical presentations by groups. Analysis of social, legal, and economic impacts. Final release of the project. Project retrospective.

SE491 Grading Scheme:

Classroom		
Impact on Society Report	5%	
$n+1$ Cohort Feedback	5%	
Peer Review Exercise	5%	
		15%
Symposium Day		
Results	25%	
Technical	40%	
Communication	20%	
		85%
		100%

All grades are assessed per team, not per individual. Individual grades will only be assessed when a team, or some part of a team, requests it. Outside of student initiated requests, the Instructor does attempt to determine individual contributions to the project.

Grades are typically assessed by letter (*e.g.*, A/B/C), and then converted to numerical grades [§3].

6.1 *Referee Selection*

Grading on Symposium Day is done by a panel of (usually three) referees. Each team may nominate who they would like to evaluate them on Symposium Day. The Instructor is likely to accept nominations of any candidate who is not an undergraduate student, but the following categories of candidates are particularly encouraged:

If the student team does not nominate at least three referees, or if some of the nominated referees are not available, then the Instructor will coordinate referees from amongst UW faculty and staff and SE alumnus.

- Faculty or staff (from any university)
- SE alumnus
- Graduate students in a related area of research
- The 'Customer' for the project — an external party who has used the software.

Any personal connections to nominated referees should be declared to the Instructor.

The Instructor retains final discretion on grades. This oversight is

For example, if one referee on a panel of three is a significant outlier, the Instructor may choose to ignore the outlier, accept only the outlier, take an average, or some other resolution.

typically exercised when there is significant disagreement on a referee panel, or when one panel of referees returns grades significantly out of line with other panels or with historical norms.

6.2 Peer Review Exercise

Each team can choose what kind of exercise they want to participate in, such as usability (§5.8), technical design (§5.9), or something unique. The point is to gather meaningful feedback and insight from a peer team.

The Instructor will endeavour to make teams from other Engineering or Computer Science programmes available as peers, in order to get a wider perspective. This will involve also providing appropriate feedback to these peers in other programmes.

6.3 $n+1$ Cohort Feedback (Retrospective)

Apply the lessons that you have learned through the Capstone Design Project to providing guidance for students in the next cohort, who have just completed SE390 before you started SE491. Your feedback to them will include:

Mechanism for submitting and distributing the n+1 cohort feedback reports

- Your project abstract.
- Three things you learned through the process.
- One point from the Handbook that, in retrospect, you find interesting or valuable or erroneous.
- Things that you find unclear or missing about their abstract.
- Identification of risks that you perceive for their chosen project.
- Recommendation as to whether they pursue their chosen project or consider alternatives, possibly from the list of mini-projects and candidate external projects from their cohort.

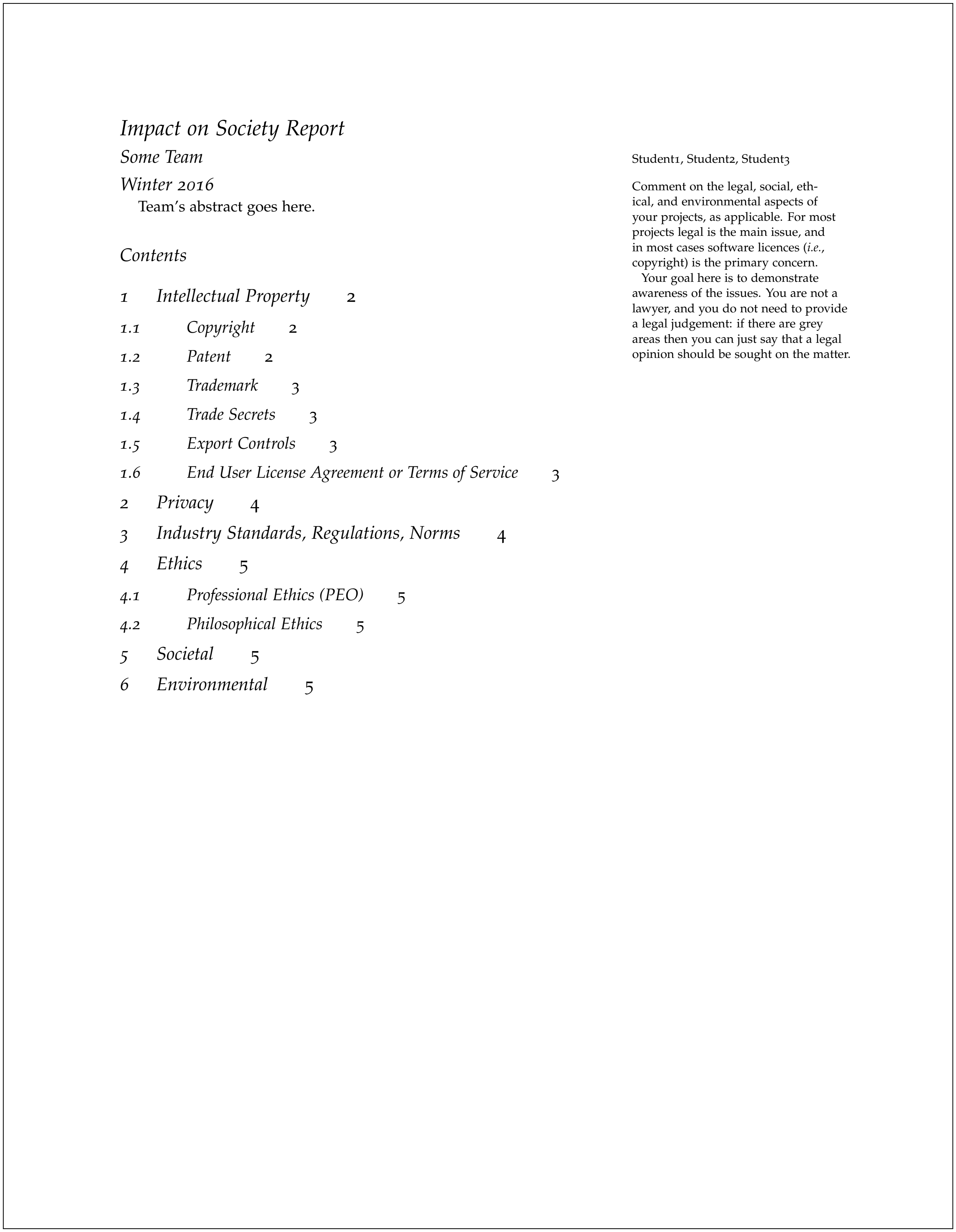
6.4 Impact on Society Report

Each team is required to complete an *Impact on Society Report* by completing the report template listed on the next five pages. The report template covers intellectual property law, privacy law, industry standards & regulations, ethics, general societal implications, and environmental considerations.

You will be provided with the LaTeX files for the report template.

It is typically the case that not all sections of the report are especially relevant for every project. For example, if your project does not store any user data then the privacy concerns are limited.

It is your professional duty to raise any relevant legal, ethical, societal, or environmental issues that are not covered by the template.



Impact on Society Report

Some Team

Winter 2016

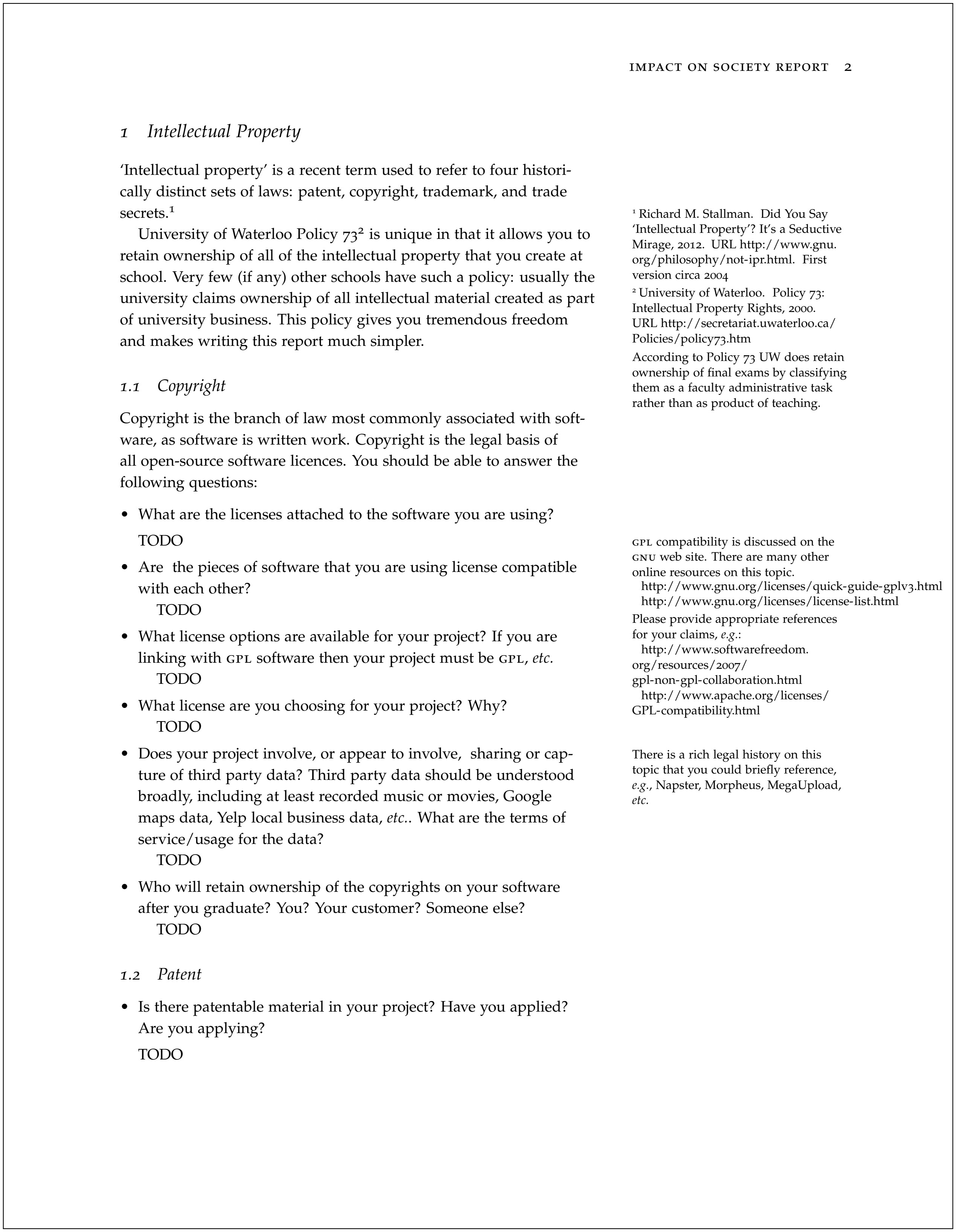
Student1, Student2, Student3

Team's abstract goes here.

Comment on the legal, social, ethical, and environmental aspects of your projects, as applicable. For most projects legal is the main issue, and in most cases software licences (*i.e.*, copyright) is the primary concern.

Your goal here is to demonstrate awareness of the issues. You are not a lawyer, and you do not need to provide a legal judgement: if there are grey areas then you can just say that a legal opinion should be sought on the matter.

Contents



IMPACT ON SOCIETY REPORT 2

1 *Intellectual Property*

'Intellectual property' is a recent term used to refer to four historically distinct sets of laws: patent, copyright, trademark, and trade secrets.[1]

University of Waterloo Policy 73[2] is unique in that it allows you to retain ownership of all of the intellectual property that you create at school. Very few (if any) other schools have such a policy: usually the university claims ownership of all intellectual material created as part of university business. This policy gives you tremendous freedom and makes writing this report much simpler.

1.1 *Copyright*

Copyright is the branch of law most commonly associated with software, as software is written work. Copyright is the legal basis of all open-source software licences. You should be able to answer the following questions:

- What are the licenses attached to the software you are using?

  TODO
- Are the pieces of software that you are using license compatible with each other?

  TODO
- What license options are available for your project? If you are linking with GPL software then your project must be GPL, *etc.*

  TODO
- What license are you choosing for your project? Why?

  TODO
- Does your project involve, or appear to involve, sharing or capture of third party data? Third party data should be understood broadly, including at least recorded music or movies, Google maps data, Yelp local business data, *etc.*. What are the terms of service/usage for the data?

  TODO
- Who will retain ownership of the copyrights on your software after you graduate? You? Your customer? Someone else?

  TODO

1.2 *Patent*

- Is there patentable material in your project? Have you applied? Are you applying?

  TODO

[1] Richard M. Stallman. Did You Say 'Intellectual Property'? It's a Seductive Mirage, 2012. URL http://www.gnu.org/philosophy/not-ipr.html. First version circa 2004

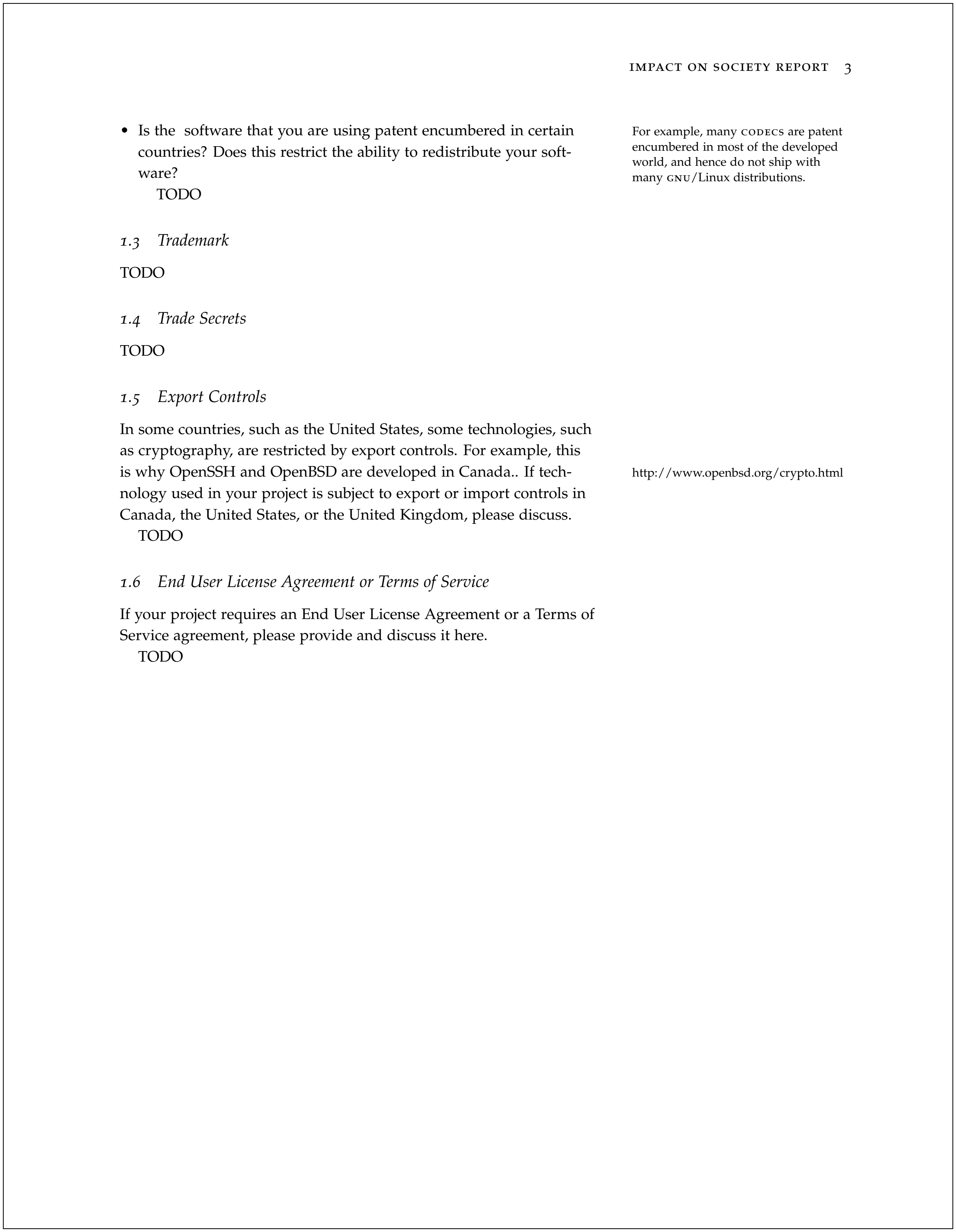
[2] University of Waterloo. Policy 73: Intellectual Property Rights, 2000. URL http://secretariat.uwaterloo.ca/Policies/policy73.htm

According to Policy 73 UW does retain ownership of final exams by classifying them as a faculty administrative task rather than as product of teaching.

GPL compatibility is discussed on the GNU web site. There are many other online resources on this topic.
http://www.gnu.org/licenses/quick-guide-gplv3.html
http://www.gnu.org/licenses/license-list.html

Please provide appropriate references for your claims, *e.g.*:
http://www.softwarefreedom.org/resources/2007/gpl-non-gpl-collaboration.html
http://www.apache.org/licenses/GPL-compatibility.html

There is a rich legal history on this topic that you could briefly reference, *e.g.*, Napster, Morpheus, MegaUpload, *etc.*



- Is the software that you are using patent encumbered in certain countries? Does this restrict the ability to redistribute your software?

  TODO

For example, many CODECS are patent encumbered in most of the developed world, and hence do not ship with many GNU/Linux distributions.

1.3 Trademark

TODO

1.4 Trade Secrets

TODO

1.5 Export Controls

In some countries, such as the United States, some technologies, such as cryptography, are restricted by export controls. For example, this is why OpenSSH and OpenBSD are developed in Canada.. If technology used in your project is subject to export or import controls in Canada, the United States, or the United Kingdom, please discuss.

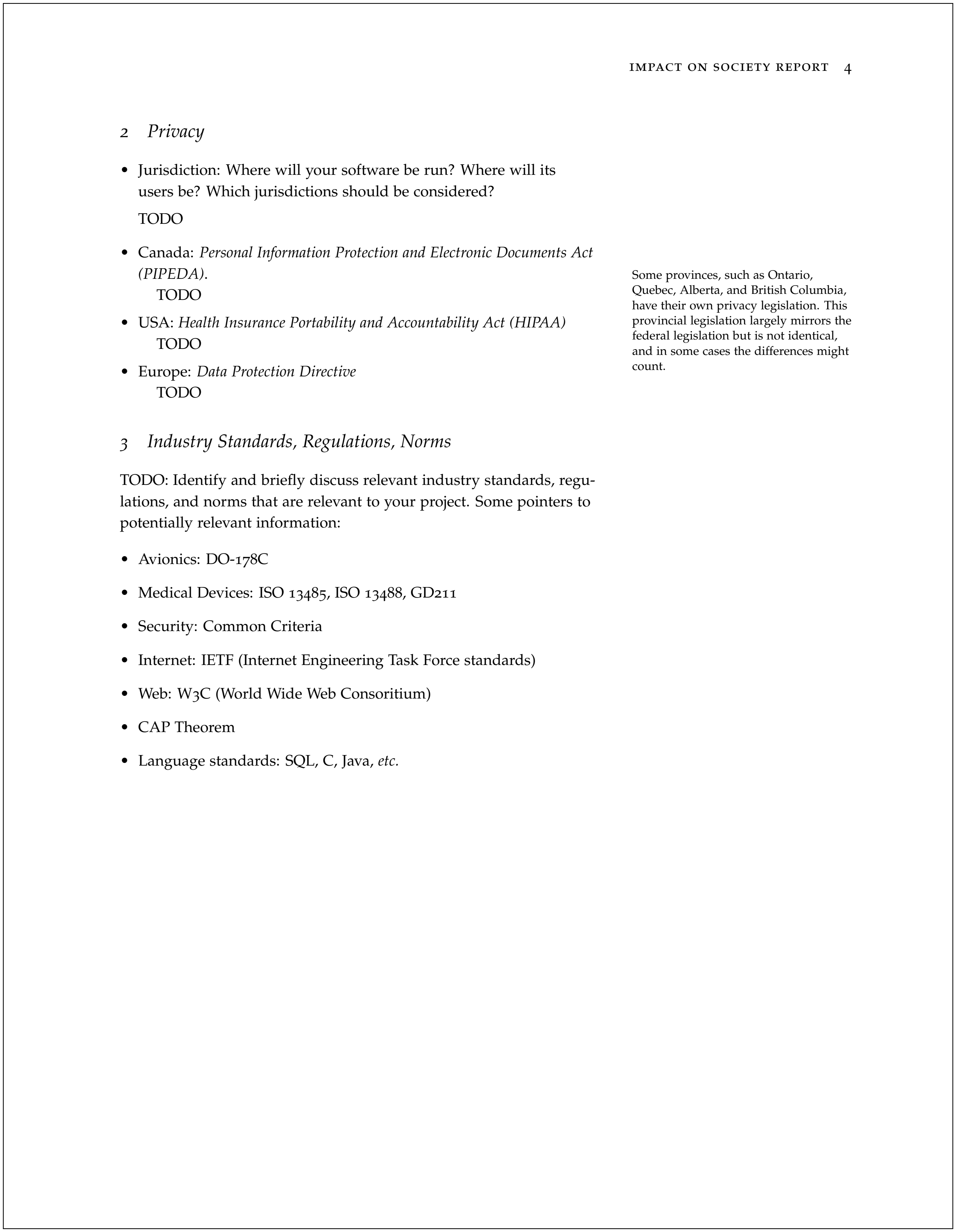
http://www.openbsd.org/crypto.html

TODO

1.6 End User License Agreement or Terms of Service

If your project requires an End User License Agreement or a Terms of Service agreement, please provide and discuss it here.

TODO



2 *Privacy*

- Jurisdiction: Where will your software be run? Where will its users be? Which jurisdictions should be considered?

  TODO

- Canada: *Personal Information Protection and Electronic Documents Act (PIPEDA)*.
  TODO

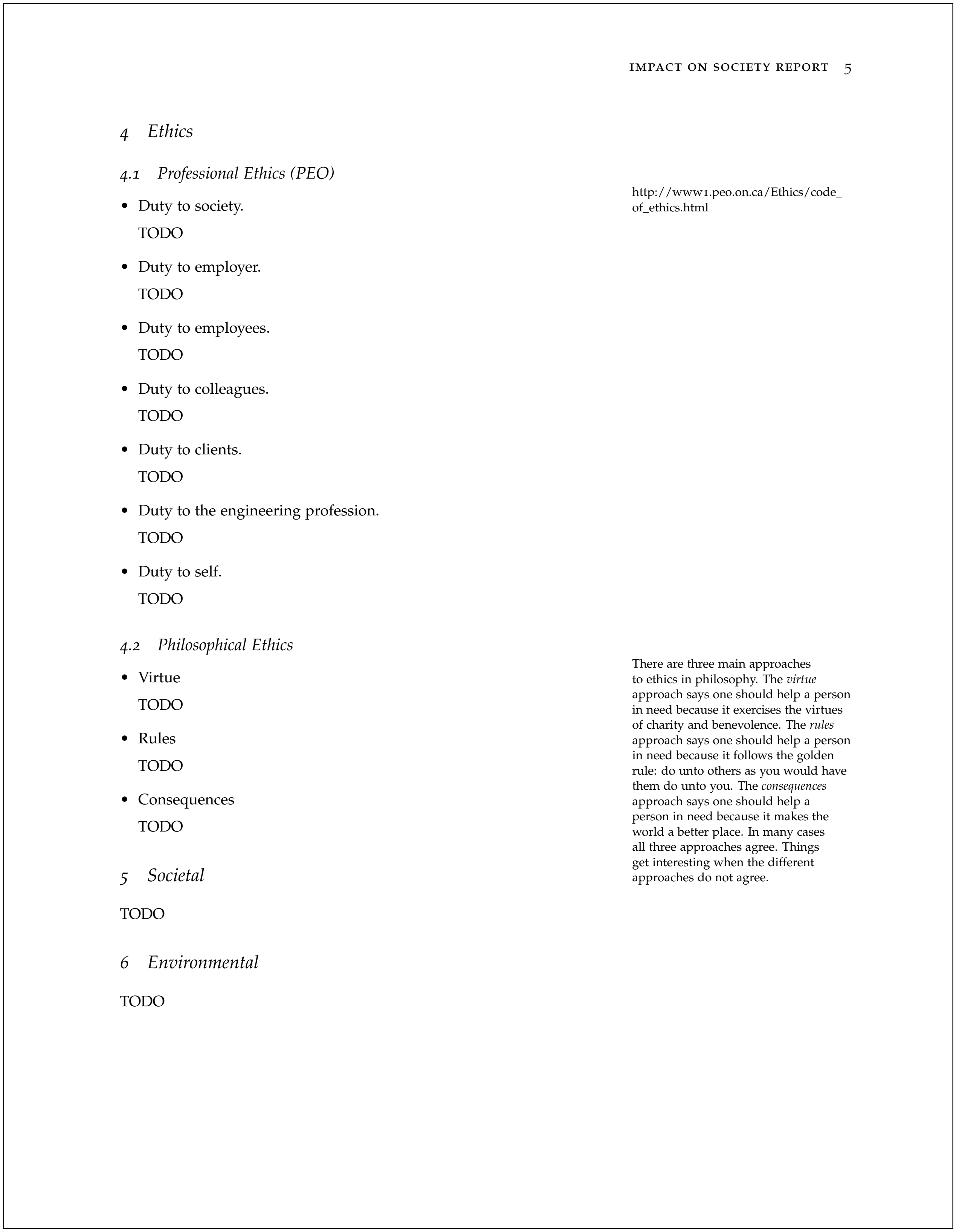
Some provinces, such as Ontario, Quebec, Alberta, and British Columbia, have their own privacy legislation. This provincial legislation largely mirrors the federal legislation but is not identical, and in some cases the differences might count.

- USA: *Health Insurance Portability and Accountability Act (HIPAA)*
  TODO
- Europe: *Data Protection Directive*
  TODO

3 *Industry Standards, Regulations, Norms*

TODO: Identify and briefly discuss relevant industry standards, regulations, and norms that are relevant to your project. Some pointers to potentially relevant information:

- Avionics: DO-178C
- Medical Devices: ISO 13485, ISO 13488, GD211
- Security: Common Criteria
- Internet: IETF (Internet Engineering Task Force standards)
- Web: W3C (World Wide Web Consoritium)
- CAP Theorem
- Language standards: SQL, C, Java, *etc.*



4 Ethics

4.1 Professional Ethics (PEO)

http://www1.peo.on.ca/Ethics/code_of_ethics.html

- Duty to society.

  TODO

- Duty to employer.

  TODO

- Duty to employees.

  TODO

- Duty to colleagues.

  TODO

- Duty to clients.

  TODO

- Duty to the engineering profession.

  TODO

- Duty to self.

  TODO

4.2 Philosophical Ethics

There are three main approaches to ethics in philosophy. The *virtue* approach says one should help a person in need because it exercises the virtues of charity and benevolence. The *rules* approach says one should help a person in need because it follows the golden rule: do unto others as you would have them do unto you. The *consequences* approach says one should help a person in need because it makes the world a better place. In many cases all three approaches agree. Things get interesting when the different approaches do not agree.

- Virtue

  TODO

- Rules

  TODO

- Consequences

  TODO

5 Societal

TODO

6 Environmental

TODO

6.5 Results

As a rule, software systems do not work well until they have been used, and have failed repeatedly, in real applications. — *David Parnas*

Results may come in many forms, even for a single project. This section discusses some criteria and evaluates some projects within each. The following subsections describe different perspectives from which projects can be evaluated. There might be other perspectives that are reasonable but not included in this document (yet). The students will indicate from which perspectives they wish their project to be evaluated.

Note that the grades we assess here are not the grades these projects received historically. In the past capstone projects were assessed on different criteria than we discuss here — the criteria have evolved over time. Teams from 2014 are more likely to score highly than teams from 2012 here because the criteria used in 2014 were closer to those presented here than what was used in 2012.

Some projects might be evaluated under multiple criteria. In that case, the referees are asked to evaluate under each criteria according to the rubrics. It is the instructor's job to merge these into a final grade. It is recommended that in the case of a high/low split (*e.g.*, *A* on one criteria and *C* on another) that the instructor take the higher value and drop the lower one. In the case of two similar evaluations the instructor might consider merging them to a higher value — although the exact nature of this merge will be left to the instructor's discretion, and might include consideration of the class as a whole.

Note that it is already the job of the instructor to merge conflicting reports from different referees on the same criteria. That is a different matter than merging reports across different criteria, which is what we discuss here.

The distinction between *A* and *A*+ in these rubrics often comes down to factors in the external world. The referee is encouraged to use their own professional judgement, and is invited to deviate from these rubrics where it makes sense to do so. Each project is unique and faces a unique set of external circumstances.

6.5.1 New Product

Take CS449 to learn how to do user studies and user-centred design.

A user study might require clearance from the UW Office of Research Ethics:
https://www.youtube.com/watch?v=9lE1cNIYayo&list=UUuGzdVC5BHHcONvGGcaXFcw
https://uwaterloo.ca/research/office-research-ethics/research-human-participants/application-process

Some projects aim to create software to be used by the general public. In this case, the software is run by individuals for their own private purposes, rather than by an organization for its collective purpose. The main criteria here is a function of the number of users and the amount of engagement each user has with the software; user studies are an alternative criteria.

These projects are often (but not always) C.R.U.D.

C.R.U.D. is an industry term from the 1980's that stands for Create, Read, Update, Delete.

Grade	*Criteria*
A+	Thousands of light users *or* hundreds of heavy users *or* positive mention in mainstream/industry press *or* winning a reputable startup pitch competition.
A	Hundreds of users you don't know *or* rigorous user study.
B	Dozens of users you don't know *or* user study.
C	Friends have tried your software.
F	No users. No user testing.

3.6in

The idea of this table is, roughly, the amount of time people have spent using your software. So, dozens of people for dozens of hours each would score similarly to thousands of people for several minutes each.

Alternatively, user studies, mentions in the press, and startup competitions are to be rewarded here.

TEAM RED COCONUT (2014) created the UWFlow.com website for

Grade: A+.

students to critique courses and share their schedules. As of Symposium Day they reported 4700 users, who had collectively performed 480,000 searches. They achieved this level of success by getting the software working before 4A and having an active marketing plan.

Team Hivemind (2013) created a peer-to-peer framework for implementing MMORPGs (Massively Multiplayer Online Role Playing Games). Nobody was using their framework to develop games. Nobody was playing games developed with their framework. They had not even implemented a variety of games to demontrate the flexibility of their framework (or at least they did not say so in their talk).

Grade: C. This project won first prize in 2013. It was a great project, and they did a great job. Results were not part of the evaluation criteria before 2014. The Hivemind group would surely have shifted their efforts in this direction — and succeeded — had they been presented with the modern criteria.

Prof Jacques Carrette, Director of Game Programming at McMaster University, argues that any game framework project should implement at least half a dozen interstingly different games to demonstrate the flexibility and value of the framework.

New products might need a marketing plan in order to grow the user base (if number of users is chosen as the metric of evaluation). UW Flow (SE2014), for example, collaborated with students in Accounting, as well as various campus orientation programs, in order to publicize their website to students on campus. Figure 6.1 shows part of the marketing plan for a new food ordering app in Toronto (this app was not developed by SE students).

The recommended time to activate marketing efforts is the co-op work term in between 4A and 4B: at this point the software should be ready for public use (after SE490), and there is still time to gather results heading towards Symposium Day (SE491).

Figure 6.1: Marketing plan for Toronto-based food ordering app, Fufu Rocks:
http://www.fufu.rocks/
Picture taken in downtown Toronto (summer 2015) and submitted by a Nano'15 grad. The monkey is also distributing free stickers to remind people to visit the website.

It is not clear from their website if the name 'Fufu' is just for fun or is supposed to be a reference to the West African dish of that name. Fufu does in fact rock, but it is not clear if they actually deliver fufu. One good source of fufu in Toronto is the Tree Top Cafe on Dundas, just west of Spadina.

Good sources of rental costumes are Queen of Hearts in Kitchener and Malabar in Toronto.

6.5.2 Research

Some teams choose to collaborate with research groups, usually on campus, on active research problems. The goal here is to advance knowledge. An indicator of that is publication in a competitive, peer-reviewed venue (*e.g.*, conference or journal).

A paper accepted in a competitive, peer-reviewed international venue earns an *A*+, even if it is a short paper that describes applying a known technique in a new context. Having a paper accepted gives some validation for the work beyond the UW context. There are a variety of logistical reasons why a paper might not yet have been accepted by Symposium Day. Here we rely on the Symposium Day referees to assess the quality of the work.

The referee is to exercise their professional judgement in assessing the novelty of the work and the validation presented for it. This table is a rough guideline.

Grade	*Criteria*
A+	Novel results or context *or* paper published, *etc.*
A	Prototype works on a wide range of reasonable inputs and some challenging ones.
B	Prototype works on reasonable inputs.
C	Prototype works on trivial inputs.
F	Prototype is vapourware.

Team Amalgam (2014) worked to parallelize an algorithm for exact, discrete, multi-objective optimization. Along the way they also developed some improvements for the computation using formula rewriting and incremental SATisfiability solving. They had a research paper accepted in a competitive, peer-reviewed computer science conference (ABZ'14) by Symposium Day.

Grade: A+. The ABZ'14 conference accepted 34 papers out of 81 submissions, including one from Team Amalgam. In other words, there were 47 papers submitted by international research groups that were rejected and hence considered to be of lesser value than Amalgam's.

Team Radiant (2014) developed a novel technique for doing R-peak detection on ECG (electro-cardiogram) data. They published this work several months after Symposium Day. The difference between the published paper and their Symposium Day result was the thoroughness of their literature review to establish novelty.

Grade: A. This could have been *A*+ on Symposium Day if they had made a more thorough argument about the novelty of their work, or if they had a referee who knew the area and could vouch for the novelty.

Team Satisfaction (2014) developed a parallel boolean SATisfiability solver, in which the main thread runs a standard DPLL/CDLC solving algorithm, while some side-threads run various formula simplifications.

Grade: A. Also see Risk Reward section below.

Team Spike (2014) parallelized the Nengo brain simulation engine, in collaboration with researchers Terry Stewart and Chris Eliasmith at the UW Centre for Theoretical Neuroscience.

Grade: A-

Team SWan (2013) developed some visualizations for multi-objective optimization. They understood the problem and devised a prototype, but they had no experiments: they had not run their visualizer on non-trivial data; they had not done any sort of user study; they had not made their tool available for others to try.

Grade: C

6.5.3 *Custom Software*

Some projects write custom software for a specific customer, usually an organization, for use in their operations. The following table gives a general guideline for grades in this area. Referees may consider other factors that they consider to be important even if those factors are not included in this table.

Grade	*Criteria*
A+	System is in production and is public-facing or part of critical operations.
A	Customer is actively working to integrate system into production, and system is public-facing or part of critical operations.
B	Customer feedback on an earlier prototype; concerns have been addressed in newer version.
C	Customer feedback on an earlier prototype.
F	System diverges significantly from customer requirements. Customer does not intend to use the system. Team has stopped speaking to customer.

A formal letter from the customer describing their integration plans and activities would be the ideal form of evidence.

TEAMS WATPARK (2012) AND NOMANAZONE (2014) created and enhanced, respectively, the system used by UW Parking Services to report which lots are full. UW Parking Services not only uses this system internally, but also makes it available to the general public.

Grade: A+

TEAM SWT GROUP (2013) worked on the Facebook application MyTopFans, which has millions of users worldwide. They revised the architecture, improved the user experience, created a mobile version, and implemented new algorithms based on social science research.

Grade: A+. MyTopFans already had millions of users when this design project started, so it is not evaluated under the *Users* category here. On Symposium Day 2013 SWT Group had some friction with referees who found the idea of MyTopFans morally distasteful (one view is that it profits from the natural insecurities of teenagers).

TEAM ÉTALLONAGE (2014) worked with JGR Optics (Ottawa) to redesign and reimplement their software for calibrating fibre optic equipment. This software is used in-house by JGR Optics to tune fibre optic equipment that they then sell. Team Étallonage reduced the lines of code in the software by an order of magnitude, from 56k to just 5.5k, while also significantly improving the user interface and the extensibility of the software.

Grade: A. While Étallonage's redesign significantly improved the software, and JGR Optics was planning to move it to production, it was not actually in production as of Symposium Day.

TEAM RADIANT (2014) worked with Life Care Networks (China), who manufacture portable ECG recorders. Team Radiant developed a new algorithm for R-peak detection, as well as automating clinical decision trees on ECG analysis. They reported that Life Care Networks was working to integrate their code as of Symposium Day.

Grade: A. Their presentation does not give much evidence as to the state of the integration. The presentation focuses on the research contribution (see above).

TEAM EVENTDEX (2012) created a summer camp management sys-

Grade: F. These points were not fully discovered until after the term had ended and the Instructor spoke with the customer. Now we expect you to present more evidence of results on Symposium Day.

tem for Engineering Science Quest. This system never went into production, and in fact significantly diverged from the customer's requirements. Moreover, they had not actually spoken to the customer for over six months.

TEAM SS-NET (ECE 2014) wrote a matching application for the UW Student Success Office's International Peer Mentorship Programme. The Student Success Office matches incoming international students with peer mentors from a group of volunteer students.

Grade: A. As of Symposium Day the Student Success Office was actively working with IST to put this system into production.

6.5.4 Free/Open-Source Software

Some teams choose to contribute to existing free/open-source software systems. We encourage such projects.

Grade	*Criteria*
A+	Patches accepted, positive mentions in press/release.
A	Patches accepted.
A-	Patches accepted but then reverted due to bug/issue.
B	Patches submitted and reviewed.
B-	Patches submitted.
C	Patch appears to work on student computers.
F	Patch is vapour.

As with all categories, we rely on referee judgement: referees are not strictly bound by these tables of guidelines. A sentence explaining departure from the guidelines is appreciated. In particular, in this context, the technical challenge of the enhancements attempted is an important factor.

TEAM CRYPTKEEPER (2014) added support for Galois/Counter Mode to the eCryptfs Linux cryptographic file system. This enhancement allows the file system to detect when files have been tampered with and warn the user. Tampering could occur by malicious actors, disk corruption, or otherwise. As of Symposium Day they had submitted their patches to the Linux kernel mailing list twice. The first time the patch was reviewed and rejected for a security flaw, and the revised second submission was pending review on Symposium Day.

Grade: B. Michael Chang, member of Team CryptKeeper, recommends submitting small focused patches starting in 4A, rather than large patches in 4B.

http://new.livestream.com/itmsstudio/events/2850051 first video, first talk

TEAM JUNTAO (2013) worked on the Sana.MIT.edu project. Here is their abstract:

> Sana Mobile is an open source project developed by MIT. Our project is to make an open source contribution to the existing code base, create a data visualization portion for a Sana mobile client, an Android mobile for collecting patient data in remote locations for the doctors in headquarters. This project should be able to produce a visual representation of selected portions of an individual patient's medical history in a meaningful fashion. Because of the nature of this app, we must resolve a wide range problems such as, data synchronization after the device has been without any internet connection, data caching for displaying most recent available data in remote areas without any connection, and designing how to display retrieved data.

Grade: A
The Sana project is interested in more collaboration with Waterloo. There is an SE2016 team working on a Sana project, and some SE2017 students are taking the Sana course at MIT remotely. We encourage you to get involved with this great project. Contact Derek Rayside.

6.6 Technical Evaluation

See Grading Sheet and Status Sheet.

6.7 Communication Evaluation

Symposium Day communication is graded holistically. That is, the grade is not broken down in terms of the communication media used: poster, demo, presentation. Historically different referees graded each artifact, so grading was structured that way. No the same panel of referees will observe all media, and so can get a deeper understanding of the project and of the team's ability to communicate it.

Some SE students have reported that the Systems Design students make excellent posters.

6.8 Demo + Poster (at Booth)

The referees will come by your booth, probably before your talk, to see your demo and poster. This initial meeting is to give them a basic idea of the outline of your project and your software's functionality.

It is recommended that you have demo scripts of different lengths, *e.g.*: 60s, 3 minutes, 5 minutes, as different people will want to spend different amounts of time. The referees are likely to want the longer demo, but members of the public might want to start with something shorter. You need to gauge your audience.

THE POSTER is intended to function both as a standalone artifact and as visual support for the demo. There are many example posters from previous years mounted for viewing in the SE lounge and labs. You are encouraged to examine them critically and develop your own judgements.

The poster mounting rails were paid for by sponsorship funds from Yelp.

There will be background template files provided with UW branding. There might also be sponsor logo files provided for you to add to the template. These files are typically PDFs.

6.9 Presentation

The presentation is the main chance for you to communicate with your referees. This is where you will talk about the design and architecture of your software, including interesting alternatives; interesting aspects of the requirements; your productivity; and your results. In short, the items described on the status sheet.

See §7 for discussion on communication in general and how to make a presentation in particular. You will want to be careful about your selection of narrative structure for your presentation.

6.10 Risk Reward (Bonus)

You are encouraged to take sensible risks, and will be rewarded for doing so in one of two ways. The best case scenario is that your risk pans out and the referees observe your success on Symposium Day. The other scenario is that your risk does not pan out, and the success you have to report on Symposium Day is not as great as you were hoping. In this latter case the referees might give you a lower mark than you were hoping for. In this case, you may petition the Instructor, in writing, before Symposium Day, explaining the risk you took and why it didn't pan out. The Instructor, in their sole discretion, can raise the grades awarded by the referees to reward you for taking the risk.

In order to be eligible for the Risk Reward you must have pivoted to a project on which you have had some success. You cannot show up on Symposium Day completely empty handed.

You may, if you choose, also discuss the risks you took (that didn't pan out) in your presentation. In that case, you are asking the referees to vouch for your Risk Reward petition with the instructor, not to adjust their interpretation of the rubrics.

Difficulty of the problem to be solved can be a basis for a Risk Reward petition. Here is a scale of difficulty:

a. Clay Mathematics Institute Millenium Prize Problem.
b. Other researchers/companies have tried and failed.
c. Requires inventing a new idea in a new area.
d. Applying an old idea in a new area.
e. Using an old idea in a known way.

You should probably not attempt Millenium Prize Problems during your capstone project — save those for grad school. You might consider attempting problems that other researches have failed at if you have good collaborators and a reason to think that your approach is different than what was done in the past.

If you are using an old idea in a known way then it will be challenging to submit a successful Risk/Reward petition on the basis of the problem difficulty.

Team Satisfaction (2014) started out attempting to accelerate a boolean satisfiability solver by using a GPU. They knew at the outset that this was a hard problem that nobody had ever succeeded at before (despite a number of attempts by research groups at other universities). This problem was, as expected, too hard. They pivoted to parallelizing a boolean satisfiability solver by running a secondary thread to do formula simplifications.

Team JSTD (2013) started out with a project to connect independent musicians with fans (Tunezy.com). By the time SE491 came around they were talking to investors. They were afraid that the investors would be scared off if they presented Tunezy at Symposium Day (perhaps an unfounded fear). So they created another project, SchoolAx, in January, and presented that at Symposium Day. SchoolAx was a website to help students find events on campus (and advertisers to find students). SchoolAx had very limited results — but it is a great example of what can be accomplished in two months. Team JSTD had their SchoolAx grade bumped because of Tunezy.

TEAM SEASALT (2013) started out in SE390 working on a flight pricing project. They invested hundreds of hours into this project before realizing that they would never really be able to get the raw data necessary to make the project succeed. So they switched projects. You can read their Risk Reward petition in the docs directory where this handbook is stored.

In light of Seasalt's experience, we now expect you to do more careful due diligence on data acquisition in SE390 before you invest hundreds of hours in a project you won't be able to complete.

Communicating Complex Projects

Poetry is more perfect than history, because poetry gives us the universal in the guise of the singular, whereas history is merely the narration singular events.
– Aristotle, *Poetics*

For most of your academic lives you have been concerned with producing enough content to meet the course requirements. Now you have the opposite problem: too much content and not enough time or space to communicate it in. This is the situation that you will face for the rest of your professional career. Communicating effectively in this context requires effective abstraction mechanisms.

Effective programming is, to a large extent, about employing appropriate abstraction mechanisms. The difference between imperative, functional, and object-oriented languages — versus each other and versus assembly — is the abstraction mechanisms afforded to the programmer. As a graduate of the UW Software Engineering programme you have excellent ability to use appropriate abstraction mechanisms when communicating your programs to machines.

Now you will develop your abilities to select and apply appropriate abstraction mechanisms when communicating your complex project to other people.

7.1 Old Stories Told in New Ways

An old saying amongst writers is that *there are no new stories — only old stories told in new ways.* Indeed, many writers have writen reflective books about this adage: *e.g.*, Northrop Frye's *Anatomy of Criticism*, Joseph Campbell's *Myths to Live By*, Robert McKee's *Story*, John Yorke's *Into the Woods: A Five-Act Journey Into Story*, Christopher Booker's *The Seven Basic Plots*, and many others. Consider, for example, the following story:[1]

> A dangerous monster threatens a community. One man takes it on himself to kill the beast and restore happiness to the kingdom ...

This is *Beowulf*, the oldest surviving epic poem in Old English (roughly 1000 years old). But it is also the 1975 movie *Jaws*, or *Godzilla*, or *Jurassic Park*. In other variants of this story, the monster might take another form. For example, in *Erin Brockovich* the monster is a corporation; in *The Towering Inferno* it is fire; in *The Posiedon Adventure* it is an upturned boat. This is every episode of the medical drama TV series *House*, where the monster takes the form of some disease.

[1] This example taken from John Yorke's 2016 article in *The Atlantic* titled *All Stories are The Same*: http://www.theatlantic.com/entertainment/archive/2016/01/into-the-woods-excerpt/421566/

In retelling an old story in a new way, some details are changed. The form of the monster; the setting of the kingdom; the gender and nature of the protagonist, *etc.* Retelling is, in software engineering terms, a process of abstraction, transformation, and concretization.

Who's on First?, as performed by Abbott & Costello, was named by Time Magazine as the best comedy sketch of the twentieth century. The gag is that the name of the first baseman is 'Who'. But they didn't 'invent' it: they perfected it. The baseball version of the sketch was already on the comedy circuit at the time, and was based on other previous variants such as *Who's the Boss?*. This gag is perhaps one of the oldest jokes in Western literature: Odysseus uses it with the Cyclops in Homer's *Odyssey*, which is the second oldest extant work of literature in the West (preceded only by its prequel, Homer's *Iliad*). After Odysseus blinds the Cyclops in his sleep, the Cyclops asks 'who did this to me?', to which Odysseus replies, 'my name is Nobody.' When the Cyclops' brothers come to help him, they ask him who blinded him and he says 'Nobody,' so they leave without helping him, thinking that he has also lost his mind. By the time Abbott & Costello got a hold of this joke it was almost three thousand years old, yet they made it the best of the twentieth century.

https://en.wikipedia.org/wiki/Who's_on_First?

https://en.wikipedia.org/wiki/Odyssey

EXERCISE 1: Find two (or more) past projects that told essentially the same story. The greater the surface differences between the teams the more interesting your result.

EXERCISE 2: Find four (or more) past projects that told stories that your team might retell. These stories will be of two kinds: *comedies* and *tragedies*.

This distinction goes back to Aristotle's *Poetics*.

A *comedy* tells of the rise in fortune of a sympathetic central character. A story with a happy ending (not necessarily a funny story). Look in the Awards chapter (§11) for comedies.

A *tragedy* depicts the downfall of a basically good person through some fatal error or misjudgment, producing suffering and insight on the part of the protagonist and arousing pity and fear on the part of the audience. In the capstone context, this corresponds to changing projects (§5.12) and changing teams (§5.13).

It is important to note that these are not mutually exclusive categories: a team may both change projects and win an award. For example, Team Parallax (§5.12.6, §11.16.6), and Team Kaze (§5.12.7, §11.16.2) both changed projects and won awards in SE2016. Agile development and startup culture both promote constructive pivoting: *embrace change* is the subtitle of the original *eXtreme Programming* book by Kent Beck.

For this exercise you should find two stories of teams who changed projects (tragedies) and two stories of teams who won awards (comedies) and re-tell them in the context of your team.

7.1.1 *Some Generic Old Stories*

There are some stories that re-occur more often than others. Some of these are listed in the chapter on *Popular Projects* (§13) that we have compiled in collaboration with Velocity. Some of the recurring patterns are:

Saving third-year undergraduates a few dollars. Many projects are motivated by trying to save third-year undergraduate students a few dollars. These projects are often misguided, because the main strategy to reduce costs is to donate free engineering labour (yours). By the time you graduate and have a job, you will likely lose motivation for this kind of project, because you will have a better understanding of why things cost money and will place greater value on your time.

Moreover, another purported cost-savings strategy that these projects employ is to abuse the terms of service of an existing business, which is clearly unethical.

If you think your project is re-telling this story, you should probably think about pivoting.

This strategy is also a violation of the peo Code of Ethics: 77.7.v states that practitioners shall 'uphold the principle of adequate compensation for engineering work'. You should not justify a business case on the basis of free engineering labour. Note that this clause does not prevent you from doing free volunteer work for a charitable cause, or from having an interesting hobby. The problem is when the motivation for the project is commercial and the competitive advantage comes from undervaluing engineering work.

Building a more integrated solution. Some capstone project proposals are motivated to build a more integrated solution in some domain where there already exists good solutions to certain aspects of the problem. Be wary of this motivation for two reasons:

(1) Does your plan involve first re-building all of the good existing components? That might be time consuming. You might end up building components that are worse than the existing ones. Would users in this domain prefer better integration of worse components, or looser integration of better components?

(2) Why haven't the developers of the existing solutions done this integration already? Did they not think of it? Probably they thought of the idea. There is probably some other practical reason why this integration hasn't happened yet. Are you somehow in a better position than they are to do the integration?

7.2 *Narrative Structure*

Narrative structure is an important abstraction mechanism that will help focus the audiences attention on the most interesting aspects of your complex project. Some options include:

Historical. Describe what you did on your project chronologically, from the beginning to the end. This is a popular choice for practice talks, but is often not the best choice for the final presentation.

The historical structure might involve suspense: will the project succeed? Will it produce results? Wait to the end to find out! The tension might escalate as the plot progresses to the climax and finally resolution.

An example where this historical structure worked very well was the TSK group from 2015. They won third prize for a series of user studies on a virtual keyboard design for the Oculus Rift. The historical structure worked well for them because the audience joined their voyage of design and discovery.

A variation on the historical structure is espoused by the creators of *South Park*:[2] if the connectives between the events in your story are 'and then', it's boring; instead, try to arrange things so that you can use connectives like 'but' and 'therefore'. These connectives create greater causal (rather than merely temporal) relationship between the events, which drives the story forward with greater purpose.

User-centered. You could alternatively tell a user-centered story, rather than a story about your group's journey through the project.

Freytag's Triangle describes the standard structure of a linear story.

Scientific. A scientific presentation starts with the claim and then presents the evidence. For example, a paper in the journal *Nature* might be titled *The Lkb1 metabolic sensor maintains haematopoietic stem cell survival*.[3] This title tells us the claim: some specific gene (Lkb1) helps blood (haematopoietic) stem cells survive. The paper will tell us the evidence. In mathematics we would expect the title to state the theorem and the paper to provide the proof.

There is no suspense in this scientific structure: the audience knows from the outset what the conclusion is.

Grad student Jon Eyolfson has kindly provided us with both a draft version of his MSR'11 talk, following a story structure, and the final version using a scientific structure.

How to give a killer presentation: Lessons from TED[4] focuses on the choice of narrative structure. The main article advocates a story narrative; the sidebar describes a spectrum of structures from story to scientific. Read the article and watch a few TED talks.

[2] http://nathanbweller.com/creators-of-south-park-storytelling-advice-but-therefore-
Reference from Josh Kergen SE2018.

[3] *Nature* 468:659–663, 2 Dec 2010

[4] *Harvard Business Review*, June 2013.

7.3 *Presentations*

This section reviews some aspects of making presentations that you might not have learned about elsewhere.

This section includes some links to external documents that are in this Handbook repo. You will have local copies of these files if you have checked out the repo. If you just got a copy of this Handbook off the web then the external file links will not work.

7.3.1 *Visual & Logical Structure*

Try to make the visual structure on your slides correspond to the logical structure of the content. For example, if discussing *before* and *after*, considering a left-to-right layout rather than bullets:

Change Happens

- Before
  - night
- After
  - day

vs.

Change Happens

Before	→	*After*
night		day

7.3.2 *Use Images Where Possible*

Use images where possible. You might be surprised at the range of possibilities for using images, even in technical talks. Two examples of talks that made good use of images to explain technical software engineering research are:

Kodkod a relational constraint solver, by Emina Torlak.

Object Ownership Profiling a technique for finding and fixing memory leaks, by Derek Rayside.

7.3.3 *Body Language*

People hear body language first, tone of voice second, and the semantic content of speech third. Practice your delivery with a (friendly) critical audience. Videotape yourself.

The supporting speaker communicates only through body language while the primary speaker is talking. Watch a newscast and observe what the supporting speaker does: they make eye contact with the audience, and direct the audience's gaze to the speaker; they nod to affirm the speaker's message; they use their face and body to show that they are actively listening to the speaker.

7.4 *Writing*

Writing good technical prose is a process.

There should be at least a day or two in between steps for your mind to refresh.

a. *Brain Dump:* Write down everything in your mind. It will come out a bit messy. Don't worry about that now. Keep writing and exploring the ideas. New thoughts will come to mind as you write.

b. *Abstraction:* Identify the key concepts and ideas. Decide on consistent names for them. Rewrite the introduction and conclusion using this terminology.

c. *Breakdown:* Using a whiteboard or a piece of paper, or mind-mapping software, make point-form notes of everything in the brain dump. Re-arrange and organize these notes.

d. *Target:* Who is your audience? Business? Software? Other engineering? How can you explain your project in terms of things they already understand? What are the three most important things you want them to know?

Thanks to Matt Magni ECE 2018 for good suggestions on this process.

e. *Rebuild:* Start with a blank slate. Re-write all new prose from the point-form notes — including the introduction and conclusion. Do not look at the old text from the brain dump. Some points might not fit into the new flow: stick them in an appendix for now.

f. *Make it Concrete:*

- Use examples wherever possible.
- Rewrite explanations in the reader's terms, rather than the technology's internal terminology. In this example, *strict ordering* is internal terminology:

  Before: ↦

  The debugger generates the following discriminating example, where at state S_2, process P_0 holds two mutexes $\{M_0, M_2\}$, but at the next state S_3, the process holds three mutexes $\{M_0, M_2, M_1\}$, but they are not in strict order. This uncovers the underconstraint issue, because the strict ordering is violated.

  After:

  The debugger generates the following discriminating example, where at state S_2, process P_0 holds two mutexes $\{M_0, M_2\}$. In the next state, S_3, process P_0 additionally acquires mutex M_1. The debugger is testing the hypothesis that M_1 can be added to a set that already contains M_2 .

- Walk the reader through any inferences or reasoning that the text might require. In the following text, the reasoning that the reader would have to do examining the figure is explicitly described in the revision:

  Before: ↦

  The engineer rejects the discriminating example because a process P_1 can take an unintended lower-indexed mutex.

  After:

  The engineer rejects this discriminating example because in state S_4 process P_1 takes mutex M_2 when it already has mutex M_3 , thereby violating the intention that mutexes are acquired in order.

g. *Remix the Extras:* Add back any important points that had be relegated to the appendix previously.

h. *Spelling, Grammar & Consistency:* Polish.

7.5 *Writing A Project Abstract*

Your abstract is an important introduction to, and summary of, your project. It should be in the following form:

This material derived from Prof Dan Davison's notes for ECE498.

The UW Writing Centre can help you revise your abstract. They have drop-in hours in the DC library.
https://uwaterloo.ca/writing-centre/

Your abstract will evolve over the course of the project.

- about 200 words (not more)
- pure text (no figures, tables, formulas, *etc.*; minor LaTeX permitted)
- primarily present tense
- limited jargon and acronyms
- good grammar and spelling

Your abstract should cover the following five issues:

The *five W questions* can help you see the big picture: who, what, where, when, why.

Context & Motivation. What is the problem that your software will solve? Who will use it? Be as clear, specific, and factual as possible. Some examples:

- "Each year, about 850,000 vehicles in North America are involved in a collision due to failure to check one's blind spot."

  ECE Group 2009.054

- "In the US and Europe, 300,000 people die every year from cardiac arrest before they can reach medical care."

  ECE Group 2012.035

Objective. What does your project aspire to accomplish within the given context? The objective must be open-ended enough to allow for multiple solutions to be considered in the design process.

Being open-ended is what makes this an *engineering design* project.

Plausible Design Approach. While the problem and objective must be rich enough to admit multiple possible solutions, you should have one in mind to start with. This will likely change as your project progresses, and might be completely different by the end.

You should identify key design challenges and the advanced technical knowledge required to meet them.

Do not mention course numbers here though. That's on the Status Sheet §1.4.

Expected Benefits over Existing Alternatives. Clearly state at least one advantage of your (proposed) design over existing alternatives. Be specific. For example:

- "The main advantage of this design over major alternatives is that physical size, strength, and/or mobility are not required to control the wheelchair."

  ECE Group 2011.043

- "The Autotabber package can be used with existing guitar systems, and can detect different playing techniques on a per-string basis, which differentiates this product from pre-existing solutions."

  ECE Group 2012.031

Summary of (Expected) Results. What evidence have you gathered (will you gather) to demonstrate that the objective has been accomplished? See §6.5 for common approaches and targets.

This will evolve as your project progresses. Perhaps it will be written in future tense in SE490 and past tense for Symposium Day.

7.5.1 *Example Revised Abstract from Team Radiant* SE2014

Original Abstract:

Our customer is a healthcare company in China. The company manufactures portable ECG recorders, and offers service to send captured ECG graphs to partnering hospitals for diagnosis. This is especially useful for elderly who may not wish to travel to hospitals regularly for checkups.

As a part of our project, we are working on extracting important features in the graphs, and researching on automatic ECG classification. Features such as the locations of prominent peaks help to improve visualization for the doctors; while having automatic classification will no doubt reduce the company's operational cost.

Revised abstract:

Electrocardiograms (ECGs) are usually recorded in a clinical setting by medical professionals using twelve leads attached to the patient. Our industry partner has developed a single-lead ECG machine for use by patients at home. Patients can then send these readings to remote doctors. The goal of the machines is to make medical expertise more accessible, affordable, and convenient.

The ECGs recorded by patients with a single-lead suffer greatly from baseline wandering and high frequency noises, as compared to ECGs recorded with twelve-leads in a clinical setting.

Accurate R-peak detection is an important step in ECG analysis. A variety of methods have been proposed in the past against standard clinical twelve-lead ECG recordings. In this study, we propose a new R-peak detection algorithm for single-lead mobile ECG recordings. Our area-based approach is built on the understanding that QRS complexes are typically narrow and tall, resulting in large areas over the curve around these locations. Our algorithm is simple to implement, computationally efficient, and does not require any signal pre-processing.

We evaluated our algorithm against data collected by patients from single-lead portable devices, and yielded 99.3% precision and 99.4% recall. The MIT/BIT Arrhythmia Database of twelve-lead clinical ECG recordings was also used to verify our algorithm. On this dataset we obtained a precision of 99.3% and recall of 98.6%.

Additional Context:

- *who:* used by patient
- *what:* 12-leads *vs.* 1-lead
- *where:* clinical *vs.* home use
- *why:* patient convenience & cost

Additional Information:

- *intuition:* shape of QRS complex
- *challenges:* baseline wandering and high frequency noise
- *advantages:* easy to implement + runs fast
- *validation:* experimental results

Intellectual Property & Collaborators

Waterloo Policy 73 says, roughly, that the creator owns their own intellectual property. By contrast, at most universities the university owns your intellectual property — just like most employers choose to do. This policy is intended to promote entrepreneurship: it is much easier for you to start a company if you own your intellectual property.

https://uwaterloo.ca/secretariat-general-counsel/policies-procedures-guidelines/policy-73-intellectual-property-rights

Some projects have an external collaborator (or 'customer'). We actively encourage external collaborations because they help clarify requirements and push you to do your best. Part of the groundwork of establishing these collaborations is communicating clearly about intellectual property.

8.1 What is Intellectual Property?

The academic concept of intellectual property is broader than the legal concepts. In academia we consider ideas to be intellectual property that must be properly credited, whereas ideas *per se* have no protection in law. The law recognizes four kinds of intellectual property: patent, copyright, trademark, and trade secret.

http://www.gnu.org/philosophy/not-ipr.html

Patent: A patent grants a time-limited commercial monopoly in exchange for public disclosure of an invention. An invention must have practical application. Once the time limit expires, anyone can use the invention. The time limit of a patent is around 20 years in most countries.

The Content Scramble System (CSS) used to encrypt DVDs is protected by patent, for example. Any software (or hardware) that implements this system without a license is in violation of the patent — regardless of whether that software was written from scratch.

Copyright: Copyright grants a time-limited distribution monopoly on some written or recorded work. The time limit on copyright in some countries is life of the author plus fifty years. Copyright is the branch of law that protects source code, and is the basis for all free and open source software licenses.

Consider the Linux kernel source code. It has many contributors. Each contributor still owns the copyright on their individual contributions. They have chosen to release those contributions under the GPLv2, which grants others the freedoms to modify and redistribute that source code. The Linux kernel will never move to GPLv3, nor to any other license, because to do so would require having every contributor sign off on the license switch, which is logistically impossible. Only the copyright owner can change the license, and copyright is the body of law that makes free and open source licences legally possible.

Project Gutenberg redistributes classic works of literature and thought on which the copyright has expired. For example, Shakespeare, Dickens, Darwin, Adam Smith, *etc.*

Trademark: The names of companies or products may be trademarked to protect their use in the market. For example, 'Kleenex' is a trade-marked name for a specific brand of tissues.

Trade Secret: Perhaps the most famous example of a trade secret is the recipe for Coca Cola. Patent, copyright, and trademark all involve public disclosure. If someone were to steal the recipe for Coca Cola and attempt to sell it, the Coca Cola company could sue them under trade secret law.

Consider the case of Einstein's work on the theory of relativity. His publications on this topic would be protected by copyright.

It is likely that Einstein transferred his ownership of that copyright to the journal the published the papers. This is a common, but increasingly controversial, aspect of scientific publication. The publisher wants to hold the copyright so that they have exclusive right to distribute the papers, since such distribution is their business. Some scientists argue that taxpayer funded research should be freely redistributable; other scientists go farther and argue that all knowledge should be freely redistributable.

But copyright does not protect the idea of the theory of relativity, just the text that he wrote for the papers. Someone else could write another paper claiming to have discovered the theory of relativity, and as long as they used different words, they would not have committed a violation of copyright law. This would be a gross academic transgression, because in academia we consider that Einstein 'owns' the idea of the theory of relativity, but that concept of ownership has no basis in law, and the offender could not be punished with incarceration or government sanctioned fines. The academic community would shun the offender, but this is the collective action of a community based on their social norms, and not the act of a central authority empowered by law.

Einstein could not patent the theory of relativity because the theory itself does not describe any practical applications. If someone made an invention that used the theory of relativity to do something practical then they could patent that invention — and they would own the patent, legally, not Einstein.

8.2 *Collaborating on Free / Open Source Software*

Collaborating on free or open source software is usually uncontentious: the students retain ownership of the copyright on the source code, and simply license it to the collaborator using some established free/open source license.

8.3 *Collaborating with an Established Corporation*

Collaborating with an established corporation is usually easy. There are two common cases.

Large corporations typically collaborate with students for the purpose of meeting potential hiring candidates. These companies have no intention of using the students' intellectual property. For many companies the legal complexity and potential liabilities of using intellectual property developed by outsiders are prohibitive.

Smaller corporations might actually be interested in using the software produced by the students. For example, some groups have used their capstone projects to write software for former co-op employers. In these cases the collaborator would typically pay the students for their work. This is not common, but there are no university rules prohibiting it. The project must be up to academic standards, and the instructor (or judges in SE491) must be able to know enough about the project in order to evaluate it.

8.4 Collaborating with a Startup

Velocity has good resources online: http://wiki.velocity.uwaterloo.ca/Legal

The intellectual property discussion is typically the most complex when collaborating with someone who has a business idea for a startup. Essentially the students are becoming co-founders with the collaborator.

A simple approach, which is not generally recommended by the university, is that the students just give their work to the collaborator. The rest of this section discusses our recommended approach of co-ownership.

8.4.1 Understanding Your Startup Collaborator

First, it is important to recognize that everyone has something at risk and something to gain. Students understand their own situation, but do not always clearly understand the position of the collaborator. The collaborator bears the greatest initial risk because they are disclosing their business idea to the students. They fear that the students will take that idea and become competitors, or that the students will disclose the ideas to others who will become competitors. They also fear the opportunity cost of investing time with the students if things don't pan out.

8.4.2 Non-Compete and Non-Disclosure Agreement

We have found that one approach to mitigating the collaborator's initial business risk concerns is to have a non-compete and non-disclosure agreement along the following lines. We have found that email agreements with clauses such as the following are sufficient to make everyone comfortable and move the conversation forward.

a. I agree to treat information from *collaborator* as confidential, and to not disclose it beyond what is necessary for my academic requirements.

Non-disclosure, to pacify collaborator

b. I agree to not pursue a project that is related to, or competes with, *collaborator's* project while I am a student at the University of Waterloo, unless that project is in collaboration with *collaborator*.

Non-compete, to pacify collaborator

c. I understand that *course instructor* will help our group find a suitable alternative project and guide us through the capstone design project process if an alternative becomes necessary.

Instructor help, to pacify students

This is an initial agreement that would be made in the first month of SE390 so that the discussion can move forward into the second month. By the third month if everyone wants to move forward then a more sophisticated agreement is probably necessary.

8.4.3 Co-Founding a Startup with a Collaborator

This is an approach that has been used successfully by a number of startups connected with UW. Thanks to Asif Khan (Nano'12) for sharing it with us.

You and your collaborator are co-founding a company. That company will own all of the intellectual property, sales contracts, *etc.*, that you and your collaborator create. But you won't formally incorporate this company until some date in the future. For now we'll consider this date to be at graduation, but it could be sooner.

When the company is incorporated, you will have to decide how much of it each person owns. Don't try to figure this out now: that approach won't work, and you'll spend endless hours arguing about different hypothetical visions of the future. Instead, what you want to agree on now are the mechanisms and criteria by which you will evaluate this in the future.

For example, consider a scenario where the role of the students is to write the software and the role of the collaborator is to sell the software. The student contribution can be valued as a function of the time they have invested in writing the software. The collaborator contribution can be valued as a function of the sales leads they have generated. At the date of incorporation, you look at the value that everyone has contributed and divide the pie accordingly.

Other valuation functions are possible. The key to moving forward in your discussion at this stage is to focus on the valuation functions rather than who will own how much of a pie that has not yet been created.

It might be the case that some people eventually want to leave the project. For example, after graduation some students might want to get regular salaried jobs, whereas others might want to continue with the startup. In this case, the parties who remain in the company buy out the departing party according to the valuation functions mentioned above.

It might also be the case that nobody wants to continue with incorporation. In that case, everyone owns what they created (which is of no interest), and goes their separate ways to greener pastures.

AN OPTIONS POOL is a good way to buy out departing parties with future investor money rather than with your own. Here's how it works. In this scenario the departing party is leaving at the time of incorporation.

Some fraction of the company shares, say up to 10%, are allocated to the options pool. The value contributed to date by the departing party is calculated with the appropriate valuation function, say $x. The departing party then gets x share options: *i.e.*, one share for each dollar of value they contributed. The departing party can exercise the options to buy x shares for $\$\frac{x}{100}$: *i.e.*, a penny per share.

The anticipated scenario is that eventually a third-party investor comes along (*i.e.*, a venture capitalist). Investors buy shares in the company with money. Suppose the investor pays $y per share. The departed party exercises their options, buys their shares for a penny a piece, and then sells them to the investor for $y each. In this way the investor provides both the cash to buy out the departed party, but also a market valuation for what the company is worth. Obviously everyone hopes that $y is greater than the nominal $1 per share that was used when the options were created.

The above scenario assumes that incorporation is happening at the same time as gradution, and that the departing party is choosing to leave then, so they get options instead of shares.

Departing after incorporation also happens. In that case, the remaining parties might have the option to buy the shares from the departing party at $1 per share. In this way, the departing party gets paid for what they put in to the company before incorporation.

After incorporation those in the company continue to contribute value. That value can be recognized with cash or with shares. Amounts can be determined based on time worked (*i.e.*, a salary) or sales commissions or some other way.

Shares that *vest* over time are one way to approach this. For example, suppose that a person is to get 10,000 shares over four years: they would receive 2,500 shares per year until, after the fourth year, they had received all 10,000 shares.

Startups are a rocky road. At any moment someone might want to, or have to, or need to, leave. The best way to handle this is to always have agreed upon mechanisms and valuation functions for these departures to happen as smoothly as possible, with as little damage as possible to the company.

8.5 The Research Licensing Approach

Graduate students and professors collaborate with companies all the time. One of the main ways in which research funding happens in Canada is that the company gives some money to the university for research, and then the government matches that money.

There are a variety of ways that intellectual property concerns can be handled in this graduate student research context. One of the main approaches advocated by the University of Waterloo is that, consistent with Policy 73, whoever invents something owns it. Sticking purely with that position offers nothing to the company, however, and so is by itself inadequate.

What the university likes to offer to the company is the right to an exclusive commercial license to the invention(s). At the time the project starts nobody knows exactly what the invention(s) will be, nor their value. It is also next to impossible to determine an outcome-based valuation function in advance. So both the potential value of the invention(s) and the valuation functions that might be used are left unspecified, to be negotiated once the project is complete.

See UW-OCE-ScheduleD.pdf for one of UW's template agreements for projects funded through the Ontario Centres of Excellence (OCE).

All of the valuation functions we discussed above are based on inputs: how much did someone contribute? Whether that be in time, sales, *etc.* The bottom line, however, is whatever the market is willing to pay. The point of the valuation functions above is to ensure that everyone is adequately compensated for their inputs. In the research contract scenario, the graduate students and professors are already paid a salary for their time, so their inputs have been covered. What remains is to value the output through the market.

The Role of the Sponsor

The Software Engineering Capstone Design Project has had two sponsors: Qualcomm (2010–2012) and Yelp (2013–present). The sponsor provides funding for the following main expenses, the largest of which are the student awards:

- Early round in the first half of 4A: two prizes of $1000 each.
- Awards on Symposium Day:
  a. $3000 for first place
  b. $2000 for second place
  c. $1000 for third place
- Lunch and dinner on Symposium Day.
- Security for Symposium Day.
- Limited budget for supporting student projects.

In exchange for funding our Symposium Day and awards, the sponsor gets to select which teams win the awards,[1] have their name and logo printed on each poster, and make (limited) recruiting announcements.

[1] The formal system for Symposium Day awards works like this: the faculty selects a short-list of six teams; the sponsor selects at least two teams from this list and at most one wildcard pick not on this list; the sponsor chooses the ordering of the teams.

In practice it sometimes works a little bit differently. For example, in 2014, the sponsor named their top six teams and then the faculty approved that list. The sponsor had some difficulty deciding the third place winner from those top six, so the faculty suggested awarding two third place prizes.

Project Funding

We have limited funds available for capstone projects from the sponsor. Speak to the SE Administrator about making an expense claim. Funding is usually for hardware, but sometimes for travel. Other things might also be possible. Some notes:

- We don't fund hosting.
- We don't fund phones, tablets, laptops, or video game consoles.
- Funding is limited to a maximum of $500 per team, but might be less depending on overall class demand.
- Funded hardware is property of the SE program. After symposium day the hardware is offered to the lower classes. If nobody claims it by convocation (mid-June) then it is given back to the original team.

ECE does have some VM infrastructure that students can use. Speak to Eric Praetzel.

Funding decisions are made at the discretion of the Director of Software Engineering.

SE2016: https://www.youtube.com/playlist?list=PLRqQHlx1JrKzbsgF6DaRSHzECsmkHI_i6
SE2015: http://new.livestream.com/itmsstudio/events/3897191
SE2014: http://new.livestream.com/itmsstudio/events/2850051
SE2013: https://www.youtube.com/playlist?list=PLRqQHlx1JrKxMO8S3tlVWfziSxZu43hmR
SE2012: https://www.youtube.com/playlist?list=PLRqQHlx1JrKwbFk5t8vuNnwig5dVqX6LI

Awards

11.16 SE2016

11.16.1 Sleekbyte: Tailor, A Linter for Swift

Sleekbyte plans on creating a quality checking tool for Apple's Swift Programming Language. Swift is a compiled language created to replace Objective-C for iOS and Mac OS development.

There are currently a number of widely accepted style guides present for Swift. We plan on using these guidelines to create a static analysis tool that will improve general code quality in a large Swift codebase. We believe this will increase developer productivity.

Our product will analyze Swift source files line-by-line and point out any lines that don't adhere to a curated set of rules and guidelines, similar to tools like pylint and jshint.

First Prize $3000

Yelp! Earlybird prize $1000 in SE490

Autodesk Award $5000

CUSEC 2016 Demo Day 1st Place

Velocity W2016 $5k Competition Finalists

Adithya Srinivasan, Aditya Trivedi, Alykhan Kanji, Neha Singh

https://tailor.sh

https://github.com/sleekbyte/tailor

https://www.youtube.com/watch?v=ohLDYjAgN_g&index=7&list=PLRqQHlx1JrKzbsgF6DaRSHzECsmkHI_i6

Results

- 4000+ installs by the end of April 2016
- 600+ stars on GitHub by the end of April 2016
- Accepted into Homebrew package manager (http://brew.sh)
- Released a Code Climate engine (https://docs.codeclimate.com/docs/tailor)

Productivity

Measured with CLOC on April 26, 2016.

Language	*files*	*blank*	*comment*	*code*
Java	106	1584	1031	7772
Swift	55	1084	273	4360
XML	3	32	57	369
Groovy	1	38	50	181
Bourne Shell	6	35	19	167
Bourne Again Shell	1	20	21	123
PowerShell	2	25	14	75
Ruby	1	0	1	35
YAML	1	0	0	22
Sum:	176	2818	1466	13,104

11.16.2 Sana: Medical Protocol Builder

Second Prize (shared)

Yelp! Earlybird prize in SE490

Stephen Li, Joshua Kalpin, Connor Cimowsky, Fasih Awan

https://www.youtube.com/watch?v=QO6I6SyQIPc&index=2&list=PLRqQHlx1JrKzbsgF6DaRSHzECsmkHI_i6

Sana is a cross-disciplinary organization that uses open-source technologies to improve healthcare around the world. Sana Mobile, an initiative of the organization, allows doctors to formally define medical procedures using an XML syntax known as the Sana Protocol. Sana Protocol Builder is a graphical web application which aids doctors in the creation of these procedures, automatically generating the resulting XML for use by remote field workers via the Sana Mobile app.

Language	*files*	*blank*	*comment*	*code*
Javascript	101	1251	246	4627
Python	70	869	147	3337
LESS	13	247	10	1154
Handlebars	43	60	2	538
JSON	5	11	0	347
Ruby	8	37	13	206
HTML	2	3	0	15
Markdown	2	8	5	12
ERB	1	2	0	10
Sum:	245	2488	423	10,246

Figure 11.1: LOC for Sana'16 team as of Symposium Day. 'This greatly greatly increased from SE490 as we threw out our entire front-end. A majority of the Ruby and automation (travis, deploys, etc.) were done during SE390. SE490 most of the server was written, with a bit more written during SE491.'

11.16.2 Kaze: League of Legends Game Analytics

Second Prize (shared)

Kevin Bedi, Zijing Lin, Zhihao Mai, Andrew Siu

https://www.youtube.com/watch?v=AoZmm4XmrcU&index=12&list=PLRqQHlx1JrKzbsgF6DaRSHzECsmkHI_i6

LolPredict, League of Legends Game Analytics: An analysis tool for the popular online game League of Legends. The system is designed to help players analyze previous games and, using personalized trends, generate suggestions on the optimal way to approach a current game or how best to improve in the future.

11.16.2 unLit: Experimental Game AI

Second Prize (shared)

Jason Hau-Ken Chan, Taylor Petrick, Delia Yao, Jonathan Yep

https://www.youtube.com/watch?v=IQ2rvoL3RYE&index=13&list=PLRqQHlx1JrKzbsgF6DaRSHzECsmkHI_i6

unLit is a 3D top-down arcade game built for the PC platform. It showcases dynamically generated maps, custom models, and an experimental game artificial intelligence that adapts to individual players' playstyles. We investigate the feasibility of implementing an adaptive game AI by comparing it to one with deterministic (rule-based) logic.

11.16.5 Fusion: Automotive Sensor Fusion

GM Canada Award for SE

Rollen D'Souza, Bowen Xu, Jake Moffatt, John Zanutto

https://www.youtube.com/watch?v=qz_GgY_tZ6k&index=16&list=PLRqQHlx1JrKzbsgF6DaRSHzECsmkHI_i6

Contributing to vehicle automation research by assisting various UW research groups in combining useful toolchains. Implemented intuitive version of Lane Keeping Assist (LKA) and Adaptive Cruise

Control (ACC) on the PreScan Simulation Environment. Bridged PreScan simulation environment with the Robot Operating System (ROS), a tool used in autonomous vehicle research.

11.16.6 Parallax: WebGL Debugger

Honourable Mention
Aaron Morais, Adrian Cheung, Dian Xiang, Victor Lai
https://www.youtube.com/watch?v=JQ_F6l-i9ZU&index=5&list=PLRqQHlx1JrKzbsgF6DaRSHzECsmkHI_i6

Web graphics is a fairly new area in the graphics development community. Desktop graphics development has been the principal driver of progress in graphics technologies, and the state of software tools reflects it. Managing the complexity of the graphics pipeline is aided by these many tools, but very few of which are available to web graphics developers. Parallax is building a debugging toolkit for WebGL developers to recreate useful graphics development tools on the more complex WebGL graphics stack.

11.16.6 Maestoso: Academic Degree Audit

Honourable Mention
Chantelle Gellert, Mengqi Liu, Chunkai Yang, Xin Xie, Tao Lue Wu
https://www.youtube.com/watch?v=XUi10fHs5Kw&index=15&list=PLRqQHlx1JrKzbsgF6DaRSHzECsmkHI_i6

DegreeAudit is a software tool to automatically check if a student transcript meets graduation requirements. It comprises a domain specific language for defining graduation requirements for each department/programme at a university, as well as translators to encode the degree audit question for a specific student in boolean logic. The analysis returns reasons justifying its conclusion as to whether the student transcript meets the degree graduation requirements.

11.16.6 Persona: Personal Information Banking

Honourable Mention
Denny Kim, Jack Jiang, Ryan Crezel, Taylor Stark
https://www.youtube.com/watch?v=MFjInu9UOZw&index=3&list=PLRqQHlx1JrKzbsgF6DaRSHzECsmkHI_i6

Personal Information Banking aims to give consumers complete control over their personal data. From collecting new information, to aggregating information from current data sources, we want to create a more complete profile of consumers. We generate value for consumers by allowing them to control who accesses their data, and compensate them when their data is used.

by Taylor Stark

REFLECTIONS ON CLIENT. Our biggest highlight, in my opinion, was getting to work for a client. They were able to provide direction, advice, and really helped to keep us on task and on schedule. It was also a great way to meet people in an industry many of us had never worked in before, and the lessons learned about client interactions are something we'll definitely take forward with us in to the future.

SE490 end	5,000 LOC
SE491 end	13,000 LOC

PRODUCTIVITY. For productivity, we had about 5,000 lines of code by the end of SE490 and 13,000 lines of code by the end of SE491 (8,000 lines of code for the backend and 5000 lines of code for the front end). Much of our time in SE490 was spent coming up with ideas, testing prototypes and performing user studies. As we went in to our final work term, we continued to perform more user studies so we could iterate on our ideas and plans we had created in SE490. The rest of the work term was spent finalizing technical plans to realize the ideas we had decided upon. SE491 was really just a sprint to the finish, where we executed on all our months of planning and iterating on feedback.

11.16.6 Zephyr: Indoor Air Quality Monitoring

Honourable Mention
Aayush Rajasekaran, Andrew Gillies, Nathan Woltman, Tom Stesco
https://www.youtube.com/watch?v=Mk-zCwmBxoY&index=14&list=PLRqQHlx1JrKzbsgF6DaRSHzECsmkHI_i6

Despite the new research underscoring the importance of indoor air quality in human health and comfort and the existence of strongly advised indoor air quality standards, there is no existing, cost effective system that is able to audit and uphold indoor air quality standards. To solve this problem, we are building a system that is able collect and manage indoor air quality data by use of an air quality sensing device and cloud data processing and storage. This system will run air quality analysis algorithms and communicate to building owners (and optionally any interested party) the current and historic air quality status of a building.

11.15 SE2015

11.15.1 *Manifold: Circuit Design Language*

First Prize
Yelp! Earlybird Prize in SE490

Murphy Berzish, Tyson Andre, Lucas Wojciechowski, Max Chen

http://new.livestream.com/itmsstudio/events/3897191 — third video, first talk

Microfluidic circuits are currently designed by hand, using educated intuition to select values for design parameters. These hand-made designs are then simulated/verified using Matlab, to confirm that the values for the design parameters respect the design constraints. This is essentially a satisfiability (constraint satisfaction problem). Recent advances in SMT (Satisfiability Modulo Theories) solvers makethem applicable to this problem. A SAT solver finds a satisfying assignment (if one exists) to a boolean formula. An SMT solver extends the domain of the formula to include other kinds of variables. Recent SMT solvers can tackle formulas with non-linear inequalities and trigonometric functions over the reals. To make these advances accessible to microfluidics engineers we need to design and implement a hardware description langauge for microfluidics circuits. This will be inspired by, but differ from, hardware description langauges for digital circuits. In digital circuits the wires are all uniform size and can be any (reasonable) length and follow any path. In microfluidics the characteristics of the channel matter: diameter, length and pressure are related; angles cannot be too sharp or they will induce turbulent flow, etc. Although microfluidics will be the focus of our development, we believe that the resulting programming language will be powerful enough to describe a very large class of designs, including digital circuits, analog circuits, neural networks, and classical fluid networks (pipe systems). We plan to investigate the possibility of writing backends for these problem domains that will allow developers using our language to use it for a variety of real-world designs.

Productivity Metrics. Written in Java.

Reported by Murphy Berzish, June 2015.
Measured with cloc.

se490 midterm:

Component	*files*	*blank*	*comment*	*code*
Digital Backend	71	584	93	2197
Microfluidic Backend	37	368	334	2389
Core	71	584	93	2197
Engine	3	37	35	359
Frontend	77	727	752	3488
bash	1	21	21	122
DOS Batch	1	24	2	64
MATLAB	2	47	40	297
Sum:	263	2392	1370	11,113

Symposium Day:

Component	*files*	*blank*	*comment*	*code*
Digital Backend	24	416	271	2532
Microfluidic Backend	37	368	334	2389
Core	68	1022	337	4480
Engine	3	37	35	359
Frontend	58	903	1177	5018
bash	1	21	21	122
DOS Batch	1	24	2	64
Sum:	192	2791	2177	14,964

11.15.2 Accio: Realtime Codebase Analytics

Second Prize
Austin Dobrik, Jeff Lee, Matt Maclean, Alexander Maguire, Adam Sils

Accio is a collection of software tools to perform analytics on a software codebase in realtime, by integrating with programmers' editors to take "snapshots" of what they're currently working on. The system integrates these snapshots to determine which areas of the codebase have hidden dependencies and keeps track of who knows what about the codebase. We expect these measures to help both individual programmers to be able to implicitly search through the knowledge of others, as well as help the project maintainer find areas of the codebase that might soon become a maintainance nightmare.

11.15.3 TSK: Oculus Rift Virtual Keyboard

Third Prize
Ariel Weingarten, Mark Roukema, Adam Watson, Theodor Gugoiu, Saba Saba
http://new.livestream.com/itmsstudio/events/3897191 — fifth video @ 26:00m

Several virtual reality products have appeared on the market in recent years. Among them are the Oculus Rift headset and the Leap Motion controller. With the proliferation of this new hardware, more developers are seizing the opportunity to develop virtual reality applications. However, developing an application without a preexisting user control library can be time-consuming. In addition, the developers cannot be sure that their user interface is intuitive and easy to use. Hence, we have decided to build a library of user controls for use with the Oculus Rift, Leap Motion controller and the Unity environment. Our focus is primarily on developing text input fields and menu systems. After development is complete, we plan to conduct a user study to assess the usability of our user control library. We also plan to release the user control library to the public as open source software.

11.15.4 *Shuffle: DSL for Card Games*

Honourable Mention
Alex Swanson, Chhavi Kankaria, Holly Babaran, Sujen Sathiyanathan

To most people, programming seems cryptic and difficult. Kids in elementary school, Grades 5 and up, may find program syntax daunting. Shuffle is a domain specific language for creating card games. It aims to break down the barrier amongst those that do not attempt to learn programming due to scary syntax. Almost everyone is familiar with card games and the mechanics behind them; by leveraging familiarity with the card game domain, we aim to make programming more accessible. Shuffle is designed to help kids and others learn programming, possibly aiding the transition into programming in more commonly used languages.

11.15.5 *DataHarmony: Big Data Spreadsheet*

Honourable Mention
Yelp! Earlybird Prize in SE490

Desiye Collier, Alex Klen, Sanjay Menakuru, Jonathan Wei

http://new.livestream.com/itmsstudio/events/3897191 — last video, first talk

We are building a web spreadsheet application that translates Excel-like commands to relational database queries for efficient processing of big data. In particular, we want to make it possible for a non-technical user to create a report summarizing and aggregating the contents of a company's database and share a link to that report with another user.

11.15.6 *WhitewaterPlayer: Music Player*

Honourable Mention
Anushervon Saidmuradov, Viktor Stanchev, Francis Williams, Rui Zhao

WhitewaterPlayer is a music player that allows users to search, explore, stream and store music torrents on the web easily.

11.14 SE2014

Awards selected by Yelp representatives and SE alumnus Artem Avdacev and Rachel Zhao. SE Director Andrew Morton, SE Associate Director Patrick Lam, and Capstone Instructor Derek Rayside convinced Yelp to offer two third place prizes (Teams Remix and Radiant), rather than selecting just one of these two teams.

11.14.1 *Amalgam: A Solver for Multi-Objective Optimization Problems*

First Prize
Yelp Earlybird award in SE490

Joseph Hong, Chris Kleynhans, Ming-Ho Yee, Atulan Zaman

http://mhyee.com/fydp.html

Published a paper at ABZ'14 conference http://www.irit.fr/ABZ2014/

Multi-objective optimization problems can be found in many different engineering fields, such as aerospace engineering, civil engineering, and software engineering. However, Moolloy, a solver for these problems, is greatly limited by the time it takes to solve large problems. We have developed two different techniques for improving Moolloy. The first technique, which involves checkpointing and formula rewriting, reduces the solving time by a factor of 200, on average. The second technique, which involves parallelizing the optimization algorithm, allows us to leverage multi-core processors.

Productivity: I went through the different repositories to get lines of code. These numbers are very rough, due to several factors:

Reported by Ming-Ho Yee, August 2015.

- Some files weren't recognized by cloc, so I had to use wc.
- A large number of our tests were generated. I counted the lines in the test generator, but not the generated code.
- Many of our tests were hand-converted from Clafer models, and each one had ten variations. I ignored the variations.
- We were working in an existing code base, so I had to estimate how much code we actually wrote.
- There was a lot of refactoring and removal of old and unnecessary code. I ignored any code we removed.
- There were many branches, some of which were merged months later, and some which were never merged. I counted any code that was used for the results we presented on symposium day.
- The branches also include prototype/throwaway code that was later abandoned. I ignored this code.
- The test infrastructure and demo repositories include library code, which I excluded.

You should also count code that wasn't merged. Anything that was written.

So here are the numbers, with some context.

By the end of 390, we still did not have access to existing third-party source code. We spent the term getting familiar with the problem. In fact, it wasn't until the first week of December that we wrote 300 lines of scripts to generate test cases.

Over the work term, we worked on our test infrastructure and more test cases. We had about 1.2 kloc of infrastructure (excluding libraries), and about 2.7 kloc of new tests (a mix of test generating code and actual tests).

The majority of our code was written during 490. We mostly refactored and expanded our test infrastructure until late May (adding 500 loc), which was when we got access to the existing third-party source code. We finished refactoring in mid-June, and then after that, wrote the majority of our code (4.5 kloc). So the total code written during 490 was 500 loc of infrastructure, and 4.5 kloc of actual code.

We did not write any code over the last work term.

During 491, we made some negligible changes to our tests, did some minor refactoring of our infrastructure (adding 800 loc), added about 2.6 kloc, and wrote 900 loc for our demo.

This gives a grand total of 7.1 kloc (Java, with negligble C++), plus 3 kloc (Alloy, with negligible Java and Ruby) of tests, 2.5 kloc (Ruby) of infrastructure, and 900 loc (Ruby) of demo code.

11.14.2 RedCoconut: UWFlow.com

Second Prize

Yelp Earlybird prize in SE490

Y-Combinator interview (*verdict:* good team; project has no commercial potential

Mack Duan, David Hu, Jamie Wong, Sandy Wu

http://uwflow.com

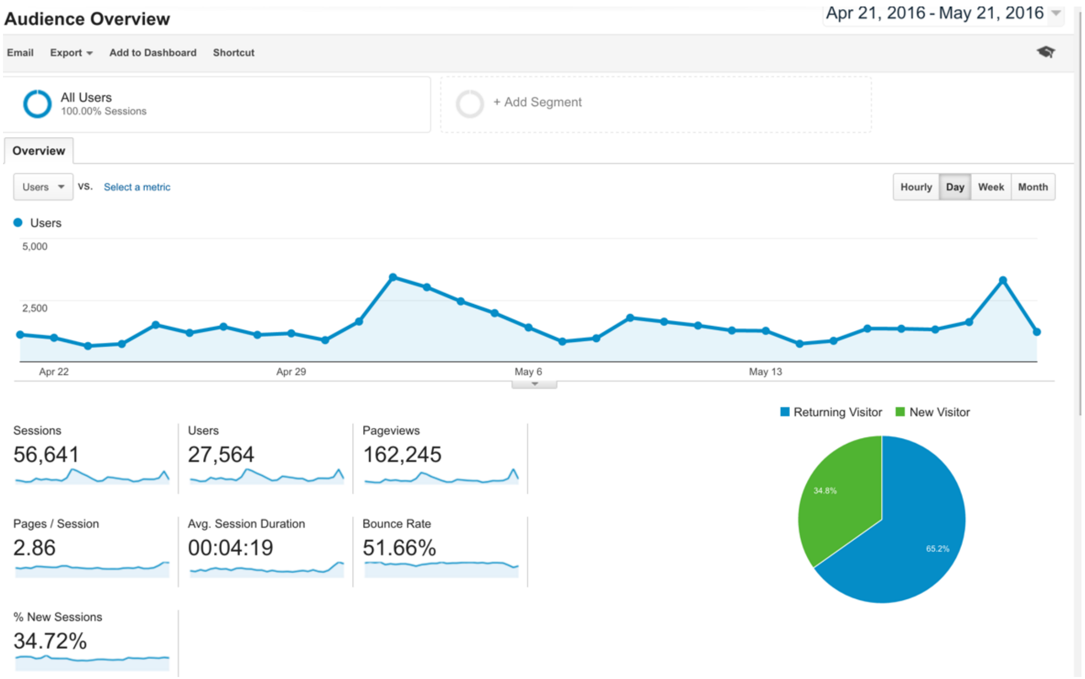
Had almost 5000 registered users by symposium day.

http://new.livestream.com/itmsstudio/events/2850051 — second video, first talk

Flow is a social course planning website for Waterloo students. It features a streamlined interface to search, filter, and sort courses; user-contributed reviews and ratings on courses and professors; class section information; and course descriptions. A student can create an account by signing in with their Facebook credentials. This lets a student upload courses they've taken by pasting in their Quest transcript; review and rate those courses; see what their Facebook friends are taking; and paste in their schedule from Quest and export that to third-party calendar applications (such as Google Calendar or iCal).

Flow was launched at the University of Waterloo in October of 2012. As of January 2014, it has 4375 registered users, 2339 of whom uploaded their transcript and 1424 of whom uploaded their schedule. Users have submitted 3766 written reviews and 41808 ratings. In the first week of January 2014, Flow served 18,502 pageviews to 4180 unique visitors.

USAGE UPDATE MAY 2016. UW Flow continued to grow after Symposium Day 2014. In the month from April 21 to May 21 2016 their Google Analytics stats reported 27,564 users. It reported similar numbers in previous months, with a low of 16,312 users in February 2016 and a high of 34,306 users in January 2016.



Audience Overview

Apr 21, 2016 - May 21, 2016

All Users
100.00% Sessions

Sessions
56,641

Users
27,564

Pageviews
162,245

Pages / Session
2.86

Avg. Session Duration
00:04:19

Bounce Rate
51.66%

% New Sessions
34.72%

11.14.3 Radiant: Automatic Classification of ElectroCardioGrams (ECG)

Third Prize
Jack Liao, Richard Tian, Tony Na, Stephen Dong, Zhen Zhang

Published a paper at IHTC'14 (after symposium day). http://ihtc.ieee.ca/

Our customer is a healthcare company in China. The company manufactures portable ECG recorders, and offers service to send captured ECG graphs to partnering hospitals for diagnosis. This is especially useful for elderly who may not wish to travel to hospitals regularly for checkups.

As a part of our project, we are working on extracting important features in the graphs, and researching on automatic ECG classification. Features such as the locations of prominent peaks help to improve visualization for the doctors; while having automatic classification will no doubt reduce the company's operational cost.

11.14.4 Remix: Music making for everybody

Norman Esch award for entrepreneurship worth $10k from the Faculty of Engineering (after symposium day)

Ryan Bateman, Wen-Hao Lue, Peter Sobot, Chris Taylor

http://mixbox.io

Music composition is an activity that has long been exclusive to those with musical training. Software applications in the music space have existed for decades, but nearly all of them require prerequisite knowledge or training in music to use effectively. **Mixbox** is a music composition app for the iPad that allows users of all skill levels to create unique music from existing music. It provides a user interface for music remixing that is easily used by non-professional music enthusiasts, but also includes powerful advanced features. Through advanced audio analysis and audio matching algorithms, Mixbox allows users to create new music that sounds professional and musical without requiring training in music theory, production, or composition.

11.13 SE2013

Awards selected by Yelp representatives Artem Avdacev and ???.

11.13.1 HiveMind: Framework for MMOGs

Renaud Bourassa, Adrian Rodrigo, Joshua Cowan, Robert Taylor
https://github.com/renaudb/hivemind

Massive Multiplayer Online Games (MMOG) have traditionally used a client-server architecture. In this model, the server holds the state of the game and is responsible for managing all updates to it. This approach results in heavy service cost due to the high traffic load on the game servers.

The goal of HiveMind is to reduce this burden by distributing most of the responsibilities of the server amongst the players in a peer-to-peer network.

Reported by Renaud, June 2015. Measured with cloc.

Productivity Metrics. We definitely ramped up late in the project. We spent a significant amount of time thinking about the protocol before working on the implementation. Also, one of our teammate was in San Francisco during the summer of 2012 which slowed us down.

SE390 final (roughly December 15, 2011):

Language	*files*	*blank*	*comment*	*code*
C++	2	42	0	184
Sum:	2	42	0	184

SE490 midterm (roughly June 15, 2012):

Language	*files*	*blank*	*comment*	*code*
C++	17	165	5	499
C/C++ Header	20	186	0	408
yacc	1	54	0	208
CMake	9	28	33	76
lex	1	15	0	34
Sum:	48	448	38	1225

Symposium Day (roughly March 15, 2013):

Language	*files*	*blank*	*comment*	*code*
C++	47	861	192	3331
C/C++ Header	54	557	78	1315
Perl	1	121	65	440
yacc	1	65	0	305
IDL	2	4	0	104
XML	3	0	0	74
CMake	8	29	30	74
lex	1	20	0	42
JSON	1	2	0	12
Bourne Shell	1	1	0	2
Sum:	119	1660	365	5699

11.13.2 Kinectitude: Framework for Kinect games

David Lawrence, Scott Reid, Ryan Tinianov, Brandon Walderman

git@bitbucket.org:kinectitude/kinectitude.git

We are a building a game development tool that allows beginner-level programmers, students, and hobbyists to create 2D arcade-style games for the Microsoft Kinect. This includes a graphical editor that allows the user to build a game from scratch using simple drag-and-drop techniques. Our product also includes the runtime that will read in a game built with our editor and run it. For more advanced users, an API is available to build custom extensions using .NET.

These productivity metrics were not available to the referees on Symposium Day 2013, because students were not asked to report them until se2016.

Measured by cloc.

Productivity Metrics. 15 kloc of the 24 kloc non-test code is devoted to the graphical editor. Total with tests around 29 kloc.

se490 final:

Program	*files*	*blank*	*comment*	*code*
C#	179	1738	487	10552
XML	9	325	36	1835
XAML	12	189	153	1549
MSBuild scripts	7	0	49	862
XSD	1	11	0	91
ASP.Net	3	0	0	23
Sum:	211	2263	725	14912

Test	*files*	*blank*	*comment*	*code*
C#	22	652	66	2362
MSBuild scripts	1	0	7	88
XML	1	6	0	24
Sum:	24	658	73	2474

Symposium Day:

Program	*files*	*blank*	*comment*	*code*
C#	314	3790	3191	20326
XAML	14	131	16	1786
MSBuild scripts	8	0	56	1483
ASP.Net	4	0	0	26
Sum:	340	3921	3263	23621

Tests	*files*	*blank*	*comment*	*code*
C#	32	1468	504	4935
MSBuild scripts	1	0	7	217
ASP.Net	1	0	0	11
Sum:	34	1468	511	5163

11.13.3 SecondScreen/Fauna: Augmented TV Viewing

Andrew Russell, Andrew Shapiro Munn, William Hughes, Noah Sugarman, Fravic Fernando

https://github.com/TeamFauna/dumbo.git

Second Screen is a better way to more efficiently split your attention between the TV and your Android phone. With one tap, it listens to and identifies what you're watching, then provides trivia and information about what's happening on screen. It continues to follow along with you as you watch and alerts you to important plot points, actors, trivia, insights, and more.

11.12 SE2012

Main awards selected by a panel of alumnus and other judges, on behalf of sponsor Qualcomm. Academic awards selected by faculty.

11.12.1 *GetHomeSafe*

x23huang, l23xu, n3zhong

Get Home Safe is a mobile application that allows the user to visualize and submit crime data using a map. It also provides a live tracking feature that allows users to notify emergency contacts when they feel unsafe and vulnerable, allowing others to see their real time location and visual data from a web interface.

11.12.2 *WatPark*

t2costa, tpjoynt, dmalakie, ewarsaba
http://www.watpark.ca

WATPark is a fourth-year design project focused on alleviating parking issues on campus. Through a collaboration with UW Parking Services, WATPark is able to give real-time car count data from Lot C. The project also provides other information regarding the remaining student lots, such as hours of operation, the location of each lot, and predictions on how full Lot C will be in the near future. The project can be accessed through the web at http://www.watpark.ca, and can also be found on the Android Market.

11.12.3 *Imgestion*

iaamariu, lajin, tchlam

Imgestion, short for “image ingestion", is a personal finance service that combines image processing with personal budgeting and context analysis to provide seamless expenditure tracking for the average person. Users simplify their finance tracking by capturing and uploading images of receipts to have their information extracted by the service. Imgestion also offers tools such as budgeting and analytics for the end user as well as intelligent tagging of expenditures by category.

11.12.4 *QuadForce (best presentation)*

kacjones, sblewchu, dspavel, aavoigt

Soapbox is an online platform for political discussion. Information is presented in a uniform, unbiased way, and the site allows users to annotate discussions with Twitter feeds, YouTube videos, and other relevant links. Discourse among users is encouraged, much like an in-person debate or a townhall meeting.

11.12.5 Blueshift (best poster)

madada, aaghi, aeaswara, o2farooq

Image search queries performed on an individual are usually not very diverse. The most well-known person possessing the search name dominates the result set. This is unfavorable when searching for a person who shares their name with a well-known individual. If additional context is not known (age, location, profession), it can be impossible to find search results on the target individual.

Diversify utilizes existing search engines and attempts to provide the most diverse result set for a particular query. For example, by leveraging services such as Google and Bing, we obtain a starting set of results. Through the use of image comparison and natural language processing (NLP), we are able to determine the amount of similarity between results and filter our result set to be as diverse as possible.

11.12.6 Symphony (best demo)

tgupta, s8hu, vyyli, c24yu

Chronostation is a how-to-draw vector illustration tool. Its main features are timeline and branching. The timeline feature allows users to playback the creation process of any illustration created through Chronostation. Users can also “branch" their drawings to create multiple versions of an illustration.

Good Example Projects

12.1 Finance-Related

12.1.1 MyMoneyMindset.com

12.2 Health-Related

12.2.1 BlackBoxHealth 2013

12.2.2 Sana.MIT.edu 2013

12.2.3 Communication Aid for Severely Disabled 2013

12.3 Commercial

12.3.1 TTC Trip Planner 2010

12.3.2 MyTopFans 2013

12.3.3 Étallonage 2014

12.4 Open Source

12.4.1 CryptKeeper 2014

http://new.livestream.com/itmsstudio/events/2850051 first video, first talk

Add support for Galois/Counter Mode in eCryptfs. eCryptfs is a cryptographic stacked Linux filesystem. eCryptfs currently uses Cipher Block Chaining (CBC) mode to encrypt files. By using Galois/Counter Mode instead of Cipher Block Chaining, we can provide integrity protection for files encrypted with eCryptfs. This means that we can detect modifications in files on the lower filesystem, and prevent the user from opening files that have been modified in the lower filesystem (by disk corruption, malicious actors, or otherwise).

12.5 Software Engineering Research

12.5.1 Whyness 2012

12.5.2 Amalgam 2014

12.5.3 GMV 2014

12.5.4 Satisfaction 2014

12.5.5 VisualStudio Enhancer (ECE 2014, 1st place)

12.6 Cognitive Science (with Paul Thagaard)

12.6.1 Causal 2012

12.6.2 2011?

12.7 Computational Neuroscience (with Terry Stewart)

12.7.1 Brainspawn 2014

12.7.2 Spike 2014

12.8 With UW/Parking Services/IST

12.8.1 WatPark 2012

12.8.2 NoManaZone (WatPark2) 2014

12.8.3 SSO International Peer Mentor (ECE 2014)

12.9 Miscellaneous

12.9.1 VerifyY: Citizen Social Science 2012

12.9.2 Vector Drawing (symphony/chronostation) 2012

Popular Project Ideas

This is a list of project ideas that we have built up in collaboration with Velocity. These are ideas that we see repeatedly, year after year.

Paul Graham, most famously known for his role in Y Combinator accelerator program, wrote that "smart people have bad ideas"[1], and later, "the best startup ideas seem at first like bad ideas"[2]. It is difficult to build a successful project around one of these ideas, let alone a business, but some students[3] have shown success in building both in these areas, provided they have a way to mitigate legal, user acquisition, and other hurdles in these spaces.

- Smartphone Apps
- Anything with NFC / RFID
- Anything with QR codes
- Anything with Bitcoin

  VCs find it less risky to invest directly in bitcoins than to invest in bitcoin-based businesses. The rationale? If bitcoin prices fall, so will the business's ability to profit. But it is also possible for the business to fail while bitcoin prices rise. So the business adds unnecessary risk over directly investing in bitcoin.
- Apps that help you meet up with friends more often
- Apps that help you meet up with friends by showing their location on a map
- Tracking food you eat
- Athlete tracking

  Commonly this is envisioned with an athlete wearing an LED-covered suit. This area is "patented to death", and entrepreneurs who attempted to build businesses in this area remarked that they couldn't figure out how to build the product without licensing the patents.

[1] http://paulgraham.com/bronze.html

[2] http://paulgraham.com/swan.html

[3] Such projects include Singspiel.ca, a medical records project with sanu.mit.edu, and BlackBoxHealth.

- Grocery list generators
- Apps for new parents

  New parents are too tired to use apps that are "nice to have" after the second or third week.

  (e.g. a photo album app containing photos of your children)
- 3D-printed Products

  Students usually envision products which don't fit well with the constraints of current consumer-grade 3D printers; their designs take too long to print, or contain overhangs that fail to print at all.
- Software targeting Education / Schools / Universities

  These institutions' software decisions are usually made on a basis of cost, not on a basis of software quality. The "Request for Proposal" (RFP) processes usually used by larger institutions for procuring software are also difficult to navigate for newcomers.
- Software that organizes Television shows / Movies / Videos

  Licensing TV/Movie content requires lots of lawyers.
- Medical records

  There are many legal requirements for handling medical data, including the Health Insurance Portability and Accountability Act (in the United States) and the Personal Health Information Protection Act (in Ontario). In particular, health records in Ontario may not be transmitted over the "US Internet", so the current industry norm is to use fax machines and dedicated ISDN lines.
- Tutor/mentor matching
- Automatic homework/activity scheduling
- Used clothing marketplace/exchange
- Clothing image recognition
- To-Do List
- Electronic / shared / communal whiteboard
- Whiteboard capture (video camera / phone camera / scanning / smart board)
- Super thin nanotechnology (contact lens, virtual nails)
- Winter snow clearing (robot or wires)

- Music teaching/tutor app

  Integration with MIDI-capable instruments is difficult on modern computers, and real-time signal processing is difficult. Live human music teachers tend to be cheaper and better than the cost of developing such software. Finally, many people already use YouTube videos as a medium for teaching/learning musical skills.

- Karaoke

  Acquiring rights to songs requires lawyers (expensive!). Also, most consumers of karaoke software/hardware are not willing to pay very much for it.

- Unified (calendar, sync, files, dropbox)

  See: https://xkcd.com/927/

- Online coupons/flyers
- SMS or App-based iClicker replacement
- Software that targets children

  Collecting information from children is regulated by the Children's Online Privacy Protection Act and other laws.

- Alarm clock apps (weather, email)
- Collaboration using existing text editors (word, emacs, vim)
- Unified remote / IR blaster on phones
- Store receipts / receipt tracking
- Expense splitting/IOUs app
- Integration with retailers / Point-of-Sale
- Ride-sharing/carpool matching
- Anything that relies on advertising or freemium business models
- An app that automatically texts your girlfriend for you
- A device (e.g. sticker) that helps you find lost items (e.g. keys) using a mobile app
- Review sites (e.g. past landlords / rental properties)
- Augmented reality

Student Wisdom

Notes from RCH112, 11:30am, April 1, 2014. SE2014 sharing their hard-won wisdom with SE2016. Recorded by Joshua Kalpin. Main text from students. Marginalia from faculty.

14.1 Wisdom from SE2014

14.1.1 Returning Projects

- UWFlow: uwflow.com
- The Notist notist.co
- WatPark watpark.ca

14.1.2 What They Wish They Knew

- Make sure your entire group isn't taking trains, graphics, etc. all at the same time.
- Don't work with your significant other. Group breakups are bad.
- Start generating ideas early on, even before starting on design project.
    - Keep a daily journal of ideas that come to mind and then discuss them with others
- Keep a goal in mind for symposium (e.g. have lots of users) and work backwards from there
- If your application requires users launch early
- Don't necessarily choose long-term technologies, it can be important to launch soon.
- Start early and launch early. It allows you to write higher quality code in the long run.
- If you are doing a research project, make sure it has good engineering. Good infrastructure, testing, build systems, etc. It can pay off in the long run (figuratively and literally).
    - Top two teams in 2014 had amazing infrastructure
    - Build/Test infrastructure pushed Amalgam over the top
    - The sponsor loves this (=> $$$)
- Research code is awful. Make sure your code is actually good. Crappy code is really bad

This can work well. Or end badly.

We try to facilitate the generation and rejection of ideas in SE390. The more you practice this the better you will be at it. It can be tough, both intellectually and emotionally. Learn how to use someone else's criticism of your ideas to make the ideas better.

  - Apparently, Rayside's code is not the WORST, but still bad

Thank you Ming-Ho for that insight.

- Customer Definitions:
  - Research -> Professor
  - Open Source -> Maintainers
  - Corporate -> Actually Client
  - Entrepreneurial -> More of a technical mentor
  - Language will be clarified in 3B
- Be pro-active about ideas & team.
- Schedule frequent meetings (Mixbox did 3 times a week). Used to work on the project
- Having one strong voice in the project to make defining decisions (Team Lead, Vocal People, etc.). Gives motivation and guidance
- If you get a lot of work done in the school terms, you most likely won't get it done over the workterm
  - Hard to arrange meetings, communication, etc.
- Need good tools to figure what everyone is doing.
  - Pivotal Tracker for task planning (iterations)
  - Hipchat for communication
  - Github for Pull Requests + Code Reviews
  - Helps for asynchronous communication

Many of the panelists were very enthusiastic about Hipchat. Apparently much better than Skype or GChat for collaborative coding.

- Having a group member going on exchange can have a lot of difficulty
  - Weird meeting times because of time zones, exam times, etc.
  - You don't see them every day
  - Workload won't be balanced while on exchange
- Inter-disciplinary groups can have some unique quirks
  - Hard to schedule
  - But can give a different perspective
- 80/90% of work will be done together

14.1.3 *What's a good strategy for maintaining momentum during co-op?*

A1: Teams can potentially live together during a co-op term (may drive you insane). Be careful of doing this ... REALLY CAREFUL!

A2: Chris & Ming-Ho lived together and didn't murder each other. Results may vary. Still tried to schedule regular meetings (cross-time-zone). Built the entire infrastructure during the co-op term. However, it was not completely steady throughout the work term. tldr; Meetings and if they didn't do it would be bad.

A3: Research and actively still looking at projects etc. can help a lot during the term. Examples are testing, different libraries to speed up development. Play with 'competitors' if you have them — could learn things.

A4: It may not be possible to work on it during a work term. Amazon, Google (Interns) and IBM are really bad at this. Keep this in mind.

A5: Set deadlines for yourself (not necessarily meaningful).

14.1.4 What are measures of success for a FYDP?

A: Measured in technical content, results and communication. Less of a undergraduate marking scheme. Pick a project that someone will care about after your graduate and do a good job. Demonstrate how people care about your project. Project specific basis for success (going add examples). UWFlow is the low bar for doing a CRUD project that everyone has tried before (did YCombinator pitch). We'll get a list of projects that have done well as examples.

Show that people care by: building a user base; having the maintainer accept your patch; winning a pitch competition; publishing a scientific paper; *etc.*

CRUD := Create, Read, Update, Delete

14.1.5 How does a measure of success change if it is a taken over project?

A: Choose a project that as an extension is as ambitious as the original project.

There have been at least two successful projects that were extensions of pre-existing projects. WatPark v1 was done in SE2012, and v2 in SE2014. UW Parking Services is still interested in further development. MyTopFans was a pre-existing Facebook application that also became a project in SE2013.

14.1.6 When do you know when you want to work on something?

A1: A lot of ideas will be pitched to us from various locations. Don't worry too much about thinking of an amazing idea.

A2: Massive brainstorming sessions with people you want to work with can be useful.

A3: Can be a challenge to rally people around your idea, it may not be interesting to anyone but yourself. Think about what sorts of things you have in common with your classmates to direct ideas.

A4: This process will tried to be facilitated in 3B (SE 390). Keep all the other ideas for a potential SE499 (Independent project) if it doesn't work out for FYDP.

A5: When deciding whether to organize an idea or a team, Mack says to organize around a team rather than an idea. Can usually find some interest. However, Rayside says to remain open-minded.

14.2 Wisdom from SE2012

Notes from MC4059, 3pm–4pm, April 5, 2012. SE2012 sharing their hard-won wisdom with SE2014. Main text from students. Marginalia from faculty.

14.2.1 Team Selection is Important

Team Selection is important. Breaking up a group is next to impossible.[1]

[1] In SE2011 a group break up precipitated a student transferring out to Computer Science.

Select teammates who have a similar work rhythm and habits. Get up early? Stay up late? Like to start assignments when they are handed out or when they are due? Like to keep files organized or organic?

Be open to the possibility of working with someone new. People who are good friends do not always make good teammates (nor roommates).

Be cautious about working in the same group as your girl-friend or boy-friend. Being in the same group can strain your relationship. If your relationship ends before you graduate you will still have to work together in the same group.

14.2.2 Save Early, Save Often

Make frequent commits with meaningful commit comments. Infrequent commits with empty comments will cause you grief in the not-too-distant future.

14.2.3 Learn some new technology.

Don't just use the technologies that you are already familiar with. Use the project as an opportunity to learn a new technology.

14.2.4 Use Tools

Build system, test harnesses, bug database, *etc.*. The design project is large enough that you will need these things.

GitHub has an integrated issue tracker and automatically links commit comments to issues.

14.2.5 Regression Tests

You will refactor your code. Without tests you are likely to break as many things as you fix during this process.

14.2.6 Focus on Core Features First

It feels terrible to be working on core features the night before the final symposium. Build a skeleton first and then flesh it out as you go.

14.2.7 Hallway Usability Testing

Get your classmates to try out your project a month or two before the symposium. Getting feedback earlier will help you improve the project before the symposium.

The term 'hallway usability testing' here is borrowed from The Joel Test.

Or even earlier.

If you are making a mobile app, post it in the appropriate marketplace so that people can download it.

14.2.8 Have a Group Decision-making Protocol

Groups from both SE2011 and SE2012 have recommended this.

It is normal for teammates to disagree about things. That's fine. In the event that the team cannot come to a consensus on an issue, there needs to be a pre-defined decision making process. In the absence of of a pre-defined decision making process, decisions on contentious issues will often be made by force of personality: *i.e.*, the person who talks the loudest or threatens to quit the group will get their way. Making decisions in this way rarely contributes to group harmony; just about any other approach is better.

Some simple group decision-making protocols include having an appointed leader, voting, and flipping coins.

Be ready to compromise. Recognize that many of the decisions you will have to make will have little impact on your final mark.

14.2.9 Group Name

Consider changing your group name at the beginning of 4A. You may have selected your group name before you knew what your project was. If you want to change your group name to match your project then the best time to do it is right at the beginning of 4A.

14.2.10 Have a Group Email Address

Have a group email address that you use to sign up for online services that the group is using. If you sign up in one person's name and that person gets sick then the group will have problems that could have been avoided.

14.2.11 You are Working Towards Symposium Day

You are working towards symposium day. At the end of 4B there is symposium day, and from the academic perspective that represents the sum totality of your project: a poster, a demo, and a presentation. Use this end-goal to frame the decisions you make along the way: *i.e.*, how will this issue change what we present on symposium day?

Truly successful projects will live on beyond symposium day, but that success exists outside the academic context.

Look at examples of past symposium day artifacts to get a sense of what goes into them and what makes them good.

14.2.12 Award Decisions are Made by External Parties

External parties will judge the idea of your project more than its implementation. Symposium Day prizes that are awarded by sponsors or alumni might be skewed towards projects that have an idea that appears useful, interesting, or viable to the general public.

The academic grades and prizes are determined more on the quality of your work, both technical and expository.

14.2.13 Brainstorming is Fun

Record your brainstorming sessions some how: *e.g.*, take pictures of the whiteboard, *etc.*.

At some point you have to make decisions and start coding.

14.2.14 Break Work up into Chunks

Some students recommend a chunk size of 40 hours.[2]

14.2.15 You do not need to pay for an SSL certificate

There are various websites where you can get an SSL certificate at no charge. Buying a certificate can be expensive.

14.2.16 Hosting Services

Most students have found technical problems with GoDaddy and recommend avoiding it. Experiences with Linode have been both positive (bare metal server) and negative (my server got rooted).

14.2.17 ORMs Have Limitations

Experimental prototypes can help you assess and manage the limitations of your chosen technologies.

All object-relational mappers have limitations, often around joins. You may end up writing straight SQL anyways, for either expressiveness or performance reasons.

[2] Some agile methods recommend alternative chunk sizes.

14.2.18 Know Thy Self

From student email.

The actual time for projects implementation is much closer to 2 months than 12 months of the project or even 4 months of the 4A term. We struggled throughout the project figuring out whether or not we can attempt certain features, and I'm glad we cut most things out and polished the few core project components. In other words, 'know thy self' and be honest with each other about the likelihood of working during work terms, weekends, *etc.*. The team that I feel did the best job of scope this term is WatPark. Deployed working application that does one thing only. They should be a good example for others.

14.2.19 Great Projects Live On

From student email.

On the symposium day the best feeling is having the project deployed, useful or in some other way available for use after 4B is done. Having it vanish feels bad.

14.2.20 Choosing a Customer

From student email.

There are different kinds of customers, but they fall into at least two broad categories, those who have a project in mind that you work on and those where you provide the idea and they act as your customer. Examples of groups with the first kind of customer are Whyness or Causal. The advantage of this type of customer is that you get a set of specifications from your customer to base your work on. The level of detail of these specification can of course vary but they do exist. This means that your team has some basic direction to help guide you and can help make the high level design decisions. The other type of customer is demonstrated by Quadforce or Synchronisity where the initial project idea was entirely developed by the team and presented to the client. The client (at least in our case) was still someone knowledgeable in the area and able to help guide us but at the end of the day our team had to fully develop the entire design and make all of the high and low level decisions ourselves. This arrangement allows teams to go through the complete ideation process from the forming of the idea through design and implementation however it means that you have no one, and no spec doc to fall back on when making your decisions. Both models have their advantages and disadvantages for the design project as they offer two distinct and relevant experiences, the first is a more traditional client relationship, and the second is a much more entrepreneurial type of experience. Thus, it is important when looking for your client you know what kind of relationship you want with them and what they will bring to your project.

The key message is that there are different relationships available with different clients, each of which offers a different experience for teams, so they should keep that in mind when selecting a project or customer.

14.3 Wisdom from SE2011

All main text from student email. Marginalia from faculty.

14.3.1 Professional Engineers Anticipate Problems

The most important lesson that I learned from the design project is to think about potential failures as early as possible in the development life cycle. You can call this paradigm as failure-oriented programming, if you will.

This is all based on my experience, and my teammates may not necessarily endorse my opinion.

In order to code in iPhone, we must use Objective C. However, none of us was an expert in this language when we started the project. Consequently, there are some quirks that are specific in Objective C framework. One such example is exception handling. There are some methods that are run by the iOS framework on a separate thread (not main thread). Because our code runs on the main thread, when we call those methods, because those methods live in a different thread, we can't catch an exception on that thread. Unfortunately, we customized those methods to throw an exception upon connection failures. We didn't encounter any problem on the simulator because we had perfect connection on the wireless network all the time. But, when we actually tested it on the iPhone, we discovered that it crashed all the time upon connection failure.

Learn how exception handling and threading work in your language of choice.

Know the differences between your simulation/test environment and the deployment environment.

This caused us a lot of grief because in order to fix this problem, we either have to resort on changing the architecture of our program in a major way or resort on putting ugly hacks everywhere. Due to time limitation, we decided to use hacks.

Had we actually tested our program on the device on the early prototyping stage, we would have caught this problem much sooner, and we would actually design the program to handle this quirkiness of the framework.

So, my words of advice for others is to always consider failures at any stage of development, especially at the early stage.

Consider writing a vertical experimental prototype to get familiar with new technology before using that technology to build your system.

14.3.2 Schemas and APIs Evolve; Decisions Have to be Made

I can think of 3 things that maybe you can turn into wisdom.

Initially we had some issues with the schema in that our model continued to get more complicated as we explored it — we ended up representing our data essentially as objects that could be represented as JSON and this, for the most part, was good for us.

Having backward compatibility with an early design was a bad idea. During the change to a generic object store from a rigid schema we were likely to have to change our API. This caused some disagreement as not everyone was happy with having to rewrite the dependent code. The solution was to build an adapter that presented the original API and did some translation. This adapter ended up causing many problems, hours of work and blocking of dependent work; especially later on. Ironically very little code that used the original API still existed; but new code had still been written to use that API.

Use vertical experimental prototypes to explore data schema design alternatives.

Don't be afraid to throw out old code if you are making a significant design change.

Finally trying to please everyone was a problem we, for the most part, solved. While this is more of a group organization issue it did have an impact on the design. We saw early on that it was important to have a leader that "owned" the problem of getting work done on our project. This person would have the final say in setting short-term, allocating tasks, and coordinating meetings. Initially goals were all decided by consensus. One of the most polarizing points was whether or not to have the majority of the work done on the client-side where there were relevant Javascript libraries; or server-side where we could have more compatibility for free without being Javascript dependent. We tried to tackle both without a clear strategy of where the divide was, or exactly what device we were supporting. In the end, we ended up featuring cross-platform Firefox support (we also worked on Chrome, but rarely tested) with Javascript enabled. This was partly due to a change in policy to have a permanent leader (rather than rotating monthly) who had final say in everything. This ended up being a good idea, though late in the game.

Have a clear leadership structure.

I'm hoping this is helpful to you. Thanks for asking us :)

14.3.3 *Use Your Work Terms Wisely*

The best piece of advice that I would give to future students would be to work on their design projects over their work terms.

Our group did this, and it greatly reduced our workload for 4B. Considering that I was taking 3 ATE's (advanced technical electives) at the time, having a good chunk of the design project done was much appreciated.

Work on your design project during your work terms.

I know it can be hard to come home from co-op and then have to do even more work, but even a little bit here and there helps out in the end.

Have fun with the next group of SE students!

14.3.4 Technology, Scope, and DB Choice

I'm not sure if these are useful to the course but here are some key points I have looking back over my FYDP project:

- I wasn't knowledgeable in the area of technology our group was working with when 2 of our members were extremely proficient and was also a past idea they've played around with. The disparity in starting knowledge made the contribution of the overall project drastic and difficult to catch up on. Looking back, not only would I suggest to someone to really aim for something novel however if new technologies will be chosen that all members have limited experience with them.

Consider writing vertical experimental prototypes to get familiar with new technology.

- Division of task wasn't clear and neither was the scope of the project. Our project felt as if it was constantly having things (new technologies & functionality) tacked on in order to make it seem more substantial.

Have a clear leadership structure.

- Our project relied heavily on a very very loose relational model (NoSQL essentially) however there was no real outlined way laid out for dealing with it. We didn't properly eat our own dog food because the API's were dysfunctional. API's are more difficult to create than they seem.

Write a vertical experimental prototype to explore your data model.

Eat your own dog food.

It's hard to write a good API.

Questions from SE2014 Symposium

These are questions that were asked on symposium day by audience members. Typically these are questions that the teams should have thought about at the beginning of 4A. On any project there are some things you will miss, but it's better to take a broad view at the beginning to ensure you have your bases covered.

15.1 Why did you use a NoSQL database?

Admitting you made a mistake is better than being surprised by the question.

UW Flow answered this question preemptively in their talk: it was a sub-optimal choice in some respects. Their data are inherently relational and their system requires many joins. Computing those joins in imperative code over a NoSQL database was cumbersome (for the programmers) and inefficient (for the computer). Their site will probably not scale to millions of users (although it is fine for the 4,700 users they had as of symposium day).

The reason they used a NoSQL database at the beginning is that it supported fast evolution and they were familiar with it. Getting the site up and running quickly was key to building their user base. It is possible that they might have saved development effort overall if they had made an effort to learn a relational database engine at the beginning.

15.2 Your user study was poorly designed

Study background in related fields.

UW Flow had a large user study, but one of the referees believed that the results from it were not as useful as they could have been because the study was poorly designed. If you are going to make the effort to conduct a large user study, it is better to also make the effort to design it well.

15.3 Why did you ask absolute rather than comparative questions?

Study background in related fields.

Team Club Sandwich conducted a user study to create training data for their machine learning algorithm. They asked people to classify photographs as interesting or not interesting. The referee pointed out that in Psych-101 they would have learned that it is better to ask people comparative questions: *i.e.*, which of these two pictures is more interesting?

15.4 What happens when students graduate?

Think about the broader context.

Team SBF created an application for students. A referee asked what happens when the students graduate? Do they still have access to their data? Are the data destroyed safely? *etc.*

15.5 Cryptography: seed values and reordering

Understand your domain.

The Cryptkeeper group learned the hard way that cryptography is hard. Their initial response from the Linux kernel mailing list was that their patch 'would destroy the security of eCryptfs' because they failed to initialize the seeds properly.

They subsequently discovered that their patches were not able to detect when the input blocks had been reordered (but otherwise unmodified).

In both cases the team was able to fix the problems, but thought that if they had spent a bit more time studying the problem they could have avoided these pitfalls.

University Policies

Academic Integrity: In order to maintain a culture of academic integrity, members of the University of Waterloo community are expected to promote honesty, trust, fairness, respect and responsibility.

http://uwaterloo.ca/academicintegrity/

Grievance: A student who believes that a decision affecting some aspect of his/her university life has been unfair or unreasonable may have grounds for initiating a grievance. When in doubt please be certain to contact the department's administrative assistant who will provide further assistance.

Policy 70, Student Petitions and Grievances, §4, http://secretariat.uwaterloo.ca/Policies/policy70.htm

Discipline: A student is expected to know what constitutes academic integrity to avoid committing an academic offence, and to take responsibility for his/her actions. A student who is unsure whether an action constitutes an offence, or who needs help in learning how to avoid offences (*e.g.*, plagiarism, cheating) or about rules for group work/collaboration should seek guidance from the course instructor, academic advisor, or the undergraduate Associate Dean.

http://uwaterloo.ca/academicintegrity/

For information on categories of offences and types of penalties, students should refer to Policy 71, Student Discipline, http://secretariat.uwaterloo.ca/Policies/policy71.htm

For typical penalties check Guidelines for the Assessment of Penalties, http://secretariat.uwaterloo.ca/guidelines/penaltyguidelines.htm

Appeals: A decision made or penalty imposed under Policy 70 (Student Petitions and Grievances) (other than a petition) or Policy 71 (Student Discipline) may be appealed if there is a ground. A student who believes he/she has a ground for an appeal should refer to Policy 72 (Student Appeals).

http://secretariat.uwaterloo.ca/Policies/policy70.htm

http://secretariat.uwaterloo.ca/Policies/policy71.htm

http://secretariat.uwaterloo.ca/Policies/policy72.htm

Note for Students with Disabilities: The Office for Persons with Disabilities (OPD) collaborates with all academic departments to arrange appropriate accommodations for students with disabilities without compromising the academic integrity of the curriculum. If you require academic accommodations to lessen the impact of your disability, please register with the OPD at the beginning of each academic term.

OPD: Needles Hall, Room 1132

Plagiarism Detection Software: The instructors may, at their discretion, use plagiarism detection software.

e.g., TurnItIn.com, MOSS

Appendix B
Bibliography

[1] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley, 2 edition, 2003.

[2] Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, 1975.

[3] Avi Bryant. Web Heresies: The Seaside Framework. OSCON, 2006. URL http://conferences.oreillynet.com/cs/os2006/view/e_sess/8942.

[4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. John Wiley & Sons, 1996.

[5] Canadian Engineering Accreditation Board. Accreditation criteria and procedures, 2013. URL http://www.engineerscanada.ca/sites/default/files/sites/default/files/accreditation_criteria_procedures_2013.pdf. Retrieved winter 2014.

[6] Computer Science Accreditation Council. Accreditation criteria for computer science, software engineering and interdisciplinary programs, August 2011. URL http://www.cips.ca/sites/default/files/CSAC_Criteria_2011_v1.pdf. Retrieved winter 2014.

[7] Edward W. Constant. *The Origins of the Turbojet Revolution*. The Johns Hopkins University Press, 1980.

[8] Melvin E. Conway. How do committees invent? *Datamation*, 14(5):28–31, April 1968. URL http://www.melconway.com/research/committees.html.

[9] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the*

4th ACM Symposium on the Principles of Programming Languages (POPL), Las Angeles, CA, January 1977.

[10] Pierre Cros. *Imagination, undeveloped resource : a critical study of techniques and programs for stimulating creative thinking in business*. Creative Training Associates, New York, 1955. URL http://hdl.handle.net/2027/uc1.l0050673813. Submitted in partial fulfillment of the requirements in Professor Georges F. Doriot's course in manufacturing at the Harvard Business School.

[11] Alan M. Davis. Operational prototyping: A new development approach. *IEEE Software*, 9(5), September 1992.

[12] Edward de Bono. *New Think: The Use of Lateral Thinking*. Basic Books, New York, 1967.

[13] Edward de Bono. *Six Thinking Hats: An Essential Approach to Business Management*. Little, Brown, & Company, 1985.

[14] Clive L. Dym and Patrick Little. *Engineering Design: A Project Based Introduction*. John Wiley & Sons, 3 edition, 2008.

[15] Michael D. Ernst. Static and dynamic analysis: synergy and duality. In Jonathan E. Cook and Michael D. Ernst, editors, *Proceedings of the Workshop on Dynamic Analysis (WODA)*, Portland, Oregon, 2003.

[16] Eugene S. Ferguson. *Engineering and the Mind's Eye*. The MIT Press, Cambridge, Mass., 1992.

[17] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.

[18] Richard P. Gabriel. Lisp: Good news, bad news, how to win big. *AI Expert*, pages 31–39, June 1991. URL http://www.dreamsongs.com/NewFiles/LispGoodNewsBadNews.pdf. Presented as the keynote address at the European Conference on the Practical Applications of Lisp, Cambridge University, March 1990. Commonly known as 'Worse is Better'.

[19] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[20] David Garlan and Dewayne E. Perry. Introduction to the special issue on software architecture. *IEEE Transactions on Software Engineering*, 2(1), April 1995.

[21] David Garlan and Mary Shaw. An introduction to software architecture. Technical Report CMU-CS-94-166, Carnegie Mellon University, January 1994. URL http://www.scs.cmu.edu/afs/cs/project/able/ftp/intro_softarch/intro_softarch.pdf.

[22] David Garlan and Mary Shaw. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, Inc., 1996.

[23] H. Goldstine. *The computer from Pascal to Von Neumann*. Princeton University Press, 1972.

[24] William J.J. Gordon. *Synectics: The Development of Creative Capacity*. Harper & Row, New York, 1961.

[25] C. A. R. Hoare. The emperor's old clothes. *Communications of the ACM*, 24(2):75–83, February 1981. Acceptance speech for 1980 Turing Award.

[26] C.A.R. Hoare. Hints on programming language design. Technical Report STAN-CS-73-403, Stanford, December 1973. URL http://web.eecs.umich.edu/~bchandra/courses/papers/Hoare_Hints.pdf. Keynote talk at POPL'73.

[27] IEEE. Recommended practice for architecture description of software-intensive systems. Technical Report ANSI/IEEE 1471-2000, 2000. URL http://www.iso-architecture.org/ieee-1471/.

[28] Michael A. Jackson. The Name and Nature of Software Engineering. In Egon Börger and Antonio Cisternino, editors, *Advances in Software Engineering: Revised Lectures of Lipari Summer School 2007*, volume 5316 of *Lecture Notes in Computer Science*, pages 1–38. Springer-Verlag, 2008.

[29] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. seL4: Formal verification of an OS kernel. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP)*, Big Sky, MT, USA, October 2009.

[30] H.-Y. Benjamin Koo. *A Meta-language for Systems Architecting*. PhD thesis, Engineering Systems Design, Massachusetts Institute of Technology, 2005.

[31] Philippe Kruchten. *The Rational Unified Process*. Addison-Wesley, 1999.

[32] Butler W. Lampson. Hints for computer system design. *ACM Operating Systems Review*, 15(5):33–48, October 1983. URL http://research.microsoft.com/en-us/um/people/blampson/33-hints/webpage.html. The online version is slightly revised.

[33] Bill Moggridge. *Designing Interactions*. The MIT Press, Cambridge, Mass., 2007.

[34] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. How Amazon web services uses formal methods. *Communications of the ACM*, 58(4): 66–73, 2015.

[35] Alex F. Osborn. *Applied Imagination: Principles and Procedures of Creative Problem Solving*. Scribner & Sons, New York, 1953.

[36] David Lorge Parnas. On the Criteria to be Used in Decomposing Systems into Modules. *Communications of the ACM*, 15(12):1053–1058, December 1972.

[37] Terence Parsons. The traditional square of opposition. *The Stanford Encyclopedia of Philosophy*, (Fall 2008 Edition), 2008. URL http://plato.stanford.edu/archives/fall2008/entries/square/.

[38] Christian Queinnec. Inverting back the inversion of control or, continuations versus page-centric programming. Technical Report 7, LIP6, May 2001. URL http://www.lip6.fr/reports/lip6.2001.007.html.

[39] Daniel Sanders and Paul Thagard. Creativity in computer science. In James C. Kaufman and John Baer, editors, *Creativity Across Domains: Faces of the Muse*. Lawrence Erlbaum Associates, Publishers, 2002.

[40] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects*. Addison-Wesley, 2000.

[41] Willard Simmons. *A Framework for Decision Support in Systems Architecting*. PhD thesis, Aeronautics & Astronautics, Massachusetts Institute of Technology, 2008.

[42] Richard N. Taylor, Nenad Medvidović, and Eric M. Dashofy. *Software Architecture: Foundations, Theory and Practice*. John Wiley & Sons, 2009.

[43] Walter G. Vincenti. *What Engineers Know and How They Know It: Analytical Studies from Aeronautical History*. The Johns Hopkins University Press, 1993.

[44] Eugene K. Von Fange. *Professional Creativity*. Prentice Hall, Englewood Cliffs, N.J., 1959.

[45] Wikipedia. CAP Theorem (Brewer's Theorem). URL https://en.wikipedia.org/wiki/CAP_theorem. Retrieved 2016-11-15.

[46] Fritz Zwicky. *Discovery, Invention, Research — Through the Morphological Approach*. The Macmillian Company, Toronto, 1969.