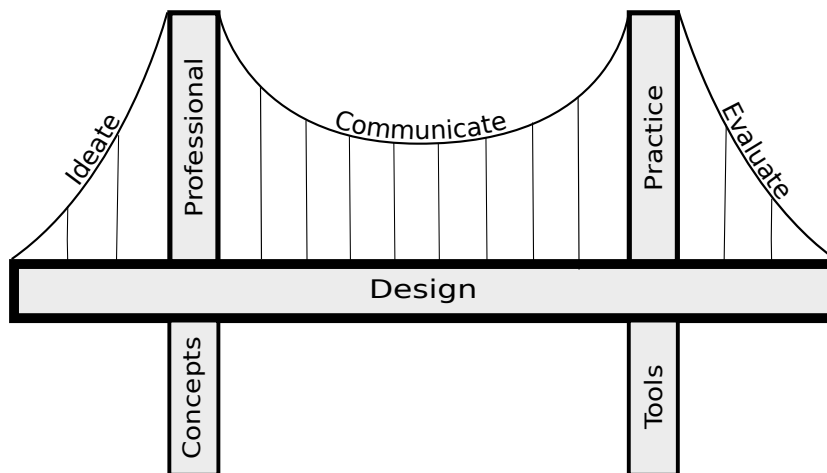


EDITED BY DEREK RAYSIDE

SE CAPSTONE DESIGN PROJECT HANDBOOK



UNIVERSITY OF WATERLOO

Copyright © 2024 by individual contributors.
Compiled March 31, 2024

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.

Licensed under Creative Commons Attribution-ShareAlike (CC BY-SA) version 2.5 or greater.
<http://creativecommons.org/licenses/by-sa/2.5/ca/>
<http://creativecommons.org/licenses/by-sa/3.0/>

Contents

1	Learning Objectives	11
1.1	Course Calendar Descriptions	12
1.2	CEAB Capstone Learning Objectives	13
1.3	CIPS Capstone Learning Objectives	15
1.4	SE Curriculum Committee Intended Graduate Attributes	17
1.5	Definitions of Engineering & Software Engineering . . .	22
1.6	Modes of Assessment: Formative & Summative	29
1.7	Modes of Instruction: Didactic & Dialectic	29
2	Project Selection	31
2.1	Problems and Opportunities	32
2.2	Kinds of Projects	33
2.3	Individual Learning Objectives	34
2.4	Team Formation	35
2.5	Activities to Assist with Project Selection	35
2.6	Changing Projects	36
2.7	Changing Teams	41
3	Teamwork Activities	43
3.1	Identify Individual Learning Objectives & Skills	43
3.2	Write a Team Working Agreement	44
3.3	Communicate about Communication	47
3.4	Host a Retrospective Meeting	48
3.5	Practice Backup Behaviour (Supporting Teammates) . . .	48
3.6	Practice Active Listening	49
3.7	Practice Assertive Communication: DESC	49
3.8	Apply Two Techniques to Manage Contagious Emotions	50
3.9	Apply Team Formation Strategies	51
3.10	Identify and Resolve Your Teams Dysfunctions	52
3.11	Conflict: Introduction	53
3.12	Conflict: Difficult Behaviours	54
3.13	Conflict: Five Handling Modes	55
3.14	Conflict: Reflection Questions	56
3.15	Conflict: Situation Assessment	57

3.16	Conflict: Personality-Based Coping Strategies	59	
3.17	Health: Balsom's 9 Attributes of Effective Teams	61	
3.18	Health: Google	61	
3.19	Health: Team Barometer	62	
3.20	Health: TeamRetro	64	
3.21	Health: Spotify	65	
3.22	Process Assessment: Joel Test	66	
3.23	Process Assessment: Scrum Checklist	67	
3.24	Process Assessment: CMMI	68	
3.25	Process Assessment: Capability Immaturity Model	68	
3.26	Process Assessment: UX Maturity	69	
3.27	Course: PD4 Teamwork	69	
3.28	Course: INTEG210 Making Collaboration Work	69	
4	Creative Activities	71	
4.1	Modes of Creative Thinking: Intense and Casual	71	
4.2	Apply SCAMPER	72	
4.3	Apply C-K Theory	72	
4.4	Use Comparison to Generate New Ideas	72	
4.5	Design Space Exploration	75	
4.6	Brainstorming	77	
4.7	6-3-5 Group Brainstorming	78	
4.8	Crazy 8s (a form of group brainstorming)	78	
4.9	Think / Pair / Share	78	
4.10	Six Thinking Hats	79	
5	Planning Activities	81	
5.1	Exploring Early Can Be A Good Strategy	81	
5.2	Select Project Success Metrics	81	
5.3	Weekly Work Intensity	82	
5.4	Plan to Prototype	83	
5.5	Old Stories Told in New Ways	88	
5.6	Compare to Popular Project Ideas	90	
6	Conceptual Activities	95	
6.1	Problem Identification and Refinement	95	
6.2	Identify the Core Conceptual Data Structure	99	
6.3	Position in Marketspace	99	
6.4	Strategic Project Positioning	100	
6.5	Understanding Project Risk	103	
6.6	OLD: How to Choose a Project	108	
6.7	Position in a Conceptual Framework	108	
6.8	Write a Research Literature Report	109	
6.9	Apply Rules of Thumb	109	
6.10	Position in Normal vs Radical Design	112	

6.11	Apply an Idea from the Project Domain	115
6.12	Apply Cognitive Bias Understanding	116
7	Requirements Activities	117
7.1	Domain Model	117
7.2	Use Cases & Scenarios	117
7.3	User Manual	117
7.4	Lean Canvas	117
7.5	Hypothesis Testing	118
7.6	Identify User’s Emotional Objectives	118
7.7	Practice Decoding Analogies/Metaphors	118
8	Design Activities	121
8.1	Describe Your Architecture	121
8.2	Extract & Analyze Your Architecture	122
8.3	Apply Formal Methods	123
8.4	Apply UI Design Guidelines	123
8.5	Incorporate Privacy by Design	123
8.6	Peer Design Exploration	124
8.7	Peer Design Review	124
8.8	Select a Database Technology	125
8.9	Apply (or Reject) the UNIX Design Philosophy	125
8.10	Read a Book on Approaches to Software Design	126
9	Testing Activities	127
9.1	Assess Testing	127
9.2	Create More Manual Tests	127
9.3	Identify Invariants	127
9.4	Identify Mathematical Properties	127
9.5	Use Automated Test Input Generation Tools	128
9.6	Use A Linter	128
9.7	Test Against an Alternative Implementation	128
9.8	Set Up Continuous Integration	129
9.9	Set Up Deployment Environments	129
9.10	Statistical Cross-Validation for Machine Learning	129
9.11	Performance Profiling	129
9.12	Scalability Assessment and Planning	129
9.13	Measure Precision and Recall	130
10	User-Centred Design (from cs449)	131
10.1	Value Proposition	131
10.2	Persona Empathy Map	132
10.3	Gather Data	134
10.4	Analyze Data	134
10.5	Crazy 8s	135

10.6	Low-Fidelity Prototyping	135
10.7	High-Fidelity Prototyping	138
11	User Activities	139
11.1	Take TCPS2 Training: Ethical Conduct with Users	139
11.2	Do a User Activity	139
11.3	Ideate about Possible User Activities that You Might Do	139
11.4	Formative <i>vs.</i> Summative Evaluations	140
11.5	Rohrer Survey of 20 Different User Activities	141
11.6	Farrell's Survey of 34 User Activity Methods	142
11.7	Moran's Survey of 9 Quantitative User Activities	143
11.8	Nielsen's 10 Usability Rules of Thumb	143
11.9	Why Testing With 5 Users is Usually Enough	143
11.10	27 Tips for Conducting Successful User Research in the Field	143
11.11	Levels of ux Design Maturity	144
11.12	Apply Universal Design for Accessibility	145
11.13	Peer Usability Review	145
11.14	User Acquisition	146
12	Reflective Activities	147
12.1	Watch Past Project Presentations	147
12.2	Analyze Past Project Awards	147
12.3	Index Past Projects	147
12.4	Read Past Project Reports	147
12.5	Read Turing Award Speeches	147
12.6	Read Video Game History: The Digital Antiquarian	148
12.7	Watch ACM Tech Talks	148
12.8	Read ACM Queue Articles	148
12.9	Read Classic SE Papers	148
12.10	Watch a Documentary	148
12.11	Read a Book	149
12.12	Assess Your Choice of Learning Activities	149
12.13	$n + 1$ Cohort Feedback (Retrospective)	149
13	Communication Activities	151
13.1	Read Edward Tufte's Presentation Advice	152
13.2	Read Trees, Maps, and Theorems Presentation Advice	152
13.3	Watch Patrick Henry Winston's Presentation Advice	152
13.4	Learn from TED Presentation Advice	152
13.5	Learn from Nancy Duarte's Presentation Advice	153
13.6	Conquer Your Fear of Public Speaking	153
13.7	Choose a Narrative Structure for Your Presentation	154
13.8	Revise Your Writing	155
13.9	Revise Your Abstract	156
13.10	Read Authors You Want to Emulate	158

13.11	Write for Accessibility and ESL	159
14	Project Evaluation	161
14.1	Project Status Sheets	162
14.2	Results (An Aspect of Validation)	171
14.3	Teamwork Assessment	178
14.4	Communication Assessment	178
14.5	How Referee Grades Get Combined	178
14.6	Referee Selection	180
14.7	Risk Reward (Bonus)	182
15	Intellectual Property & Collaborators	185
15.1	What is Intellectual Property?	185
15.2	Collaborating on Free / Open Source Software	186
15.3	Collaborating with an Established Corporation	186
15.4	Collaborating with a Startup	187
15.5	The Research Licensing Approach	190
15.6	Intellectual Property & Standards Report	190
16	Bibliography	195

List of Figures

1.1	Suspension bridge concept map for sE Capstone Design Project	11
2.1	Project inputs and outputs.	31
3.1	Phases of team development	51
3.2	Lencioni's pyramid of the five dysfunctions of a team . .	52
3.3	Five conflict-handling modes	55
3.4	Scrum checklist by Henrik Kniberg	67
4.1	Some kinds of comparisons	73
4.2	The traditional square of opposition.	73
5.1	When to build which kind of prototype: experimental, evolutionary, operational	86
5.2	Icons for different kinds of prototypes.	87
5.3	The story of the seL4 verified microkernel	87
6.1	The space between Normal and Radical design in terms of components and composition	114
6.2	Maslow's hierarchy of human needs	115
10.1	Example of what a persona looks like	132
10.2	Example of what an empathy map looks like	133
10.3	Image of affinity diagram	136
10.4	Image of Crazy 8	137
10.5	Image of sketch	137
11.1	Central figure from Rohrer's user-activity survey article for the Nielsen/Norman Group.	141
11.2	Central figure from Farrell's user-activity survey article for the Nielsen/Norman Group.	142
13.1	Freytag's Pyramid	154
14.1	Marketing plan for food ordering app	173

Back where I come from we have universities — seats of great learning — where [people] 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

Learning Objectives

Capstone projects are a common component of engineering and computer science undergraduate degrees. They are intended to provide an opportunity to apply what has been learned across all (or most of) the prior courses in a holistic way. Consequently, there are many learning objectives.

The Canadian Engineering Accreditation Board (CEAB) identifies 12 learning objectives for capstone projects. The Canadian Information Processing Society (CIPS) identifies 14 learning objectives for capstone projects. The UWaterloo SE Curriculum Committee identifies 111 attributes, in 19 categories, that graduates should have. The committee has mapped those graduate attributes to every course in the core curriculum: 94 of these 111 intended graduate attributes (85%) are mapped to the capstone courses.

Figure 1.1 attempts to summarize these learning objectives using the visual metaphor of a bridge — a classic engineering artifact. ‘Hard’ skills are illustrated with the compressive members of the bridge. ‘Soft’ skills are illustrated with the tensile members of the bridge. The road deck — *design* — requires both kinds of skills for solid support.

§1.2 CEAB Capstone Learning Objectives

§1.3 CIPS Capstone Learning Objectives

§1.4 SE Capstone Learning Objectives

This kind of visual metaphor is sometimes called a *concept map*.

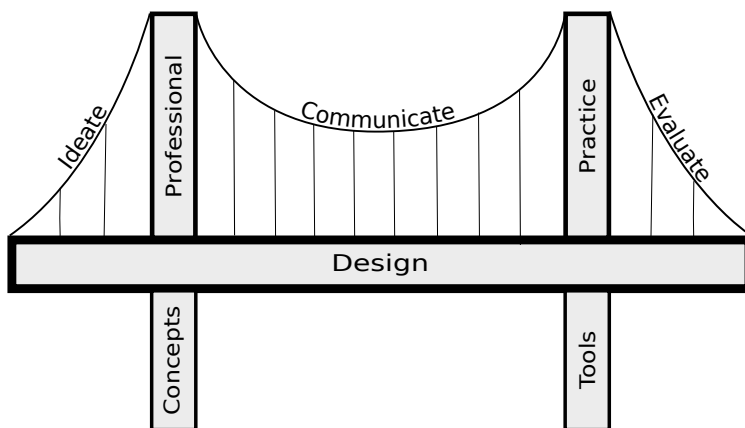


Figure 1.1: Suspension bridge concept map for SE Capstone Design Project. Design is supported by both compressive and tensile members, representing ‘hard’ skills and ‘soft’ skills, respectively.

The capstone project is an opportunity to explore a significant subset of the learning objectives of the undergraduate degree in an integrated project. In broad overview, for capstone courses we might say the learning objectives are:

- To develop *technical leadership and judgement*.
- To do a project of *enduring value*, that has meaning to someone (including yourself) after you graduate.
- To develop *broad familiarity* with a wide range of techniques, tools, skills, concepts, and professional practices. This additional breadth of learning, beyond the courses you have taken and your prior experiences, will come through through the exploration of your project, your project mentors, your peers' feedback on your project, your observation of your peers' projects, and this handbook.
- To be able to *select and apply appropriate* techniques, tools, skills, concepts, and professional practices for a particular project. Both for your project, and projects of your peers that you provide feedback on.
- To *communicate* in a clear, correct, complete, and concise manner.
- To *give and receive constructive feedback*, both within your team and with other teams.
- To work in a *team*.
- To demonstrate appropriate software engineering *process maturity*. Version control, it goes without saying, is required. This handbook describes various industry scales of process maturity.
- To understand and be able to apply a variety of definitions of engineering and software engineering.

1.1 Course Calendar Descriptions

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.

SE490

Continuing from SE390, students undertake a substantial customer-driven group project. Project groups establish and maintain project

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.

control processes, delivering a series of iterations on their SE390 prototype. Adaptive methods are encouraged and supported.

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.

1.2 CEAB Capstone Learning Objectives

The Canadian Engineering Accreditation Board (CEAB) is a standing committee of Engineers Canada¹ that accredits undergraduate engineering programs in Canada. A number of the requirements in CEAB's Accreditation Criteria and Procedures² 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.
- 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.

¹ Engineers Canada is a national organization of the 12 provincial and territorial associations that regulate engineering and license engineers in Canada.

² 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

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.

1.3 CIPS Capstone Learning Objectives

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 Criteria³ 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.

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,

³ 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

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”.

1.4 SE Curriculum Committee Intended Graduate Attributes

The UWaterloo 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. The capstone project courses cover almost all of graduate attributes identified by the SE] curriculum committee, and are listed here (including the number of graduate attributes covered in each category).

KNOWLEDGE BASE

3/7

- Understand software systems: a) operating systems, b) software security, c) networking, d) distributed systems, e) database design and use, f) human factors.
- Apply theory and practice of software programs: a) procedural, object-oriented and functional coding, b) translation, c) machine execution, d) exception control flow, e) concurrent control flow.
- Apply discrete mathematics to software development: a) set theory, b) combinatorics, c) graphs and trees, d) discrete probability, e) propositional and predicate logic, f) direct, contradiction, and inductive proofs, g) boolean logic, h) grammars, i) finite-state automata.

INVESTIGATION

3/3

- 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.
- Analyze succinct, non-obvious task specifications to interpret goals and identify relevant research materials.
- Apply independent research to complement course materials.

PROBLEM ANALYSIS

10/10

- Determine (Analyze) the problem to be solved.
- Elicit (Create) or invent behavioural and non-behavioural requirements of complex problems.
- Identify and eliminate by negotiation (Evaluate) conflicts among requirements.
- Rank (Evaluate) requirements by priority.
- Evaluate risks, i.e., identify requirements for detecting, avoiding, and mitigating hazards.
- Determine (Create) appropriate algorithms and data structures to solve the problem at hand according to the elicited or invented requirements.
- Evaluate correctness, consistency, completeness, reliability, and availability of requirements.
- Evaluate project duration and costs from requirements.
- Create test cases and test harnesses from requirements.
- Analyze and create user interfaces that adhere to sound human-computer interface principles.

SPECIFICATION

3/3

- Evaluate project duration and costs from requirements specifications.
- Document (Apply) behavioural and non-behavioural requirements using formal and informal notations, including natural language prose, to produce a requirements specification.
- Evaluate correctness, consistency, and completeness of requirements specifications.

DESIGN

8/8

- Understand the software architecture and design process and their models.
- Apply informal and formal design representations.
- Apply design patterns, reference architectures, and design plans to design non-trivial software systems.
- Determine (Create) appropriate algorithms to solve problems.
- Create (Create) alternatives that explore the design space.
- 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.
- Select (Evaluate) from alternatives by evaluating multiple objectives and trade-off analysis.
- Create experimental and evolutionary prototypes.

PROGRAMMING TECHNOLOGY

6/6

- Understand several programming languages.
- Understand several scripting languages.
- Understand several shell languages.
- Apply at least one a) scripting language, b) shell language, c) HTML.
- Apply at least one editor.
- Apply at least one debugging tool.

IMPLEMENTATION

4/7

- Understand programming in the large.
- Analyze code in many programming languages for understanding.
- Analyze a specification to determine what must be implemented.
- Create algorithms expressed in some programming language.

VERIFICATION & VALIDATION

9/12

- Create test cases and test harnesses from requirements specifications.
- Evaluate the extent to which programs satisfy their specifications.
- Investigate (Create) program behaviours to generate working hypotheses about defects.
- Confirm (Evaluate) hypotheses about root causes of defects.
- Resolve (Create) defects.
- Confirm (Analyze) validity of resolutions of defects.
- Evaluate correctness, consistency, completeness, reliability, and availability of designs.
- Evaluate correctness, consistency, completeness, reliability, and availability of code.
- Evaluate correctness, consistency, completeness, reliability, and availability of test cases.

MAINTENANCE 3/4

- Apply common software maintenance processes and techniques.
- Report and correct (Analyze) defects in a system.
- Enhance (Create) functionality in a system.

INDIVIDUAL WORK 5/6

- Know (Analyze) own productivity rates to be able to make accurate time estimates for own work.
- Manage (Apply) own time.
- Assume (Apply) responsibility for own work.
- Assimilate (Evaluate) constructive criticism.
- Function (Evaluate) effectively as an individual in a team.

TEAM WORK 4/4

- Apply conflict resolution strategies.
- Actively listen (Evaluate) and seek expertise of others.
- Interact (Apply) with stakeholders.
- Interact (Apply) effectively with people in non-software disciplines.

COMMUNICATION SKILLS 5/5

- Apply grammar, spelling, and style appropriate to written technical communication.
- Use (Apply) clear and logical organization in written or oral technical communication.
- Use (Apply) figures and tables effectively in written or oral technical communication.
- Use (Apply) rhetoric to inform and justify in written or oral technical communication.
- Make (Create) effective oral technical presentations.

ECONOMICS 5/6

- Understand economics of software projects: a) time-value of money, b) technical debt, c) cost-benefit analysis, d) break-even analysis.
- Understand cost to fix defects as a function of development lifecycle stage.
- Understand when defects are introduced to systems under development.
- Understand Brooks's law of development team size.
- Understand cost-estimation techniques, function points and Cocomo to software development projects.

PROJECT MANAGEMENT 4/6

- Evaluate project duration and costs from requirements specification.
- Make decisions (Evaluate) under uncertainty, including project risks.
- Evaluate software reliability and availability.
- Create effective business plans.

TOOLS

5/5

- Select (Apply) appropriate tools.
- Configure (Apply) and deploy tools.
- Analyze (Analyze) and interpret tool results for correctness and completeness.
- Apply configuration management tools for non-trivial software projects.
- Apply management tools to software project schedules and deliverables.

PROFESSIONALISM

7/7

- Understand the role of professional licensing and regulation to protect the public good.
- Understand the relevance of diversity and equity in engineering practice.
- Remember professional societies relevant to software engineering.
- Remember software standards organizations.
- Understand intellectual property rights with regard to software, e.g., copyright, utility patents, design patents, trade marks, license agreements, trade secrets.
- Apply appropriate knowledge resources.
- Apply relevant software standards and best practices.

IMPACT OF ENGINEERING ON SOCIETY AND THE ENVIRONMENT

3/5

- 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.

ETHICS AND EQUITY

3/3

- Understand the relevance of diversity and equity in engineering practice.
- Understand intellectual property rights with regard to software, e.g., copyright, utility patents, design patents, trade marks, license agreements, trade secrets.
- Apply PEO's Code of Ethics.

LIFELONG LEARNING

4/4

- Remember professional and technical societies relevant to software engineering.
- Evaluate information for authority, currency, and objectivity.
- Evaluate knowledge gaps and learning needs.
- Find and apply (Apply) appropriate knowledge resources and best practice guidelines.

1.5 Definitions of Engineering & Software Engineering

Definitions, by their nature, are tools of inclusion and exclusion, of similarity and differentiation. This collection of definitions explores the following general intellectual threads:

- differentiating software engineering from computer science, software development, programming, solo programming, *etc.*
- concerns of the public interest and public safety in relation to engineering practice and artifacts
- generalizing the concept of engineering to ways of knowing and working, which can then be applied in various domains
- exploring the connection of software engineering with the physical world, either through embedded systems of software for automating traditional engineering disciplines

ToDo: Consider your project or project concept in relation to the following definitions of engineering and software engineering:

1. Does your project align with the definition? Most projects will align with some definitions but not with others. Make two lists, one for those that align with your project and one for those that don't.
2. For the definitions your project aligns with, how will you produce evidence that your project is aligned with the definition?

1.5.1 Engineers Canada definition of Engineering

Engineers Canada is an umbrella organization that makes recommendations to the provincial regulators (*e.g.*, Professional Engineers Ontario (PEO)), as well as setting the academic criteria for university engineering programs. Their recommended definition for the practice of engineering is:

The practice of engineering means any act of planning, designing, composing, evaluating, advising, reporting, directing or supervising, or managing any of the foregoing, that requires the application of engineering principles, and that concerns the safeguarding of life, health, property, economic interests, the public welfare or the environment.

The Definition extends to include certain areas that are sometimes considered peripheral to engineering, such as teaching engineers or engineering students, supervising engineers, engineering sales, or certain computer applications to engineering works. The issue of protecting the public interest and the question of whether the public is at risk must be considered in the broadest terms. The component, product, device, system, process, etc. that is the outcome of the engineering undertaking must be viewed from its broader societal perspective - the

<https://engineerscanada.ca/public-guideline-on-the-practice-of-engineering-in-canada>

judgement of the engineer's employer or client, or the engineer, are not necessarily adequate.

1.5.2 *The Professional Engineers Act of Ontario*

Ontario provincial law defines the practice of professional engineering as follows. Every province and territory in Canada has provincial law that defines the practice of professional engineering. Most of them are similar to the Engineers Canada definition, which is the model that Ontario follows. A consequence of these laws, for example, is that highways need to be designed by licensed civil engineers.

practice of professional engineering means any act of planning, designing, composing, evaluating, advising, reporting, directing or supervising that requires the application of engineering principles and concerns the safeguarding of life, health, property, economic interests, the public welfare or the environment, or the managing of any such act; (*exercice de la profession d'ingénieur*)

Ontario's Professional Engineers Act has some specific exclusions of kinds of engineering work that may be done by groups of people other than engineers. For example, there is a detailed section about engineering work that is permitted to be done by architects. Similarly, there are exclusions for tool-and-die work and machinists.

1.5.3 *Engineering is Math/Science + Commerce*

Mary Shaw is a distinguished software engineering researcher at the Software Engineering Institute (SEI) at Carnegie Mellon University (CMU). She has studied the histories of civil and chemical engineering to better understand what an engineering discipline is and how it evolves.

Through historical study of the evolution of civil and chemical engineering, Shaw has developed a three-stage model for the maturation of a field into a complete engineering discipline. She has shown that an engineering discipline begins with a craft stage, characterized by the use of intuition and casually learned techniques by talented amateurs; it then proceeds through a commercial stage, in which large-scale manufacturing relies on skilled craftsmen using established techniques that are refined over time. Finally, as a scientific basis for the discipline emerges, the third stage evolves, in which educated professionals using analysis and theory create new applications and specialties and embody the knowledge of the discipline in treatises and handbooks. Shaw has concluded that contemporary software engineering lies somewhere between the craft and commercial stages, and this conclusion has led to an effort on her part first to promote an understanding of where software engineering should be headed and second to develop the scientific understanding needed to move the discipline into the third stage.

<https://www.ontario.ca/laws/statute/90p28>

She is a recipient of the US National Medal of Technology, amongst other honours.

<https://www.encyclopedia.com/people/literature-and-arts/theater-biographies/mary-shaw>

One unique aspect of Shaw’s historical approach is the focus on *commerce*. This deserves some discussion, especially in the context of capstone projects. Capstone projects are typically *non-commercial* in the sense that they are not done for hire, do not generate revenue, *etc.* This notion of commerce isn’t exactly what Shaw is talking about.

Shaw’s concept of *commerce* is primarily about aspects such as repeatability of the process, reliability of the results, and the skill level of the practitioners. For example, the Apollo space program to land on the moon would not meet her concept of *commerce*: it was extremely high risk in every way, it required extraordinarily high skill levels from all participants, and it was not really repeatable.

By contrast, the aircraft industry meets Shaw’s notion of commerce: there are multiple companies competing in the market; they each make multiple instances of multiple designs; all of the products are very reliable; the engineers need a high level of skill, but not necessarily virtuosic. There is a process in place that ensures safe, reliable, and manufacturable design outputs from ordinary engineers (or from brilliant engineers on a bad day).

Walter Vincenti⁴ has a technical concept that aligns well with Shaw’s historical concept: *normal design* versus *radical design*. Normal design uses normal components arranged in normal ways. There is still innovation in normal design. Products get better, faster, smaller (or bigger), more powerful, more efficient, lower cost, *etc.* But the overall concepts are known. For example, cars have new models every year that are typically refinements of the previous year’s model.

Radical design, by contrast, involves inventing new components or arranging existing components in new ways. It hasn’t been tried before. There is often less confidence in reliability.

When one focuses on the intellectual activities of engineers, as in many of the other definitions of engineering, it can be tempting to start to think that only radical design is truly engineering, since the application of math and science are more romantically obvious in radical design. When focusing on the application of math and science as the sole criteria for engineering, one is often trying to distinguish engineering from the work done by technicians, machinists, *etc.* Shaw’s historical studies remind us that engineering is in a middle ground between technologists and extreme radical design.

1.5.4 Origins of ‘Software Engineering’

The term software engineering originated at some conferences sponsored by NATO in the late 1960s. The most common reference is:

P. Naur and B. Randell, eds., *Software Engineering: Report on a Conference Sponsored by the NATO Science Committee*, (7–11 October

The Romans meet Shaw’s concept of commerce: they built roads, bridges, aqueducts, *etc.*, at massive scale and quality. But they weren’t engineers as they lacked math and science.

The Wright Brothers’ would not meet Shaw’s notion of engineering. They were doing radical design, applying a new understanding of aerodynamics in a way nobody had done before. The results were not (yet) reliable, repeatable, manufacturable, *etc.*

⁴Walter G. Vincenti. *What Engineers Know and How They Know It: Analytical Studies from Aeronautical History*. The Johns Hopkins University Press, 1993

1968), Scientific Affairs Division, NATO, 1969.

The term is said to have been coined by Margaret Hamilton:

U.S. developer Margaret Hamilton is widely credited with coining the term *software engineer* after working on the on-board flight software for the U.S. National Aeronautics and Space Administration's (NASA) Apollo program in the 1960s. She said her team's work should be taken as seriously as other engineering and scientific disciplines. Many agreed.

<https://www.theglobeandmail.com/business/technology/article-is-a-software-engineer-an-engineer-alberta-regulat>

1.5.5 Canadian national occupational classification of SE

Government of Canada National Occupational Classification says:

Software engineers and designers research, design, evaluate, integrate and maintain software applications, technical environments, operating systems, embedded software, information warehouses and telecommunications software. They are employed in information technology consulting firms, information technology research and development firms, and information technology units throughout the private and public sectors, or they may be self-employed.

<https://noc.esdc.gc.ca/Structure/NocProfile?objectid=s%2B18U2GgCu7IJq7TKb3GrKyNg8uVSExHroQyk7WAV4%3D>

1.5.6 IEEE: SE is the application of engineering to software

There are several definitions along the lines of *software engineering is the engineering of software*, including from the IEEE and Engineers Canada. While this sounds both tautological, the idea is that *engineering* is an organized system of knowledge and an approach to doing things. This approach might have originally been developed in the traditional engineering disciplines, but it can be generalized and applied in other areas.

The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software. [IEEE 2010]

<https://www.acm.org/binaries/content/assets/education/se2014.pdf>

Walter Vincenti⁵ contributed to a philosophical foundation for this viewpoint that engineering is a way of doing things that asks certain kinds of questions and creates certain kinds of knowledge. He argues that there is an engineering method (variation-selection) as distinct from the scientific method. Engineering involves the application of science, but engineering also generates new questions for scientists to investigate. Engineers sometimes work in areas that scientists do not yet understand. Engineers take measurements to characterize the behaviour of phenomena, and base predictions and designs on those measurements — even when scientists do not yet understand why those phenomena occur. Engineers want to make practical predictions about behaviours, but do not necessarily need to know why

⁵ Walter G. Vincenti. *What Engineers Know and How They Know It: Analytical Studies from Aeronautical History*. The Johns Hopkins University Press, 1993
https://en.wikipedia.org/wiki/What_Engineers_Know_and_How_They_Know_It

those behaviours occur. Scientists want to investigate why behaviours occur, but cannot necessarily make practical predictions about behaviours. Scientists are sometimes focused on smaller scales and mechanisms, whereas engineers are often concerned with behaviours in aggregate and at scale.

1.5.7 Engineers Canada definition of Software Engineering

Engineers Canada has a position paper on professional practice in software engineering, which is primarily concerned with the questions of when regulation and licensing are required. It builds on both their general definition of engineering and the IEEE definition of software engineering.

Software engineering is a discipline that has proven challenging for the engineering profession to recognize and, therefore, regulate. For the purposes of regulation and enforcement, the scope of software engineering is consistent with the existing definition of engineering provided in *National guideline on the practice of engineering in Canada*, which states:

The *practice of engineering* means any act of planning, designing, composing, evaluating, advising, reporting, directing or supervising, or managing any of the foregoing, that requires the application of engineering principles, and that concerns the safeguarding of life, health, property, economic interests, the public welfare or the environment.

In the case of software engineering, a piece of software (or a software intensive system) can therefore be considered an *engineering work* if **both** of the following conditions are true:

- The development [4] of the software required *the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software* [5].
- There is a reasonable expectation that failure or inappropriate functioning of the system would result in harm to life, health, property, economic interests, the public welfare or the environment.

1.5.8 PEO's definition of Software Engineering

PEO Council (Professional Engineers Ontario) approved the following proposed working definition of software engineering:

Software engineering is deemed to fall within the practice of professional engineering:

- Where the software is used in a product that already falls within the practice of engineering (e.g. elevator controls, nuclear reactor controls, medical equipment such as gamma-ray cameras, etc.);
- Where the use of the software poses a risk to life, health, property or the public welfare; and

<https://engineerscanada.ca/engineers-canada-paper-on-professional-practice-in-software-engineering>

[4] In software engineering, the term *development* refers to the full product lifecycle including design, implementation, testing and sometimes installation and maintenance (fault repair and feature enhancement), as per *Institute of Electrical and Electronics Engineers, Guide to the Software Engineering Body of Knowledge (SWEBOK)*, 2004. <http://www.computer.org/portal/web/swbok/2004guide>. Within this document, *development* should be understood to comprise this full range of activities.

[5] Institute of Electrical and Electronic Engineers, *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std 610.12-1990, New York, NY. <http://standards.ieee.org/findstds/standard/610.12-1990.html>

<https://www.peo.on.ca/public-protection/complaints-and-illegal-practice/report-unlicensed-individuals-or-companies-2#software>

- Where the design or analysis requires the application of engineering principles within the program (e.g. does engineering calculations), meets a requirement of engineering practice (e.g. a fail-safe system), or requires the application of the principles of engineering in its development.

1.5.9 SE is Embedded Systems

PEO's *External Groups Task Force – Software* wrote a report in 2002 that basically took the position that software engineering is embedded safety-critical systems. This report was prepared in consultation with the *Canadian Information Processing Society (CIPS)*. CIPS has an *Information Systems Professional (ISP)* designation that has legal recognition in six provinces including Ontario.

This PEO task force was trying to answer a regulatory question: when does the *Professional Engineers Act of Ontario* require work to be done by a licensed software engineer versus an ISP? Their answer is that a software engineering license should be required to work in safety-critical embedded systems: automobiles, elevators, nuclear, *etc.* The argument is that a practitioner with an ISP certification might not have appropriate background in the physical sciences, whereas a licensed engineer would have this education.

An alternative recommendation that this task force could have made would have been to amend the *Professional Engineers Act of Ontario* to have an exclusion for ISPs to do some engineering work — as is already done for architects, machinists, *etc.* They could have kept the definition of software engineering broader, and then defined that the subset of software engineering that does not involve safety-critical embedded systems may be done by ISP practitioners.

1.5.10 SE is automation of traditional engineering

The *Association of Professional Engineers and Geoscientists of Alberta (APEGGA)* wrote a position paper in 2006 on *Guidelines for Professional Responsibility in Developing Software*. This paper doesn't go so far as to define software engineering, but it does narrow the scope of APEGGA's regulatory concern to "engineering, geological, and geophysical software." That is, software to be used by engineers in traditional disciplines. In other words, software engineering is the automation of traditional engineering disciplines via software.

There is a need to provide proactive means to defend public interest, safety, and security as it may be affected by software failure. This guideline sidesteps the *software engineering versus computer science* debate by narrowing the focus to the professional responsibilities of APEGGA members in developing and using engineering, geological, and

PEO External Groups Task Force, *Report to PEO Council on Software Practice*, 2002. Task force chair Peter DeVita.

<https://cips.ca/certification>

See Figure 3 on page 31 of the task force's report for how they segment software work. Safety-critical embedded systems are the top right segment.

<https://www.apega.ca/docs/default-source/pdfs/standards-guidelines/software.pdf>

geophysical software in Alberta. This guideline reinforces our regulatory jurisdiction while pragmatically protecting the public interest in this area.

1.5.11 *SE is Teamwork*

David Parnas and Brian Randall proposed, back around the origin times of the term software engineering (i.e., around 1970), the following:

Software engineering is multi-person development of multi-version programs.

Their motivation here was to distinguish software engineering from *solo programming* (their phrase). This aligns with the folk-wisdom that anything small enough to be done by one person isn't engineering. Mary Shaw puts this view on a more solid historical foundation with the concept of *commerce*.

Elsewhere, and in much of his work, Parnas focused on the *software engineering is the engineering of software* kind of definition, with particular emphasis on the application of math and science.

1.5.12 *SE is Programming Integrated over Time*

The book *Software Engineering at Google* (2020) introduced a new definition:

Software engineering is programming integrated over time.

They go on to elaborate as follows:

We can also say that software engineering is different from programming in terms of the complexity of decisions that need to be made and their stakes. In software engineering, we are regularly forced to evaluate the trade-offs between several paths forward, sometimes with high stakes and often with imperfect value metrics. . . . With those inputs in mind, evaluate your trade-offs and make rational decisions.

This is another definition intended to distinguish software engineering from other forms of software development. This focuses on the commerce aspect of engineering: the major cost centre of software is maintenance, not initial development. The key emphasis here is on managing the costs of system performance and of engineering labour over time — research has consistently shown that maintenance is the most expensive phase of the software lifecycle.

This definition implicitly builds on the concepts of the IEEE definition: it assumes that engineering is a systematic way of knowing and doing (like the IEEE definition), and it focuses that energy on the most expensive part of the lifecycle.

<https://www.oreilly.com/library/view/software-engineering-at/9781492082781/>

The authors of this book have also distributed PDF copies online.

1.6 Modes of Assessment: Formative & Summative

FORMATIVE ASSESSMENT: qualitative feedback to identify and help resolve misconceptions, struggles, and learning gaps. Helps student to achieve intended learning outcomes and prepare for future summative assessments.

SUMMATIVE ASSESSMENT: measure the degree of success demonstrated with the learning. Often quantitative. Tests and exams are the classic examples. Project presentations and demonstrations are typical summative assessments in capstone courses.

1.7 Modes of Instruction: Didactic & Dialectic

DIDACTIC is the mode of instruction and learning for most engineering / science / math courses, and is characterized by:

- regularly scheduled lectures
- textbooks
- assignments / problem sets
- tests / exams
- most/all assessment is *summative*

DIALECTIC is the main mode of instruction and learning for capstone courses. It's about learning through discussion and dialogue: with teammates, with peers, with instructors, with ideas and texts.

- interaction and discussion
- self/team-directed learning
- broad reference materials
- presentations + Q&A
- much assessment is *formative*

In HCI we also use the concepts of *formative* and *summative* assessment of the software. For example, a *thinkaloud* is a formative technique where we learn about the software by listening to a user express their thoughts out loud as they are using the software. This gives us qualitative insights into how to improve the software.

A *summative* assessment of the software might measure the time it takes the user to do a task, or the user's task error-rate while using the software. This gives us a quantitative measure of how well the software works, but doesn't give us any insights into how to improve the software.

Didactic: a teacher-centered approach in which the teacher talks and students learn by listening.

Dialectic: discussion and reasoning by dialogue as a method of intellectual investigation. Everyone takes turns talking and listening. Students learn by talking, by listening to their peers, and by listening to the instructor.

Project Selection

Your capstone project is probably the largest project that you will undertake during your studies. It's important to choose wisely. There are several factors you might consider, including:

- your interests (*passion*)
- your personal *learning* objectives
- having an interesting technical *problem* to work on
- find a *solution* to that problem
- something that someone else might be interested in (*opportunity*)
- recognition that someone else is interested (*results*)

We can organize these considerations in two dimensions as shown in Figure 2.1, as *inputs* or *outputs*, and also from in terms of the number of people concerned: from individual (you), to other engineers (starting with your teammates), and outwards to society at large (your users or other stakeholders).

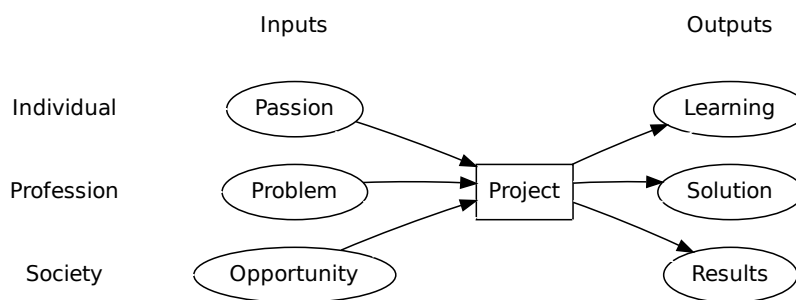


Figure 2.1: Project selection considerations categorized in two dimensions: inputs vs. outputs; and internal to external.

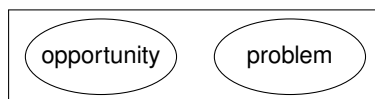
2.1 Problems and Opportunities

There is a famous quote about problems and opportunities:

Results are obtained by exploiting opportunities, not by solving problems.
— Peter F. Drucker

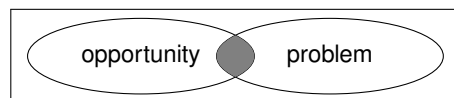
What does Drucker mean by this? There are a variety of ways that one could interpret the words ‘problem’ and ‘opportunity’. From our perspective as engineers, we are going to interpret ‘problem’ to mean a technical challenge: something defined by the current state of technology, and potential future improvements. For example, making internal-combustion-engine vehicles more fuel efficient.

By contrast, we will understand ‘opportunity’ to denote something defined by society at large — something defined outside of the engineering profession. What are society’s needs and wants? Given these conceptions of ‘problem’ and ‘opportunity’, we might understand Drucker to be saying that problems and opportunities are disjoint:



For example, social networking websites such as Yelp, Facebook, or LinkedIn did not solve a technical problem: they addressed a societal opportunity. By contrast, from our current point in history, it looks like trying to make internal-combustion-engine (ICE) vehicles more fuel efficient is solving a problem with no corresponding opportunity: many countries and cities across Europe and Asia have declared that they will ban the sale of ICE vehicles in the coming years. Major vehicle manufacturers in Europe, Asia — and North America (such as GM) have said that they will go all-electric or majority-electric in the near future. The market for ICE engines, regardless of their efficiency, is going to get smaller in the near future.

Are all problems and opportunities disjoint? No. Ideally we find interesting technical challenges that also address societal needs. Ideally we find projects that are in the intersection:



For example, Watt’s steam engine both solved a technical problem with Newcomen’s steam engine and responded to society’s need for mechanized power. Many people mistakenly believe that Watt invented the steam engine. In fact, Newcomen did. But Newcomen’s engine was so inefficient that it had limited uses. Watt’s breakthrough was to separate the heating and cooling of the steam into separate chambers: the hot chamber was always hot, and the cold chamber was always cold. Newcomen’s engine, by contrast, had a single chamber that went through a cycle of heating and cooling. Watt’s technical breakthrough enabled the Industrial Revolution, and is why his name is most commonly associated with steam engines.

Drucker has been described as ‘the founder of modern management.’ He is widely considered a visionary: for example, he coined the term ‘knowledge worker’ in 1959. Harvard Business Review writes highly of him: <https://hbr.org/2009/11/why-read-peter-drucker> <https://hbr.org/2016/10/why-peter-druckers-writing-still-feels-so-relevant>

Or, before them, Friendster, Orkut, MySpace, etc.

Society does not always know what it wants or needs. For example, an old story is that Henry Ford said that if he had asked people what they wanted, they would have said faster horses. <https://hbr.org/2011/08/henry-ford-never-said-the-fast>

2.2 Kinds of Projects

There are a variety of perspectives one could use to classify capstone projects. The perspective that we will use here, and elsewhere in this handbook, is to classify projects by the kind of *results* that they aim to produce. There are currently five categories:

FOSS: Contribute patches to an existing Free/Open-Source Software project. For example, write patches for the Rust compiler, or the Xen hypervisor, or the Habitica todo-list game, or the digital billboard software written by UW Science Computing.

Research: Collaborate with a professor on a research topic, with the aim of publishing a small paper with your results. This kind of project is a great choice if you are potentially interested in graduate school. Note that you can satisfy the teamwork component of capstone outside your class if you have significant interactions with the prof's research group.

Consultant: Write software for a specific external partner. For example, write a citizen-science bee-tracking app for the international non-profit Friends of The Earth; or collaborate with grad students from the UW School of Pharmacy to write an app for patients to track chemotherapy side-effects.

New Product: Create a new product. This might be, but does not have to be, a viable business. For example, UW Flow was a capstone project from SE2014; Dynalist.io was a capstone project from SE2017; and Tutturu.tv was a capstone project from SE2021.

Advanced Technology: Do something interesting with your technical skills, that typically combines knowledge and skills from multiple advanced technical elective courses. For example, build a distributed storage system, or develop a data auditing/cleaning system, or build a graphics tool that makes it easier for artists to generate sprites for video games.

See §14.2 for more discussion of these result categories.

You can help define new results categories! SE2020 defined the Advanced Technology category, for example.

Usually established FOSS projects have a list of features that they want help implementing — that is, the FOSS project defines the *opportunity*.

The prof will know more about potential *opportunities* than you do. Remember, an opportunity is something that someone else cares about. In this case, the prof will know what the research community might be interested in reading.

The external partner defines the *opportunity*.

You define the opportunity! That's exciting, but also a bit scary, and will require some work.

This kind of project is heavy on the technical *problem*. The *opportunity* here is typically that other engineers find the project interesting to hear about.

2.3 Individual Learning Objectives

Every project is different, but there are some common themes about how various kinds of projects most likely connect to the curriculum, and which capstone learning activities are likely to be most relevant.

<i>Kind of Project</i>	<i>Courses</i>	<i>Likely Learning Activities</i>
<i>Advanced Technology</i>	SE464 Design SE465 Testing ATES	Design Testing Literature Review Read Turing Awards
<i>Free/Open-Source Software</i>	SE464 Design SE465 Testing ATES	Design Testing Literature Review Read Turing Awards
<i>Research</i>	SE464 Design SE465 Testing ATES	Design Testing Literature Review Read Turing Awards
<i>Consultant</i>	SE463 Reqs SE464 Design SE465 Testing CS449 HCI ATES	User Requirements Intellectual Property
<i>New Product</i>	SE463 Reqs CS449 HCI BET*	User Requirements Strategy Market Position Problem Refinement Apply Domain Knowledge Privacy by Design Selected a Database Technology Intellectual Property

2.4 Team Formation

Teams can form in several ways, including:

- *Friends* get together to form a team. This has the advantage that you know each other, but the disadvantage that it can sometimes be challenging to be productive together.
- *Shared interests*: You can form a team with classmates who have shared interests with you. This takes a bit of effort initially to connect with them, but can be very productive and rewarding.
- *External collaborations*: It is possible, although uncommon, for your teammates to be outside of your cohort. There are several variations of this:
 - *GENE403+404*: These are the capstone courses for collaborating with students from other engineering programs.
 - *ECE498A+B*: These are the ECE capstone courses. If you want to work with a team of ECE students, then you can join their capstone courses instead.
 - *Student Design Team*: If you are part of a student design team, such as Midnight Sun, UWAFIT, or Watonomous, you can form your team with them.
 - *Graduate Students*: You can collaborate with a team of graduate students, for example, as part of a research project.

2.5 Activities to Assist with Project Selection

1. *Watch* old project videos. See what projects look like when they are finished. Think about what you want your project to look like when it is finished.
2. *Observe* past awards.
3. *Creative* activities to generate new project proposals.
4. *Conceptual* activities to analyze project proposals.
5. *Hackathons* are sometimes focused on specific application areas, and can connect you with other people in those spaces.
6. *UW Problem Lab* is an effort that originated in UW's Economics Department, to identify opportunities that can make a difference.

<https://uwaterloo.ca/problem-lab/>

2.6 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.

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 accompanying text to elaborate on the status sheet and explain why the project was abandoned.

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

2.6.1 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

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

A good example report was written by Team SeaSalt in SE2013: <https://git.uwaterloo.ca/secapstone/abstracts/-/blob/master/reports/project-change-se2013-seasalt.pdf> This example report pre-dates the current project status sheets, but speaks to the important issues.

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 vigorously pursuing partnership opportunities with clubs and organizations to use Schoolax as their official hub for communicating with members about upcoming events.

2.6.2 *Difficulty Acquiring Data*

SE2013 Team SeaSalt began with the following project:

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).

2.6.3 *Problem Too Hard*

2014 Team Satisfaction started with the following project:

Text provided by James Rossy, member of Team SeaSalt.

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.

One 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.

2.6.4 *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.

2.6.5 *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.

2.6.6 *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 (§??) 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.

2.6.7 *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 calendars easy. The primary principle behind Avizu is

the ability to share calendars with others or find calendars that you feel would interest you, and later be able to receive notifications and changes as these calendars 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 (§??).

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.

2.7 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.

*Everything is awesome. Everything is cool
when you're part of a team.*
— The Lego Movie theme song

Teamwork Activities

Working together in a team is a major component of the capstone design experience. Together you can learn more — and accomplish more — than on your own. Additionally, almost all professional engineering work is done in teams, so learning to work together is a vital professional skill. This chapter has a variety of activities to guide your team to realize its potential.

A good complementary resource to this chapter is *Successful Strategies for Teams* by Kennedy & Nilson.

3.1 Identify Individual Learning Objectives & Skills

Each individual on the team makes two lists: one for motivations and learning objectives, and another for current strengths and skills. Team formation and project selection sometimes skews towards current strengths and skills, but you will get more out of the experience if you place sufficient emphasis on learning objectives and motivations. You have plenty of time to learn new skills in this project.

Learning objectives and motivations are not limited to technical topics, and might include societal or entrepreneurial objectives, *etc.*

When everyone has their two lists, then look for different ways in which your various objectives and skills might complement each other. For example, perhaps person x has a technical skill that person y would like to learn, so person y can be responsible for that part of the project and get guidance from person x .

3.2 Write a Team Working Agreement

HOW THEY HELP:

- Develop a *sense* of shared responsibility.
- Increase member's *awareness* of their own behaviour.
- *Empower* the facilitator to lead according to the agreements.
- *Enhance* the quality of the group process.

<http://www.payton-consulting.com/agile-team-working-agreements-guide/>

AGREEMENTS WORK WELL WHEN:

- They are *important* to the team.
- They are *limited* in number.
- They are *fully supported* by each member.
- The members are reminded of agreements during *process checks*.
- The members are reminded of agreements when they are *broken*.

REVISE EVERY RELEASE CYCLE. Agreements should be reviewed and revised every release cycle (or term).

WHEN AGREEMENTS ARE BROKEN

- gently call it out; perhaps with friendly humour
- ask 'should the agreement be updated?'

TEAM DISCIPLINES. Choose ~ 5 to prioritize. Choose items that the team needs to work on. Probably most of the items on this list are good for most teams, but your team might already be doing well implicitly on many of them. Pick some things to focus on improving.

- Don't interrupt teammates
- Practice active listening
- Loud people talk last
- Cellphones off
- Dissenter hat: one person takes the role of finding weaknesses in the idea being discussed, so the team can make it stronger
- Focus on the Sprint Goal — defer things that come up
- Don't be afraid to ask for help
- Be on time, end on time, have an agenda.
- Tell the truth
- Communicate individual schedule
- All changes to the Sprint / Backlog must be approved by the Team
- Use the Impediment Backlog (or swimlane) for blocked issues
- Define and adhere to 'done' criteria for stories
- Define and adhere to Version Control rules

These examples gathered from a number of sources, as indicated by the citations in the margin.

<https://agilepainrelief.com/blog/team-friction-inspires-working-agreements.html>

https://tech.gsa.gov/guides/agile_team_working_agreement/

- Adhere to code documentation standards
- Update Backlog before Standup daily
- Respect your team member's time
- Support each other
- Raise concerns/impediments/problems promptly
- If behind schedule, remove lowest priority work items first
- If ahead of schedule, add work items from the backlog by priority
- Be positive
- No blame
- Focus on the situation, issue, or behaviour, not the person.
- Maintain the self-confidence and self-esteem of others.
- Maintain constructive relationships.
- Take initiative to make things better
- Lead by example.
- Think beyond the moment.
- All members will attend meetings or notify the team in advance of anticipated absences.
- Decisions requiring unanimity can be made during meetings even if all team members are not present
- Timely attendance and active contribution (providing constructive feedback/ suggestions/ discussion) is expected from ALL members.
- All members will be fully engaged in team meetings and will not work on other assignments during meetings.
- Each member will take turns listening as well as talking, and active listening will be a strategy for all group discussions.
- If a solution to a discussion/conflict cannot be agreed upon, the solution will be derived by mandatory vote
- The entire team must come to a unanimous decision to make any future changes
- Our team will meet every Thursday after CHE102 class at Williams Coffee in the plaza
- Agendas for the weekly meetings will be discussed in the team's Facebook page.
- X will take notes during all meetings and email them to the team within 1 day.
- Meetings will still occur if a member cannot attend. They will be responsible for updating themselves on the discussion points by reading the minutes and asking any follow up questions.
- Decisions requiring unanimity cannot be made during meetings if all team members are not present
- All group members will come to the meetings prepared by
 - reading the assigned material (as much as possible), and
 - coming with ideas pertaining to the tasks and decisions to be

<https://www.c-sharpcorner.com/article/what-is-scrum-team-working-agreement-and-why-we-ne>

<https://uwaterloo.ca/organizational-human-development/learning-development-programs/basic-principles>

<https://git.uwaterloo.ca/secapstone/teamwork-clinic>

made.

- All members are expected to communicate any concerns/issues/ideas to the team
- In the event that a member cannot make a meeting or deadline, all members will be informed as immediately as possible.

3.3 *Communicate about Communication*

Talk about how you are going to talk. Establish a regular meeting schedule and attendance expectations, and then work together to figure out how you are going to stay in communication between meetings. Also establish what is on the agenda for whole-team meetings and what should be discussed 1:1. Come to an understanding about how often team members are going to monitor the communication channels. Especially for larger teams, or teams where there are diverse activities going on, it can be useful to compartmentalize discussions, enabling easier search and archiving. But it's also possible to set up too many channels, all empty. One team reported that having a separate channel for pull requests was useful.

Being explicit about communication can be especially useful if you happen to have team members that are not physically in Waterloo (off-stream, on coop, etc). A well-functioning team from SE2023 reports:

We really improved our offline communication by setting up a Discord server with multiple channels for the different facets of the work we had to do this term. We also had an urgent channel of communication in a Facebook Messenger chat. Having this set up at the start of the term was really beneficial so it would be good to incorporate this as a learning activity requiring some thoughtfulness about how these communication channels will be used.

3.4 *Host a Retrospective Meeting*

The team should periodically meet to reflect on its process:

- What is working well?
- What isn't working as well as it might?
- What can be improved for next time? Can it be measured?

There are a variety of different ways to approach these questions, such as:

- *Individual intuition*: First, privately write down your immediate thoughts, then share with the team.
- *Collective data*: First work together as a team to gather and assemble data, then generate insights from the data.
- *Structured questions*: Use a structured set of questions, such as Edward De Bono's *six thinking hats* §4.10.

<https://www.agility11.com/blog/2019/9/4/the-six-thinking-hats-retrospective>

3.5 *Practice Backup Behaviour (Supporting Teammates)*

Teammates need to reach out for help when they need it, and other teammates need to step up to provide it.

3.6 Practice Active Listening

Active listening is an important communication skill that can make your team more effective. Here are some techniques to practice:

1. The Basics

- Focus your attention on the speaker
- Take notes
- Paraphrasing: repeat back what you think they said
- Summarizing

2. Questions

- Ask open-ended questions
- Ask clarifying questions
- Ask for more information
- Ask for their opinions and analysis
- Let the speaker know when you don't understand
- Listen all the way to the end, then ask: is there anything else?

3. Nuance

- Use silence
- Use body language
- Acknowledge, and ask about, emotions
- Validate concerns
- Verify assumptions

3.7 Practice Assertive Communication: DESC

The DESC script¹ might help you communicate effectively in challenging situations:

1. *Describe* the behaviour/situation as completely and objectively as possible.
2. *Express* your feelings or thoughts about the behaviour/situation. Use 'I' statements.
3. *Specify* what behaviour/outcome you would prefer.
4. *Consequences*, both positive and negative.

<https://agilelearninglabs.com/2008/03/active-listening-techniques/>

¹ Sharon Anthony Bower and Gordon H. Bower. *Asserting Yourself*. Da Capo Lifelong Books, 2004

https://your.yale.edu/sites/default/files/advifomanagers_usingdesctomakeyourdifficultconversations.pdf

The Basic Principles for the University of Waterloo Workplace:

1. Focus on situation, issue, or behaviour, not the person. Do not create unnecessary relationship conflict.
2. Maintain the self-confidence and self-esteem of others.
3. Maintain constructive relationships.
4. Take initiative to make things better.
5. Lead by example.

<https://uwaterloo.ca/organizational-human-development/learning-development-programs/basic-principles>

3.8 *Apply Two Techniques to Manage Contagious Emotions*

Emotions, including prickly ones, can be contagious. Some emotions can help us perform better, but others sometimes distract us from achieving our goals as well as we might.

There are two main techniques for managing contagious emotions. Both of them involve being self-aware when you are emotionally triggered, and mentally standing back from the situation. The strategies are compatible: you can apply both.

COGNITIVE REAPPRAISAL is a technique for using your mind to think of alternative ways to view the situation — to *reframe* the situation. *Cognitive appraisal* is the mental act of assigning emotional meaning to a situation based on your understanding of the situation. For example, if a friend gives you the cold shoulder, you might think they are upset with you, and then you might think that's unfair and get upset yourself. So that's your first cognitive appraisal of the situation: my friend is upset with me. That first cognitive appraisal is going to happen — don't try to stop it.

The key is to then stand back and try to *reappraise* the situation. Could there be another explanation for your friend's behaviour? Maybe they are deep in thought. Maybe something happened in their life. These reappraisals change the emotional meaning of the experience for you.

ACCEPTANCE is about observing and acknowledging your emotions and physiological response to a situation. It doesn't mean that you are ok with what happened. It's about accepting a situation for what it is, rather than what you want it to be. There are three questions:²

1. *What?* Mentally and emotionally stand back and allow yourself space to recognize what's happening and your response to it. Get an aerial view of the situation. Don't try to control or judge your emotions: just recognize them.
2. *So what?* What does the situation mean? Why is it important? How important is it?
3. *Now what?* What next steps are going to help you and the team move forward towards your goals?

https://www.ted.com/talks/jessica_woods_what_a_cactus_taught_me_about_prickly_emotions/transcript?language=en

<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6188704/>

A lateral thinking skill.

² G. Rolfe, D. Freshwater, and M. Jasper. *Critical reflection in nursing and the helping professions: a user's guide*. Palgrave Macmillan, Basingstoke, 2001

3.9 Apply Team Formation Strategies

Identify which stage of formation your team is in, and apply appropriate strategies to move your team forward. The most commonly used model of team formation has four stages:

1. *Forming* — getting to know each other
2. *Storming* — figuring out how the team fits together; initial conflicts
3. *Norming* — getting on the same page
4. *Performing* — consistent performance

Tuckman model of team development: https://en.wikipedia.org/wiki/Tuckman's_stages_of_group_development

Figure 3.1 illustrates these four stages (as well as a fifth stage for project wind-down), and identifies relevant team formation strategies at each stage. Identify which stage of formation your team is in and apply appropriate strategies.

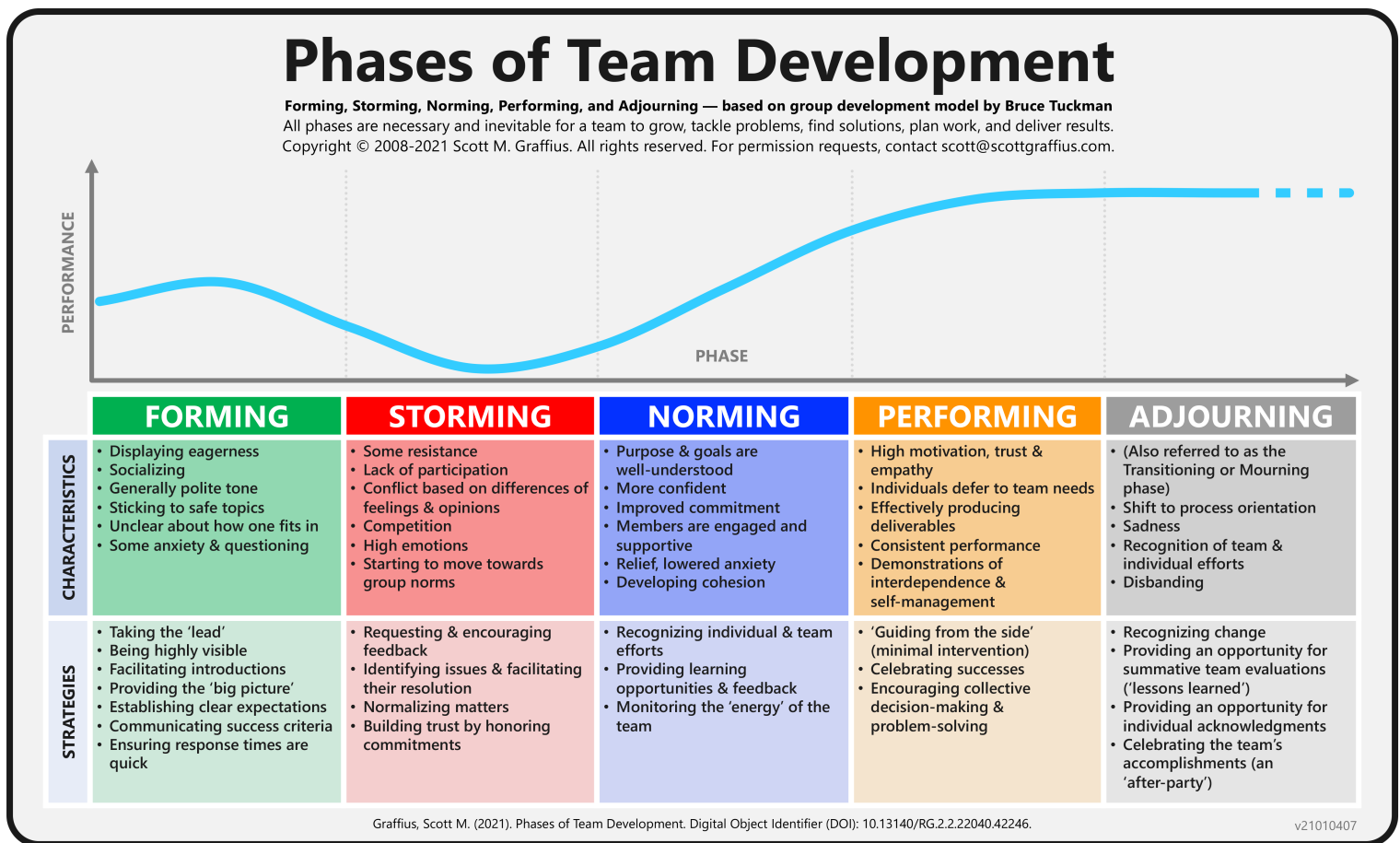


Figure 3.1: Phases of team development. Image used with permission of owner. Copyright ©2008-2021 Scott M. Graffius. All rights reserved.

3.10 Identify and Resolve Your Teams Dysfunctions

There are five common ways that a team can be dysfunctional,³ as depicted in Figure 3.2. A key insight is to work from the bottom of the pyramid upwards when resolving dysfunctions. Without trust, nothing else can be resolved. Lack of commitment cannot be resolved when there is fear of (healthy, productive) conflict. And so on.

³ P. Lencioni. *Five Dysfunctions of a Team*. John Wiley and Sons Inc., New York, NY, 2002
<https://www.youtube.com/watch?v=GCxct4CR-To> (2min video)
<https://www.youtube.com/watch?v=O5EQW026aY> (36min video)

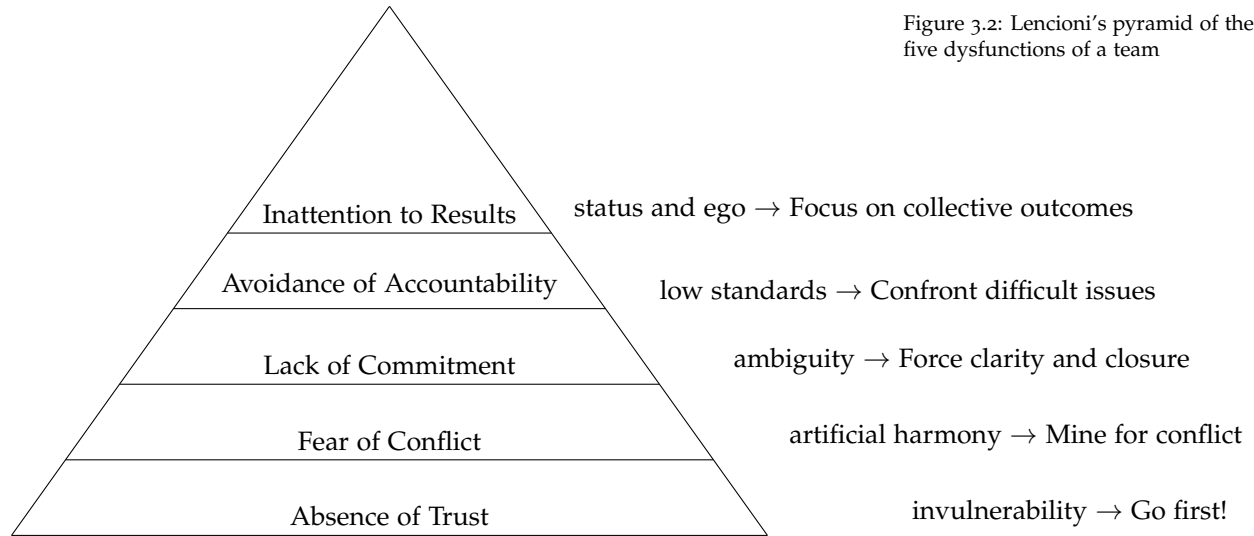


Figure 3.2: Lencioni's pyramid of the five dysfunctions of a team

3.11 *Conflict: Introduction*

The content in this section is from the University of Waterloo Teamwork Clinic.

CONFLICT CAN BE PRODUCTIVE when it:

- clarifies goals
- leads to better decision-making
- leads to self-development

Conflict can also be destructive when it is centred on relationships.

THREE CATEGORIES OF CONFLICT:

- relationship / personal
- task (the work itself; also includes resource conflict)
- process (how to do the work)

COMMON SOURCES OF CONFLICT:

- Value asymmetry – differing goals/interest in a project and its outcomes
- Social Loafing – unequal contributions to the project (quality/quantity)
- Ego/personality – poor relationships and team culture
- Poor Communication – differing expectations or communication channels
- Poor Project Management – missing meeting notes, action items, time lines

3.12 Conflict: Difficult Behaviours

The content in this section is from the University of Waterloo Teamwork Clinic.

DIFFICULT BEHAVIOURS include

- Assigning blame to others
- Insulting other team members
- Over-distress as a response to criticism
- Becoming easily frustrated and showing this inappropriately
- Refusing to negotiate or pushing for one way as the only way
- Undermining/sabotaging the work or reputation of others (gossip)
- Having difficulty taking responsibility for own behaviour

DEALING WITH DIFFICULT BEHAVIOUR:

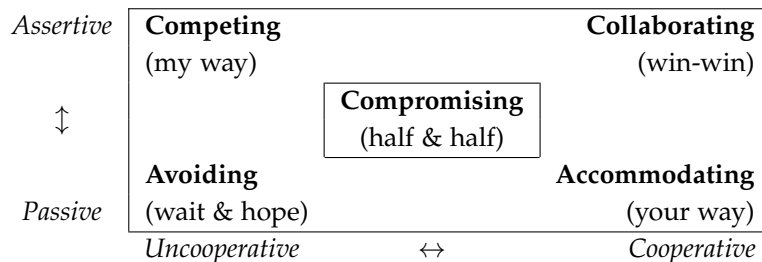
1. Develop an awareness of your *biases* before confrontation
 - Have you had *previous* conflicts/encounters with this individual before?
 - What *assumptions* are you making about why they may be acting this way?
 - What *hesitations* do you have about confronting them?
2. Approach with *concern* for the individual instead of personal defence/bias/dislike. To do this focus on two outcomes:
 - Can the conflict lead to the self-development of those involved? How?
 - Can the conflict lead to a better solution for the team? How?
3. Plan to *depolarize* the situation
 - Finding common ground on a different topic to repair the relationship after a conflict discussion
 - How can you depolarize the situation afterwards?

ADDRESSING DIFFICULT BEHAVIOUR:

1. First Attempt: Select one member to approach the individual 1-on-1
2. Second Attempt (if there is no change): select one member to approach the individual 1-on-1 but also involve a third individual (preferably a non-member) who does not contribute or take sides but observes the process
3. Third Attempt (if there is still no change): involve the professor/TA in the conversation using a similar approach to explain the process and outline the Assertive Communication points made in previous attempts

3.13 Conflict: Five Handling Modes

There are five different modes for handling conflict,⁴ which can be organized as depicted in Figure 3.3. Each of these modes is appropriate in some context, and each requires different skills.



⁴ K. W. Thomas and R. H. Kilmann. *Thomas-Kilmann Conflict Mode Instrument*. Xicom, Tuxedo NY, 1974
https://en.wikipedia.org/wiki/Thomas-Kilmann_Conflict_Mode_Instrument

Figure 3.3: Five conflict-handling modes

Skills material below from the University of Waterloo Teamwork Clinic.

COLLABORATION SKILLS — two heads are better than one

- Use active and effective listening
- Ability to apply non-threatening confrontation
- Analyzing input in order to identifying concerns

Apply when:

- the conflict is important
- compromise isn't good enough
- merging perspectives
- gaining commitment
- improving relationships
- learning

COMPROMISE SKILLS — split the difference

- Ability to negotiate and keep an open dialogue
- Assessing and attributing value of all perspectives
- Finding a resolution that is fair to both sides (middle ground)
- Making concessions or give up some of your desires

Apply when:

- issues of moderate importance
- you have equal power status
- when you have a strong commitment for resolution
- temporary solution when there are time constraints

ACCOMMODATING SKILLS — let's do it your way

- Willingness to sacrifice your desires and act with selflessness
- Ability to yield to the desires of others
- Ability to obeying orders

Apply:

- to show reasonableness
- to develop performance
- to create good will
- to keep peace
- when issue is unimportant to you

AVOIDING SKILLS — leave well enough alone

- Knowing when to withdraw
- Ability to leave things unresolved
- Ability to sidestep issues or sensitive topics diplomatically
- Sense of timing

Apply when:

- issues of low importance
- to reduce tensions
- to buy some time
- you are in a position of lower power

COMPETING SKILLS — might makes right

- Arguing/debating
- Standing your ground
- Stating your position clearly
- Asserting your opinions/feelings
- Using rank/influence

Apply when:

- quick action needs to be taken
- unpopular decisions need to be made
- vital issues must be handled
- protecting self-interests

3.14 *Conflict: Reflection Questions*

1. Identify your personality type and your teammates personality types
2. Which strategies did you select from the Personality Differences sheet to implement this term?
3. Describe one particular conflict that have occurred amongst your team this term.
4. What type of conflict was this, task, process or relationship? What makes you think this?
5. What led to the conflict?
6. What strategies did you implement to try and resolve the conflict?
7. In hindsight, which other strategies could you have considered to help resolve the conflict?
8. What one thing might you do differently next time you work in a team to help avoid this type of conflict?

Team conflict reflection questions from the University of Waterloo Teamwork Clinic.

3.15 Conflict: Situation Assessment

Team conflict situation assessment worksheet from the University of Waterloo Teamwork Clinic.

Team Conflict Assessment

This assessment will help your team decide the best handling style (avoiding, accommodating, compromising, collaborating, competing) to implement to resolve the conflict being discussed. Discuss each statement below as a team and check all that accurately describe your team's conflict situation.

Check all that are true	Conflict Situation Statements	Conflict Style
	The issue is small and there are more important issues to deal with	AVOID
	The issue is not having a negative impact on any relationships in the team	AVOID
	Before we can deal with this issue effectively we need to gather more information	AVOID
	There are negative emotions or feelings lingering in some members due to this issue	AVOID
	TOTAL	
	The issue is not seen as important to all team member(s)	ACCOMMODATE
	There is a chance that the information we have is wrong or incomplete	ACCOMMODATE
	It is not a good time to resolve this issue	ACCOMMODATE
	Maintaining harmony in the relationships is extremely important	ACCOMMODATE
	TOTAL	
	Everyone involved is equal in power	COMPROMISE
	Those involved have opposing views and are resistant to changing them	COMPROMISE
	To save time, we can reach a temporary solution for parts of a more complex issue	COMPROMISE
	We have tried to collaborate around this issue and failed	COMPROMISE
	TOTAL	
	Better understanding the different perspectives on this issue is important for resolution	COLLABORATE
	It is important for all individuals to accept and commit to the outcome	COLLABORATE
	The individuals involved are flexible in their thinking and are willing to adjust as more information is presented and new options are suggested	COLLABORATE
	There are existing hard feelings/animosity among members	COLLABORATE
	TOTAL	
	It is not important to consider opposing perspectives in order to deal with this issue	COMPETE
	Time is short and a quick decision is needed immediately	COMPETE
	It is clear that some members do not have a chance to contribute their perspectives to the discussion and need to stand up for their rights/ideas	COMPETE
	The issue involves implementing a difficult decision (ie. enforcing rules/discipline)	COMPETE
	TOTAL	

The style with the most checks can be considered a reasonable approach in managing this conflict. However, consider the following statements before deciding as a team which style should be used to resolve your conflict.

CHECK ALL THAT ARE TRUE:

	It is important that the issue is resolved	DO NOT AVOID
	It is important that everyone's ideas/voice are recognized and not neglected	DO NOT ACCOMMODATE
	Innovation is an important outcome for our team	DO NOT COMPROMISE
	We do not have a lot of time to reach a solution	DO NOT COLLABORATE
	It is very important that the others involved do not come to resent one	DO NOT COMPETE

Team Conflict Assessment

In order to make an informed decision about which style to use, consider the potential outcomes or consequences of applying each conflict handling style below.

COMPETE: *"Might makes right" (I win, you lose)*

Positive Outcome:

Negative Outcome:

COLLABORATE: *"Two heads are better than one" (I win, you win)*

Positive Outcome

Negative Outcome

COMPROMISE: *"Split the difference" (I win some, you win some)*

Positive Outcome

Negative Outcome

ACCOMMODATE: *"Kill your enemies with kindness" (I lose, you win)*

Positive Outcome

Negative Outcome

AVOID: *"Leave well enough alone" (I lose, you lose)*

Positive Outcome

Negative Outcome

BEST APPROACH TO MANAGE THE TEAM'S CONFLICT: _____

3.16 Conflict: Personality-Based Coping Strategies

Personality-based conflict coping strategies from the University of Waterloo Teamwork Clinic.

Coping with Being Different

Your Preference



Extraversion

Workgroup's Preference



Introversion

Consider these tactics:

- Networking with others outside your team
- Asking them to voice their ideas
- Paying attention to written notices and email
- Allowing others to think about your idea before they provide feedback (count to three – or ten...)

Your Preference



Introversion

Workgroup's Preference



Extraversion

Consider these tactics:

- Arriving at work early to take advantage of quiet time
- Intentionally seeking out private/reflective time – take the long way home
- Planning private breaks throughout the day to collect your thoughts
- In meetings, voicing even partially thought-through perspectives

Coping with Being Different

Your Preference



Sensing

Workgroup's Preference



Intuition

Consider these tactics:

- Getting involved in projects that require long-range or future thinking
- Practice "brainstorming" with the rest of the team
- Preparing yourself for "roundabout" discussions – look for patterns
- Going beyond specifics – try to discover meanings and themes

Your Preference



Intuition

Workgroup's Preference



Sensing

Consider these tactics:

- Practice presenting information in a step-by-step manner
- Providing specific examples of vital information
- Honouring organisational values surrounding experience and tradition
- Reading the fine print and getting the facts straight

Coping with Being Different

Your Preference



Thinking

Workgroup's Preference



Feeling

Consider these tactics:

- Working on projects in which alternative causes and solutions are evaluated in personal terms
- Reminding yourself that factoring in the impact on people is logical even if people aren't
- Softening critical remarks – finding the positive, too
- Asking for others' opinions and concerns, looking for points of agreement before discussing issues

Your Preference



Feeling

Workgroup's Preference



Thinking

Consider these tactics:

- Practice laying out an argument logically by saying if...then, or by considering the causes and effects
- Understanding that critical feedback is often given in the spirit of improving your professionalism
- Bringing attention to stakeholders' concern regarding projects/work
- Using brief and concise language to express your wants and needs

Coping with Being Different

Your Preference



Judging

Workgroup's Preference



Perceiving

Consider these tactics:

- Seeking out projects that have definite milestones and a final deadline
- Trying to wait on a decision for a few days, continuing to gather more information and paying attention to ideas that may come up
- Understanding that work is progressing despite differences in work styles
- Making your own milestones and deadlines

Your Preference



Perceiving

Workgroup's Preference



Judging

Consider these tactics:

- Recognising that deadlines set by the organisation may not be negotiable
- Using a past decision you believe others rushed to demonstrate the advantages of slowing down to gather more information
- Becoming active in projects where the process is just as important as the outcome
- Keeping "surprises" to a minimum and reducing your options

3.17 Health: Balsom's 9 Attributes of Effective Teams⁵

Briefly assess your team on each of these nine attributes:

1. *Mission & Goals*: High performing teams have a clear mission and/or goals.
2. *Leadership*: Leadership functions are performed reliably.
3. *Communication*: Members learn enough of what others believe, soon enough.
4. *Decision Making*: Systematic, agreed-upon decision processes are used.
5. *Culture*: The cultures of high performing teams provide predictability and alignment.
6. *Group Motivation*: Psychological membership in a valued group can be a source of motivation and a resource for overcoming problems.
7. *Conflict*: Effective teams address task conflict productively and prevent personalized conflict.
8. *Processes/Meetings*: Processes/meetings facilitate communication and decision making.
9. *Self-Management*: Individual members are efficient self-managers.

⁵ M. Balsom, R. Barrass, J. Michela, and A. Zdaniuk. Processes and attributes of highly effective teams. Technical Report The WORC Group, University of Waterloo, 2009
Balsom.pdf

3.18 Health: Google

Google conducted over 200 interviews with employees and examined 250 attributes of 180+ Google teams. What they discovered is that the résumés of the team members matters less than how the team members interact with each other. They discovered five key dynamics of successful teams. Self-assess your team (green/yellow/red) on these five dynamics.

<https://rework.withgoogle.com/blog/five-keys-to-a-successful-google-team/>
<https://www.nytimes.com/2016/02/28/magazine/what-google-learned-from-its-quest-to-build-the-perfect.html>
<https://apnews.com/article/8c60341cc1da47e084b8e17e62e83c98>

<i>Dynamic</i>	<i>Ideal (Green)</i>
<i>Psychological Safety</i>	Can we take risks on this team without feeling insecure or embarrassed?
<i>Dependability</i>	Can we count on each other to do high quality work on time?
<i>Structure & Clarity</i>	Are goals, roles, and execution plans on our team clear?
<i>Meaning of work</i>	Are we working on something that is personally important for each of us?
<i>Impact of work</i>	Do we fundamentally believe that the work we're doing matters?

Google has also developed a complementary *Discussion Guide* with indicators and questions for each dynamic. Use this discussion guide in a team conversation.

Discussion Guide links:
<https://docs.google.com/...>
<https://git.uwaterloo.ca/secapstone/teamwork-clinic/...>

3.19 *Health: Team Barometer*

Assess the team with green/yellow/red in each of the areas.

By Jimmy Janlén

<https://blog.crisp.se/2014/01/30/jimmyjanlen/team-barometer-self-evaluation-tool>

<i>Area</i>	<i>Green</i>	<i>Red</i>
<i>Trust</i>	We have the courage to be honest with each other. We don't hesitate to engage in constructive conflicts.	Members rarely speak their mind. We avoid conflicts. Discussions are tentative and polite.
<i>Collaboration</i>	The team cross-pollinates, sharing perspectives, context and innovations with other teams and other parts of the organization.	Work is done individually. Little or no collaboration within the team or with other teams.
<i>Feedback</i>	We give positive feedback, but also call out one another's deficiencies and unproductive behaviours.	We rarely praise each other or give feedback or criticize each other for acting irresponsibly or breaking our Working Agreement.
<i>Meeting Engagement</i>	People are engaged in meetings. They want to be there. Discussions are passionate.	Many feel like prisoners in the meeting. Only a few participate in discussions.
<i>Commitment</i>	We commit to our plans and hold each other accountable for doing our best to reach our goal and execute assigned action points.	We don't have real consensus about our goals. We don't really buy in to the plan or follow up that people keep their commitments.
<i>Improving</i>	We passionately strive to figure out how to work better and more efficiently as a team. We try to 'know' if we get better.	We don't focus on questioning our process or way of working. If someone asked us to prove that we've gotten better we have no clue how we would demonstrate that.
<i>Mutually Responsible</i>	We feel mutually responsible for achieving our goals. We win and fail as a team.	When we fail we try to figure out who did what wrong. When we succeed we celebrate individuals. If we pay attention to it at all ...
<i>Power</i>	We go out of our way to unblock ourselves when we run into impediments or dependencies.	When we run into problems or dependencies we alert managers, ask for their help, and then wait.
<i>Pride</i>	We feel pride in our work and what we accomplish.	We feel ashamed of our pace and the quality of our results.

...continued next page ...

... *Team Barometer continued from previous page* ...

<i>Relationships</i>	Team members spend time and effort building strong relationships among themselves, as well as with partners outside the team.	We don't really know each other or what makes other's 'tick.'
<i>Ownership</i>	We engage in defining our own goals and take ownership of our destiny.	We act as pawns in a game of chess. We don't demand involvement in defining our goals and destiny.
<i>Sharing</i>	We share what we know and learn. No one withholds information that affects the team.	People do stuff under the radar and often forget to share news or relevant information.
<i>Boosts each other</i>	We unleash each other's passion and care for each other's personal development. We leverage our differences.	We don't know in which areas people want to grow. We have trouble collaborating since we are very different and view things differently.
<i>Loyalty</i>	No one has hidden agendas. We feel that everyone's loyalty is with this team.	The team feels like a disconnected group of people with different goals and loyalties that lie elsewhere.
<i>Passion</i>	Each member wants this team to be great and successful.	People just come to work and focus on their own tasks.
<i>Integrity</i>	We honour processes and working agreements even when we are put under pressure.	Our behaviours, collaboration and communication fall apart when we get stressed.

3.20 *Health: TeamRetro*

Assess your team with green/yellow/red in the following areas.

<https://www.teamretro.com/health-checks/team-health-check/>

<i>Area</i>	<i>Ideal</i>
<i>Ownership</i>	The team has clear ownership or a dedicated product owner who is accountable for team's results and champions the mission inside and outside of the team.
<i>Value</i>	We can define, measure and deliver the value we provide to the business and the user.
<i>Goal alignment</i>	Everyone understands why they are here, supports the idea and believe they have what it takes to create solutions that add value.
<i>Communication</i>	We have clear and consistent communication that ensures that issues are shared, conflict is reduced, and everyone can work at greater efficiency.
<i>Team roles</i>	The current team skill set is right for the current stage and there are clear roles and responsibilities for each person in the team.
<i>Velocity</i>	We learn and implement lessons leading to incremental progress in iterations and production as we go.
<i>Support and resources</i>	We are equipped with the right tools and resources and can easily access support from within and outside the team.
<i>Process</i>	Our processes are aligned, effective and do not cause unnecessary delays and blocks. We have metrics in place to measure our goals.
<i>Fun</i>	We enjoy our work and working as a team. We are being challenged and can develop our skill set or acquire new ones.

3.21 Health: Spotify

Assign green, yellow, or red for each area of inquiry.

- *Green*: working well (not necessarily perfect).
- *Yellow*: some issues need to be improved, but not a disaster.
- *Red*: serious problems.

<https://engineering.atspotify.com/2014/09/16/squad-health-check-model/>
<https://www.andycleff.com/2020/05/agile-team-health-check-models/>

AREAS OF INQUIRY:

<i>Area</i>	<i>Example of Awesome</i>	<i>Example of Crappy</i>
Easy to release	Releasing is simple, safe, painless & mostly automated.	Releasing is risky, painful, lots of manual work, and takes forever.
Suitable process	Our way of working fits us perfectly	Our way of working sucks
Tech quality (code base health)	We're proud of the quality of our code! It is clean, easy to read, and has great test coverage.	Our code is a pile of dung, and technical debt is raging out of control
Value	We deliver great stuff! We're proud of it and our stakeholders are really happy.	We deliver crap. We feel ashamed to deliver it. Our stakeholders hate us.
Speed	We get stuff done really quickly. No waiting, no delays.	We never seem to get done with anything. We keep getting stuck or interrupted. Stories keep getting stuck on dependencies
Mission	We know exactly why we are here, and we are really excited about it	We have no idea why we are here, there is no high level picture or focus. Our so-called mission is completely unclear and uninspiring.
Fun	We love going to work, and have great fun working together	Boooooooring.
Learning	We're learning lots of interesting stuff all the time!	We never have time to learn anything
Support	We always get great support & help when we ask for it!	We keep getting stuck because we can't get the support & help that we ask for.
Pawns or players	We are in control of our destiny! We decide what to build and how to build it.	We are just pawns in a game of chess, with no influence over what we build or how we build it

3.22 Process Assessment: Joel Test

Joel Spolsky created *The Joel Test* in 2000 as an alternative to heavy-weight process assessments. It is twelve simple yes/no questions:

<https://www.joelonsoftware.com/2000/08/09/the-joel-test-12-steps-to-better-code/>

1. Do you use source control?
2. Can you make a build in one step?
3. Do you make daily builds?
4. Do you have a bug database?
5. Do you fix bugs before writing new code?
6. Do you have an up-to-date schedule?
7. Do you have a spec?
8. Do programmers have quiet working conditions?
9. Do you use the best tools money can buy?
10. Do you have testers?
11. Do new candidates write code during their interview?
12. Do you do hallway usability testing?

Joel says that professional companies score 12/12, and that scores of 10 or less are a concern. That interpretation doesn't make sense in the context of student capstone projects, because some of the questions are about the organization as a whole (*e.g.*, hiring practices). For question 9, do you use the best tools money can buy?, be aware that Joel's company sells software engineering tools (bug/issue tracker). The economics of software engineering tools are a rich area for discussion.

The Joel Test has proven popular and useful. In 2019, Brian Conway proposed some updates for how practices have evolved:

<https://medium.com/meshify/an-updated-joel-test-for-2019-fc732ad24dc6>

1. Is your source control effective for the offline programmer?
2. Can you make a build in one step?
3. Do you build, test, and deploy (somewhere) on every commit?
4. Do you have a bug database?
5. Do you fix bugs before writing new code?
6. Does your source have a stable branch from which releases can be cut at any time?
7. Can all of a product's functionality be verified in one step?
8. Do programmers have access to a door that can be closed for uninterrupted work?
9. Do programmers have the freedom to choose the best tools for the job?
10. Is testing the team's responsibility?
11. Do new candidates write code during their interview?
12. Do you do hallway usability testing?

3.23 Process Assessment: Scrum Checklist

Scrum is a lightweight teamwork process framework that is popular in software engineering (and beyond).

Henrik Kniberg (Mojang/Minecraft) has developed a nice Scrum checklist, which is used all over the world and has been translated into about twenty different languages.

Complete the Scrum checklist and discuss your findings.

[https://en.wikipedia.org/wiki/Scrum_\(software_development\)](https://en.wikipedia.org/wiki/Scrum_(software_development))

<https://www.scrum.org/resources/what-is-scrum>

<https://www.crisp.se/gratis-material-och-guider/scrum-checklist>

The bottom line
If you achieve these you can ignore the rest of the checklist. Your process is fine.

- Delivering **working, tested software** every 4 weeks or less
- Delivering what the **business needs** most
- Process is **continuously improving**

Core Scrum
These are central to Scrum. Without these you probably shouldn't call it Scrum.

- Retrospective** happens after every sprint
- Results in concrete improvement **proposals**
- Some proposals actually get **implemented**
- Whole team + PO** participates
- PO has a **product backlog (PBL)**
- Top items are **prioritized** by business value
- Top items are **estimated**
- Estimates written by the team**
- Top items in **PBL small enough to fit** in a sprint
- PO understands **purpose** of all backlog items
- Have **sprint planning meetings**
- PO participates**
- PO brings **up-to-date PBL**
- Whole team** participates
- Results in a **sprint plan**
- Whole team believes plan is **achievable**
- PO **satisfied with priorities**
- Timeboxed iterations**
- Iteration length **4 weeks or less**
- Always **end on time**
- Team **not disrupted or controlled** by outsiders
- Team usually **delivers what they committed to**
- Team members **sit together**
- Max 9 people** per team

Recommended but not always necessary
Most of these will usually be needed, but not always all of them. Experiment!

- Team has **all skills** needed to bring backlog items to Done
- Team members **not locked into specific roles**
- Iterations that are **doomed to fail** are terminated early
- PO has **product vision** that is in sync with PBL
- PBL and product vision is **highly visible**
- Everyone on the **team participates in estimating**
- PO available** when team is estimating
- Estimate **relative size** (story points) rather than time
- Whole team knows top 1-3 **impediments**
- SM has strategy** for how to fix top impediment
- SM focusing** on removing impediments
- Escalated to management** when team can't solve
- Team has a **Scrum Master (SM)**
- SM sits with the team**
- PBL items are **broken into tasks** within a sprint
- Sprint tasks are **estimated**
- Estimates for ongoing tasks are **updated daily**
- Velocity** is measured
- All items in sprint plan have an **estimate**
- PO uses velocity for **release planning**
- Velocity only includes items that are **Done**
- Team has a **sprint burndown chart**
- Highly visible**
- Updated daily**
- Daily Scrum** is every day, same time & place
- PO participates** at least a few times per week
- Max 15 minutes**
- Each team member **knows what the others are doing**

Scaling
These are pretty fundamental to any Scrum scaling effort.

- You have a **Chief Product Owner** (if many POs)
- Dependent teams do **Scrum of Scrums**
- Dependent teams **integrate within each sprint**

Positive indicators
Leading indicators of a good Scrum implementation.

- Having fun!** High energy level.
- Overtime work is rare** and happens voluntarily
- Discussing, criticizing, and **experimenting** with the process

PO = Product owner SM = Scrum Master PBL = Product Backlog DoD = Definition of Done
<http://www.crisp.se/scrum/checklist> | Version 2.2 (2010-10-04)

Figure 3.4: Scrum checklist by Henrik Kniberg. Available at <https://www.crisp.se/gratis-material-och-guider/scrum-checklist>

3.24 Process Assessment: CMMI

The *Capability Maturity Model Integration* (CMMI) is an organizational assessment that originated from the Software Engineering Institute (SEI) and Carnegie Mellon University (CMU). Version 2.0 was published in 2018, but the original Capability Maturity Model dates from 1986. There are six maturity levels:

0. *Incomplete*: Ad hoc and unknown. Work may or may not get completed.
1. *Initial*: Unpredictable and reactive. Work gets completed but is often delayed and over budget.
2. *Managed*: Managed on the project level. Projects are planned, performed, measured, and controlled.
3. *Defined*: Proactive, rather than reactive. Organization-wide standards provide guidance across projects, programs, and portfolios.
4. *Quantitatively Managed*: Measured and controlled. Organization is data-driven with quantitative performance improvement objectives that are predictable and align to meet the needs of internal and external stakeholders.
5. *Optimizing*: Stable and flexible. Organization is focused on continuous improvement and is built to pivot and respond to opportunity and change. The organization's stability provides a platform for agility and innovation.

3.25 Process Assessment: Capability Immaturity Model

The Capability Immaturity Model (CIMM)⁶ is a parody of the Capability Maturity Model (originally CMM, now CMMI).

0. *Negligent*: Team pays lip service to process, but then panics and flails about chaotically.
- 1. *Obstructive*: Strict adherence to inappropriate and ineffective processes. Disregard for quality of actual work.
- 2. *Contemptuous*: Everyone knows how ineffective team is, but the team doesn't care. The team fudges measurements and indicators. Poor results are blamed on factors outside the team's control.
- 3. *Undermining*: Actively downplaying and sabotaging the efforts of rivals; draining resources from more effective areas.

International standard ISO/IEC 33001:2015 *Information technology – Process assessment – Concepts and Terminology* is roughly similar to CMMI.

Text defining these levels from <https://cmmiinstitute.com/learning/appraisals/levels>

For a capstone project, consider standards or norms defined outside your team, for example in industry, government, or the open source community.

⁶ Tom Schorsch. The Capability Im-Maturity Model (CIMM). *CrossTalk Magazine*, 1995; and Anthony Finkelstein. *SIGSOFT Software Engineering Notes*, 1992. URL <http://www.cs.ucl.ac.uk/staff/A.Finkelstein/papers/immaturity.pdf> https://en.wikipedia.org/wiki/Capability_Immaturity_Model

3.26 *Process Assessment: UX Maturity*

Nielsen⁷ describes eight levels of process maturity with regards to UX maturity. These are listed in elsewhere in this document in §11.11.

3.27 *Course: PD4 Teamwork*

PD4 is a good course to take on a work term — especially in third year. Here is the course description:

An introduction to the processes and skill required of high-performance teams. Application of teamwork skills to decision making, conflict resolution and leadership. Development of self-awareness and relational skills to improve one's ability to collaborate effectively, give and receive assistance, and empower others. Personal reflection and case discussion is integrated with observations of teamwork in the co-op experience work environment.

3.28 *Course: INTEG210 Making Collaboration Work*

INTEG210 *Making Collaboration Work* is a worthwhile open elective course. It is taught by the *Knowledge Integration* department, which is focused on interdisciplinary collaboration. You can take it any time after first year. Here is the calendar description:

Collaboration and teamwork are essential for solving complex, real-world problems and are therefore in high demand by employers. Yet students rarely have the opportunity to study and apply the theory and best practices for making collaboration work. In this course, you will learn how to leverage this research to acquire a variety of important skills. These include: effective communication in groups, proactively managing group conflict, identifying biases that hinder creativity, and leveraging diversity to improve outcomes. You will also put those skills into practice throughout the course and reflect on how you can apply them in other situations.

⁷ Jakob Nielsen. Corporate ux maturity: Stages 1–4. Technical report, Nielsen/Norman Group, 2006. URL <https://www.nngroup.com/articles/ux-maturity-stages-1-4/>; and Jakob Nielsen. Corporate ux maturity: Stages 5–8. Technical report, Nielsen/Norman Group, 2006. URL <https://www.nngroup.com/articles/ux-maturity-stages-5-8/> <https://uwaterloo.ca/professional-development-program/courses/pd4-teamwork>

<https://uwaterloo.ca/knowledge-integration/current-undergraduates/course-offerings/ki-elective-courses/integ-210-making-collaboration-work>

Creative Activities

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.

4.1 Modes of Creative Thinking: Intense and Casual

Sanders & Thagard¹ 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 shower² or when out for a walk³ — 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:

1. Immersion in problem domain
2. Absence of immediate pressure
3. Absence of distractions
4. Mental relaxation
5. Unstructured time
6. 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.

¹ 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

² Alan Kay had his own shower in the basement of Xerox PARC to facilitate casual mode creativity. [54]

³ Alan Turing found that many of his ideas came to him on long walks or runs. [54]

4.2 Apply SCAMPER

Start with an existing project concept and apply these operators to it:

- S: Substitute
- C: Combine
- A: Adapt
- M: Modify / Maximize / Minimize
- P: Put to another use
- E: Eliminate
- R: Reverse

<https://www.interaction-design.org/literature/article/learn-how-to-use-the-best-ideation-methods-scamper>

4.3 Apply C-K Theory

Concept-Knowledge (C-K) theory is a relatively recent design methodology from France intended to spur innovation and the generation of new ideas.

THE CONCEPT SPACE is organized as a tree of ideas.

THE KNOWLEDGE SPACE is a bag of facts — statements that can be true or false.

THERE ARE FOUR MOVES in the methodology:

- C→C: one concept leads to another
- C→K: concept leads to knowledge
- K→C: knowledge leads to a new concept
- K→K: knowledge leads to knowledge

<https://www.ck-theory.org/c-k-theory/>
<https://www.consuunt.com/c-k-theory/>

Herbert A. Simon (Turing Award 1975, Nobel Prize in Economics 1978) famously conceptualized design as search within a known problem space. C-K theory originated with a rejection of that approach, and instead emphasized exploration within a partially unknown concept space.

4.4 Use Comparison to Generate 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 4.1 identifies some kinds of comparisons. Comparisons such as symmetry and duality involve both similarity and difference, and so we categorize them separately.

The comparisons of difference in Figure 4.1 come from what is known as the *traditional square of opposition* in logic, depicted in Figure 4.2. 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.

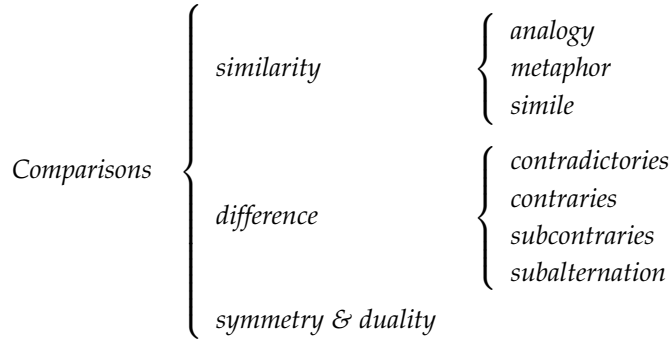


Figure 4.1: Some kinds of comparisons

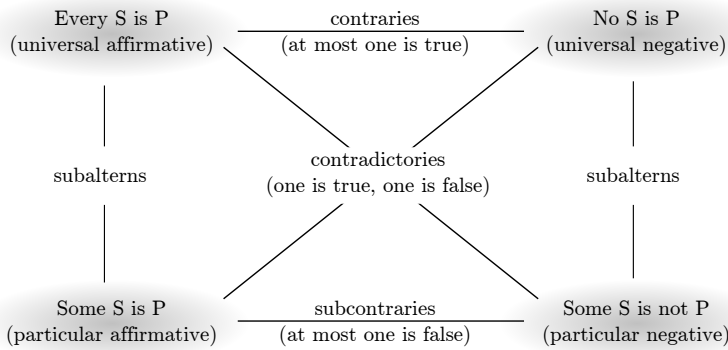


Figure 4.2: The traditional square of opposition. Originally described by Aristotle in *De Interpretatione*, and subsequently developed by many others. See, for example, the entry in the *Stanford Encyclopedia of Philosophy* by Parsons [50]: <http://plato.stanford.edu/entries/square/>

A STRATEGY FOR USING COMPARISONS to generate new ideas was proposed and tested by W.J.J. Gordon⁴: *make the familiar strange, and make the strange familiar.*

⁴ William J.J. Gordon. *Synectics: The Development of Creative Capacity*. Harper & Row, New York, 1961

LOCAL AND DISTANT ANALOGIES. Sanders⁵ & Thagard⁶ 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 Lovelace⁷ in a letter she wrote to Charles Babbage⁸ comparing Babbage's Analytical Engine⁹ to the Jacquard Loom.¹⁰

Analogy	{	<i>local</i>	Mac \approx Xerox Alto
			genetic algorithms
		<i>distant</i>	neural networks
			Analytical Engine \approx Jacquard Loom

We may say most aptly that the Analytical Engine weaves algebraic patterns just as the Jacquard loom weaves flowers and leaves.

— Ada Lovelace¹¹

SYMMETRY AND DUALITY. *e.g.*, dynamic invariant detection¹² is the dual¹³ of abstract interpretation¹⁴

IDEAS MAY BE COMBINED TO FORM NEW HYBRIDS. Von Fange¹⁵ 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.

⁵ 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

⁶ Director of Waterloo's Cognitive Science Program

⁷ The first software engineer.

⁸ The first hardware engineer.

⁹ The first computer.

¹⁰ The Jacquard Loom was programmable in the sense that it used punched cards to describe the pattern to be woven.

¹¹ H. Goldstine. *The computer from Pascal to Von Neumann*. Princeton University Press, 1972

¹² daikon

¹³ 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

¹⁴ 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

¹⁵ Eugene K. Von Fange. *Professional Creativity*. Prentice Hall, Englewood Cliffs, N.J., 1959

4.5 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. There are several kinds of *normal* programs that we have a good understanding of how to design. These are often given their own courses in an undergraduate curriculum. For example, compilers, operating systems, database engines, network protocols, text editors, and so on.

Try conceptualizing your problem as one or more of these normal programs. For example, if you designed your solution like a compiler how would it look like? What if you designed your solution like an operating system?

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. Parnas¹⁶ presented two different designs for his toy example KWIC (Key Word In Context) program: flowchart and encapsulated. Garlan & Shaw¹⁷ 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.^{18,19,20,21}

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

¹⁶ David Lorge Parnas. On the Criteria to be Used in Decomposing Systems into Modules. *Communications of the ACM*, 15(12):1053–1058, December 1972

¹⁷ 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

¹⁸ Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995

¹⁹ 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

²⁰ D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects*. Addison-Wesley, 2000

²¹ Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002

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.²² Try Prolog, Ruby, Scheme, or Haskell for a different way of thinking.

²² 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:²³

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 trade-offs: 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.

²³ Avi Bryant. Web Heresies: The Seaside Framework. OSCON, 2006. URL http://conferences.oreillynet.com/cs/os2006/view/e_sess/8942

MORPHOLOGICAL ANALYSIS.²⁴ Identify all of the design decisions and their possible alternatives, then systematically explore every possible combination.

²⁴ Fritz Zwicky. *Discovery, Invention, Research — Through the Morphological Approach*. The Macmillan Company, Toronto, 1969

WHAT WOULD DIJKSTRA 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: Edsger Dijkstra, Simon Peyton-Jones, Butler Lampson, Tony Hoare, Rob Pike, Joshua Bloch, Michael Stonebraker, Ted Codd, Linus Torvalds, Larry Wall, Donald Knuth, David Parnas, Fred Brooks, Michael Jackson.

4.6 Brainstorming

BRAINSTORMING is a commonly known technique originally developed by Alex Osborn²⁵. 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:

²⁵ Alex F. Osborn. *Applied Imagination: Principles and Procedures of Creative Problem Solving*. Scribner & Sons, New York, 1953

1. Judgement is ruled out. Criticism of ideas must be withheld until later.
2. Free-wheeling is welcomed. The wilder the ideas, the better; it is easier to tame down than to think up.
3. Quantity is wanted. The greater the number of ideas, the more the likelihood of winners.
4. 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.

HARVARD BUSINESS REVIEW has an article on brainstorming remotely: <https://hbr.org/2020/07/how-to-brainstorm-remotely>

4.7 6-3-5 Group Brainstorming

Each participant starts with a blank sheet. Iterate for 6 rounds:

1. Each participant adds 3 ideas (within 5 minutes) to their sheet.
2. Pass sheets one step around the circle.

<https://www.designthinking-methods.com/en/3Ideenfindung/6-3-5.html>
https://en.wikipedia.org/wiki/6-3-5_Brainwriting

4.8 Crazy 8s (a form of group brainstorming)

Crazy 8s is form of group brainstorming.

- <https://blog.prototypr.io/how-to-run-a-crazy-eights-workshop-60doa67b29a>
- <https://www.iamnotmypixels.com/how-to-use-crazy-8s-to-generate-design-ideas/>
- <https://uxdesign.cc/how-to-do-crazy-8s-remotely-223d7fbd5e98>

4.9 Think / Pair / Share

Dym & Little²⁶ present three related techniques that I'll summarize with the pedagogical phrase *think / pair / share*.







1. *Think*: Each person in the group sketches k ideas, for some pre-determined value of k usually in the range 1–3. A sketch may be a drawing or prose.
2. *Pair*: The sketches are passed around for written commentary. There is no talking in this step nor the previous step: all communication is written.
3. *Share*: The annotated sketches are posted on the wall and form the basis of a discussion.





²⁶ Clive L. Dym and Patrick Little. *Engineering Design: A Project Based Introduction*. John Wiley & Sons, 3 edition, 2008

4.10 *Six Thinking Hats*

Edward de Bono, who coined the term *lateral thinking*, also created the *six thinking hats* technique²⁷. 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 facilitator 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.

²⁷ Edward de Bono. *Six Thinking Hats: An Essential Approach to Business Management*. Little, Brown, & Company, 1985

-  *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.

<i>Task</i>	<i>Hat Sequence</i>
Solving Problems	
Choosing	
Performance review	
Process improvement	

SOLVE AN ANALOGOUS PROBLEM.^{28,29} 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.

²⁸ This technique was developed by W.J.J. Gordon, who also wrote the *Synectics* book. Gordon's ideas are documented by Cros.²⁹

²⁹ 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/uc1.10050673813>. Submitted in partial fulfillment of the requirements in Professor Georges F. Doriot's course in manufacturing at the Harvard Business School

Planning Activities

5.1 Exploring Early Can Be A Good Strategy

Watch the TED talk by David Epstein. He contrasts two general planning strategies: specializing early vs exploring early. In sports, Tiger Woods is an example of specializing early (started golfing as a baby). Roger Federer, by contrast, is an example of exploring early: he tried many other sports before specializing in tennis. Epstein also gives examples of technologists (*e.g.*, the founder of Nintendo) and mathematicians (*e.g.*, the first woman to win a Fields Medal).

Plans that involve specializing early are often focused on outcomes. Plans that involve exploring early are generally about skill development first, and the desired outcome isn't identified until part-way through the process.

Create two draft plans for your capstone experience: one that involves specializing early in order to achieve a specific outcome, and one that involves exploring early in order to develop skills that you are interested in.

https://www.ted.com/talks/david_epstein_why_specializing_early_doesn_t_always_mean_career_success/transcript?language=en

5.2 Select Project Success Metrics

Especially for New Product projects, it can be important to select and define the success metrics. Here are a few for consideration. You should also do your own research. Tools like Google Analytics can help measure some of these metrics.

Downloads: The number of times your app has been downloaded.

MAU (Monthly Active Users): The number of users active in a given month. Be specific how you define 'active': Facebook and Twitter, for example, have different definitions tailored to their own sites. Twitter also reports DAU (daily active users).

Bounce Rate: Number of users who get to the landing page, but do not interact with the site further.

<https://www.ycombinator.com/library/1y-key-metrics>
<https://marketingplatform.google.com/about/analytics/>

<https://www.investopedia.com/terms/m/monthly-active-user-mau.asp>

Average Session Duration: Time that users spend on the site.

Number of page views: Total number of pages viewed.

Number of pages per session.

Number of sessions per user.

5.3 *Weekly Work Intensity*

Rate each week of the term (low / medium / high) for how much effort you think you'll be able to put into your project, balancing against deliverables in other courses. Track your actual experience against your prediction. Discuss how your prediction compared to your actual experience and whether you need to change anything going forward.

5.4 Plan to Prototype

1. Learn about different kinds of prototypes in §??
2. Identify key technical and non-technical risks for your project
3. Determine which kind of prototype to use to mitigate each risk
4. Think of any other prototypes that might be helpful
5. Organize the order of building the prototypes into a plan

5.4.1 Different Kinds of Prototypes

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.

5.4.2 Experimental Prototypes

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 R¹ as an experimental prototype for DB/2.² The developers of the seL4 microkernel³ 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

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 5.3 tells the story of the seL4 microkernel as a prototype plan

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

¹ An early relational database engine from IBM Research.

² A commercial relational database engine from IBM.

³ 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

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.

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.

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 students⁴ 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 quiriness of the framework.

5.4.3 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.

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.

⁴ The iLesion group from SE2011. Their project was to make an iPhone app for radiologists diagnosing brain lesions (tumours).

This is more the Pokémon concept of evolution 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 biology, this kind of transformation is called *metamorphosis*.

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

Evolutionary prototypes have been widely advocated in software engineering. Brooks⁵ talks about ‘organic growth’. Gabriel⁶ (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.

5.4.4 Operational Prototyping

Add experimental prototypes to an evolutionary base, then throw the experimental bits away and re-implement them properly within the evolutionary base.⁷ The modern manifestation of this idea is using branches in a version control system to explore new features.

5.4.5 When to build which kind of prototype

<i>Characteristics</i>	<i>Experimental Prototyping</i>	<i>Evolutionary Prototyping</i>	<i>Regular Development</i>
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

⁵ Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, 1975

⁶ 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’

⁷ Alan M. Davis. Operational prototyping: A new development approach. *IEEE Software*, 9(5), September 1992

Figure 5.1: 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

5.4.6 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 they relate to each other. First we introduce some iconography for the different kinds of prototypes discussed above:

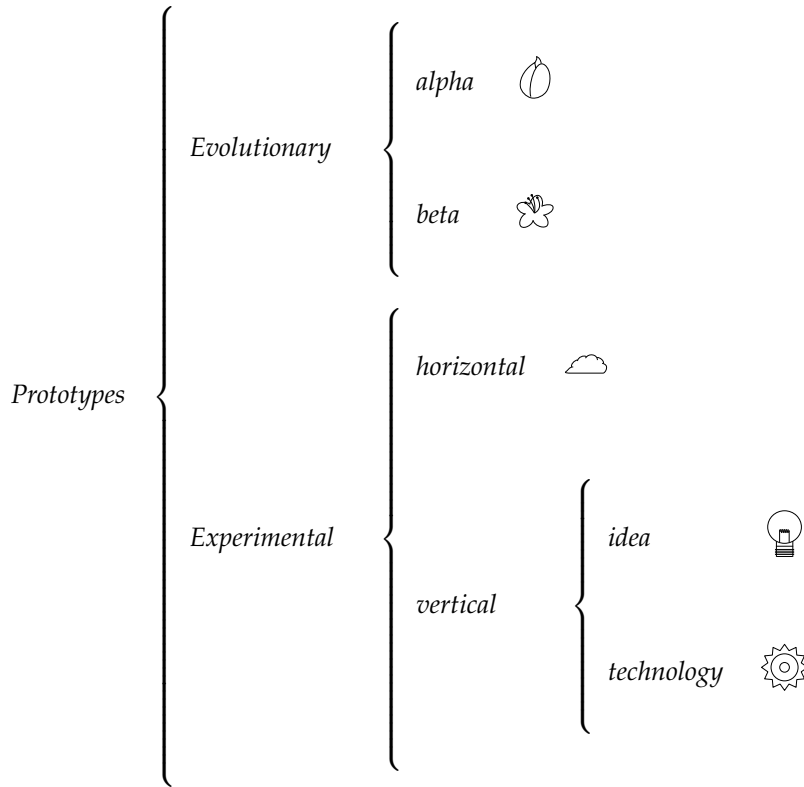


Figure 5.2: Icons for different kinds of prototypes. Icons drawn by Albert T. Yuen (SE2012). Reuse by SE students permitted.

With these icons we can tell the story of the seL4 verified microkernel⁸ (or of your fourth year design project).

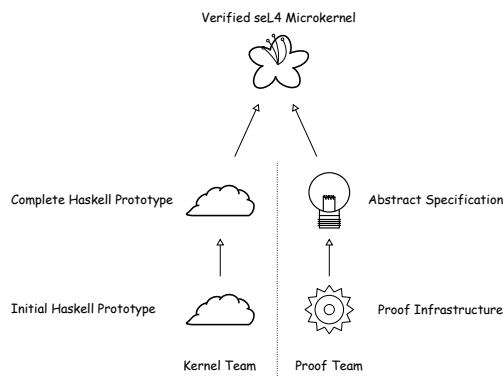


Figure 5.3: The story of the seL4 verified microkernel

⁸ 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

*

5.5 *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 written 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:⁹

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 Poseidon 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.

⁹ 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 (§??) 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 (§2.6) and changing teams (§2.7).

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 (§2.6.6, §??), and Team Kaze (§2.6.7, §??) 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 (come-

dies) and re-tell them in the context of your team.

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* (§5.6) 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.

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?

5.6 Compare to 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. This activity is most appropriate for New Product projects and generally not applicable to others.

Paul Graham, most famously known for his role in Y Combinator

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.

accelerator program, wrote that “smart people have bad ideas”¹⁰, and later, “the best startup ideas seem at first like bad ideas”¹¹. It is difficult to build a successful project around one of these ideas, let alone a business, but some students¹² 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.

¹⁰ <http://paulgraham.com/bronze.html>

¹¹ <http://paulgraham.com/swan.html>

¹² Such projects include Singspiel.ca, a medical records project with sanu.mit.edu, and BlackBoxHealth.

- 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.

- 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

Conceptual Activities

6.1 Problem Identification and Refinement

Project concepts are often conceived with a statement such as: ‘a fridge inventory app to reduce food waste’. This statement includes a vague description of both the problem and the proposed solution. Let’s sharpen them.

6.1.1 Problem-Space Exploration

Identify a broad set of related problems, as well as other possible uses for the proposed solution. Some of these might be out of scope for your project. Identifying what is in scope and out of scope is a valuable exercise both for you and for communicating with your audience.

For our example project, we might consider the various kinds or causes of domestic food waste. We can approach this in two ways: (1) lateral thinking, and (2) reading the research literature, government guidance, advice from non-governmental organizations (NGO), *etc.* Let’s do some lateral thinking about food waste first, to get our brains warmed-up to the topic.

LATERAL THINKING about why domestic food waste occurs:

- Bought too much food to begin with.
- Bought food that they think they should eat, but don’t actually really want to eat. For example, maybe the user bought lots of broccoli, but maybe they don’t actually know how to cook it properly, and they don’t like eating tough + soggy vegetables, so then they let it expire instead of eating it.
- Bought unbalanced groceries. Maybe the user likes chicken+broccoli, but forgot to buy the chicken, and don’t want to eat just broccoli, so the broccoli goes to waste.

An important technique in AI is *near-miss learning*: giving examples that are close to the concept, but not quite it. That’s part of what you are doing here. <http://groups.csail.mit.edu/genesis/papers/2018JakeBarnwell.pdf>

Inexperienced Costco shopper?

Pro tip: peel the broccoli first, then do not overboil it. Overboiling makes the broccoli soggy and unappealing, but does not actually solve the toughness issue nearly as well as peeling.

- Food gets lost at the back of the fridge, behind other food.
- Fridge does not actually get opened. Perhaps the user buys groceries, but then goes out to eat.
- User makes food selections based on mood of the moment rather than prioritizing upcoming expiration dates. They know the broccoli is there, but keep making alternative choices until the broccoli expires.
- Food expires sooner than expected.
- Dislikes eating leftovers.
- User lacks education: best-before dates are different than expiration dates; slightly past-prime fruits are still great in smoothies and baked goods; soaking wilted vegetables in ice-water for 5-10 minutes reinvigorates them.
- User actually forgets what is in the fridge. The knowledge is no longer in their brain. Has Alzheimer's, for example.

PROBLEM RESEARCH. Looking online quickly reveals that domestic food waste is a well-studied topic. That's fantastic: the software design can be informed by and benefit from all of this existing knowledge about the problem of domestic food waste.

- *Household Food Waste—How to Avoid It? An Integrative Review* by Lianne van Geffen, Erica van Herpen, and Hans van Trijp, in the book *Food Waste Management: Solving the Wicked Problem*, published by Springer in 2019 (a scientific publisher). https://link.springer.com/chapter/10.1007/978-3-030-20561-4_2
- *Valuing the Multiple Impacts of Household Food Waste* in the scientific journal *Frontiers in Nutrition*. <https://www.frontiersin.org/articles/10.3389/fnut.2019.00143/full>
- Canadian National Zero Waste Council (a non-governmental organization, NGO). <https://lovefoodhatewaste.ca/about/national-zero-waste-council/>
- City of Toronto public guidance. <https://www.toronto.ca/services-payments/recycling-organics-garbage/long-term-waste-strategy/waste-reduction/food-waste/>
- Toronto NGO dedicated to reducing food waste. <https://tfpc.to/food-waste-landing/food-waste-theissue>
- USA NGO dedicated to reducing food waste. <https://foodprint.org/issues/the-problem-of-food-waste/>

- USA federal government Environmental Protection Agency (EPA) website of public guidance. <https://www.epa.gov/recycle/reducing-wasted-food-home>
- UK NGO report on food waste. https://shiftdesign.org/content/uploads/2014/09/Shift_Food-Waste-insights.pdf

6.1.2 *Solution-Space Exploration*

What other solutions might solve some of the problems? Include both software solutions, software+hardware solutions, and solutions that do not involve computing. As with our problem-space exploration above, we can do this both by lateral thinking and by reading.

LATERAL THINKING about some possible solutions:

- User gets a reward of some sort for actually looking in the fridge to see what is going to expire in the next 2-3 days. Maybe a habit-tracking kind of app, or maybe just a bowl of candy on top of the fridge with the rule that the user can only eat a candy if they inspect the fridge contents.
- Install a camera inside the fridge door, to send a picture to the user's phone. Many manufacturers are doing that kind of thing these days.
- Buy a counter-depth fridge, which is shallower than normal, so it's harder for food to get lost at the back.
- Buy a fridge with the freezer on the bottom, so it's easier to look in to the fridge.
- Add a lazy-susan rotating shelf to the fridge, so things at the back can be spun around to the front.
- Improve shopping habits to match acquisition with consumption.
- Improve meal planning to match acquisition with consumption.
- Don't eat out so much.
- Make choices to eat food approaching expiry date, rather than prioritizing the mood of the moment.

READING ABOUT EXISTING SOLUTION STRATEGIES in the articles and websites listed above. Generate a list of them, and use that list as a starting point for how you can add software improve existing ideas.

6.1.3 *Application-Space Exploration*

What else could be done with the proposed solution? In our example, what other problems could be solved with a fridge inventory app? Lateral thinking:

- reduce trips to the grocery store
- ensure that staples are always in stock
- recipe planning/suggestion

6.1.4 *Technology-Space Exploration*

What is the best technology for the solution, including both hardware and algorithms? For our food inventory example, a major challenge would be data entry. What are some possible technologies that can ease that burden from the user? Again, start with lateral thinking.

- Take a picture of the food as it goes in to the fridge, and say the date it will expire.
- Put a cheap tablet on the door of the fridge, dedicated to running this app. The screen turns on when the door opens.
- Make a voice-assistant app using one of those devices from Amazon/Apple/Google. Many people like talking to technology.
- Scan groceries as user puts them in their cart in the store, instead of when they put them in the fridge at home.
- Consumption data tracking. Knowing what is about to expire is not just about knowing what went in to the fridge, but also about what has already come out of it.
 - Record each time user takes things out of the fridge.
 - Display the list of foods sorted by expiration date, and let the user just check off what they have already eaten. This could be done at a separate time from when one is grabbing food out of the fridge to eat.

6.1.5 *Refinement*

Refine your conception of the problem and your solution based on the insights gained from the above exercises. A key objective here is to ensure that your proposed solution aligns with the problem you are trying to solve. Perhaps you will have a new solution, a new feature to add to your solution, or decide to solve a slightly different problem. Probably you will refine how you explain the alignment of the problem and your solution.

6.2 Identify the Core Conceptual Data Structure

Many projects have a core conceptual data structure. For example, consider an app to share favourite places. Is the core conceptual data structure a list? a set? a bag? Each supports different kinds of usage scenarios. For example, a walking tour of a city or a museum needs to be a list, because there is a clear temporal order to the places, and it could make sense to revisit a place after having learned something important elsewhere. But a 'list' of favourite restaurants is really a set: there would be no intrinsic ordering, and duplicates would be forbidden.

Another example: a personal habit app, based on the idea of *the habit loop*. The habit loop says that a habit has three components: a cue, the activity, and a reward. So, in computer-science-speak, a habit is a three-tuple, and a habit app's core conceptual data structure would be a set of such three-tuples.

Once this conceptual data structure is identified, then discuss different ways the user might interact with it. Consider the habit app, which is a set of three-tuples of $\langle \text{cue}, \text{activity}, \text{reward} \rangle$. Different use cases might be:

- *Destroy an Existing Tuple*: That is, break a bad habit.
- *Discover an Existing Tuple*: Perhaps keep a log to discover the cues that lead a particular behaviour.
- *Create a New Tuple*: Since these are three-tuples, the user might have at least three different starting points, depending which part of the tuple is known. Then the user's task can be conceptualized as a search to fill in the other values of the tuple. Consider:
 - *Known Cue*: Match it up with a new activity + reward. For example, suppose one already goes for a walk in the morning. That can be a cue for an additional activity. Bring along an exercise band to do some shoulder rotations, wrist exercises, *etc.*, while walking.
 - *Known Activity*: Find cues that can be used to trigger the activity.
 - *Known Reward*: Suppose the user wants to buy a new bike. They could change their lunch habits to save money towards that goal.

6.3 Position in Marketspace

Identify one or two dimensions of differentiation in the relevant marketspace, and position your project with respect to existing alternatives. Presumably your project represents a different tradeoff as

compared to existing alternatives, and so populates a new position in the marketplace.

6.4 Strategic Project Positioning

One of the key factors in choosing your project [§??] and understanding project risk [§6.5] 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 [§14.2]. At a high-level, your software must have done something for someone. There are currently four different categories of results to aim for [§14.2]: New Product, Custom Software, Research, and Free/Open Source Software.

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:

1. *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.
2. *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.
3. *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.
4. *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.
5. *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.
6. *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,

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.

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

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.

HiveMind sE2013, Kinectitude sE2013.

A few more ...

- On Trend
- Focused target population
- Underserved Space. *e.g.*, video games for the blind.
- New technology / change in technological landscape.
- New data source. *e.g.*, Riot Games
- Clear influencers in market, open to innovation. For example, teachers might influence students.
- Familiarity with the target user community.
- Novel idea.

6.4.1 Strategic Blunders

AGAINST VESTED INTERESTS: Are there large corporations, unions, social structures, or regulations that the project is likely to come in conflict with? If so, then the project is probably too big for capstone: fighting vested interests takes years.

Let's start with a simple non-software example: flushless urinals. These are obviously a good idea from an environmental perspective. Plumbers unions didn't like them though because they reduced the amount of work that plumbers would get: flushless urinals do not require water supply lines. It took years to come to an agreement that building codes would require water supply lines to be installed behind the wall anyway, in case the building ever switched to flushing urinals. Without agreement on the building code, it was difficult to sell flushless urinals.

Uber is closer to the software space. Uber attempts to disrupt entrenched interests in the taxi industry — often by breaking the law. Maybe the existing laws and regulations around the taxi industry are not fully in the consumer's best interests and need to be updated in light of modern technology. But planning to pick this kind of fight — or planning to break the law — is probably beyond the scope of a capstone project: consider the volume of time, money, lawsuits, and people that Uber is investing in its efforts.

Around 2010 it was popular for SE project proposals to involve sharing digital media files (songs, movies, *etc.*). Those proposals often broke copyright laws, and so students selected other projects. Copyright laws do need to be reformed, but breaking the law on a school project is not the way to achieve that reform. Join an existing advocacy effort.

blunder, noun: a gross error or mistake, resulting from carelessness, stupidity, or culpable ignorance. [Webster 1913]

https://www.wired.com/2010/06/ff_waterless_urinal/

<http://www.michaelgeist.ca/tech-law-topics/copyright/>
<http://www.lessig.org/books/>
<http://www.reformingcopyright.org/>

ACCESS TO DATA:

FREE ENGINEERING LABOUR: If the commercial competitive advantage of your project is that you could charge less to users because you are not being paid for your labour, then it might be time to pause and re-consider. The main exceptions to this are *pro-bono* projects that benefit the public good: FOSS projects, projects for academic environments, and projects for non-profit organizations. Examples of past pro-bono projects include:

- UWFlow
- Chantelle + follow-up
- Blueprint
- Speculative Execution
- eCrypt-FS
- ...

Dynalist, for example, is not undervaluing their engineering labour as a means to compete: they charge market rates (similar to their competitors), and compete on features.

DENSELY POPULATED MARKET SPACE: Are there many competitors already existing in the space? Have many people done this project before? If so, succeeding might require a high degree of both insight in the domain and awesomeness might be required. Some examples of projects that have succeeded in densely populated market spaces include:

- *UWFlow* SE2014
- *MixBot* SE2014
- *Parallax WebGL Debugger* SE2016
- *Dynalist.io* SE2017

6.5 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.

“Pro-bono is a Latin phrase for professional work undertaken voluntarily and without payment. Unlike traditional volunteerism, it is service that uses the specific skills of professionals to provide services to those who are unable to afford them.” https://en.wikipedia.org/wiki/Pro_bono

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. [§2.6.2]

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 [§5.6]. It is often difficult to produce results with these project ideas — although there are notable exceptions, such as UWFlow [§??].

Students often under-estimate risk for popular project ideas.

DISAGREEMENTS ABOUT POTENTIAL. Sometimes the instructor disagrees with the student assessment of the project potential — especially around new products. But professors are not experts at assessing new products and market needs and opportunities. The best way to navigate this issue is to take it out of the realm of opinion by providing substantive arguments and evidence for your opportunity. Basing the case on your opinion alone invites the instructor to have different opinions.

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.

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.

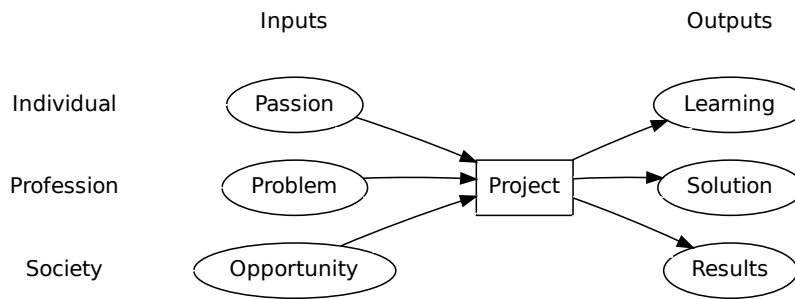
Students usually propose projects of reasonable scope.

For example, in SE2021 Team SEXXI-Goose chose to patch the Rust Compiler, yet nobody on the team had prior experience with Rust. They learned the language and succeeded. You can learn new things and succeed too.

Popular Project Ideas §5.6

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

6.5.1 Project Positioning Assessment



STRATEGIC ADVANTAGES

- passion for project
- clear learning outcomes
- new technology
- new data source
- clear technical problem
- team's background knowledge
- target population:
 - focused
 - underserved
 - clear influencers
 - familiarity to team
- novelty
- awesomeness/commitment

STRATEGIC BLUNDERS

- against vested interests
- access to data
- undervalued engineering labour
- densely populated market space
- significant marketing required
- scope: too small; too large

6.5.2 *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:

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

3. 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.

6.6 OLD: How to Choose a Project

PASSION. Projects that students are excited about tend to succeed. Capstone is a unique opportunity for you 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 [§14.2] 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.

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.

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.

6.7 Position in a Conceptual Framework

Find a conceptual framework relevant to your project and write about where your project fits. Perhaps there is a difference between

Symposium Day Results Rubrics [§14.2]

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.

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.

what your users want and what you can legally provide, for example.

Here is an example spectrum from anonymous to identified; something like this was used on a past project:

- *Unconditional Anonymity*: Anonymity is cryptographically protected. For example, Tor aims to provide this level of anonymity.
- *Revocable Anonymity*: Anonymous so long as rules are followed, but authorities can discover identity if rules are broken. For example, by getting a search warrant.
- *Partial Anonymity*: Anonymous to most others, but perhaps not all. For example, a post in a class discussion forum that is anonymous to classmates but not to the instructors.
- *Pseudonymous Identity*: Like a reddit userid. Not necessarily connected to your actual identity, but something you use repeatedly over time on the same site, and possibly also on different sites.
- *Self-Asserted Identity*: You tell people who you are, but without providing any supporting evidence.
- *Socially-Validated Identity*: Other people vouch for who you are.
- *Verified Identity*: Government issued ID.

6.8 Write a Research Literature Report

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 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.

6.9 Apply Rules of Thumb

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

BROOKS'S LAW¹ (1975) Adding manpower to a late software project makes it later.

BROOKS ON PROTOTYPING² (1975) Plan to throw one away; you will anyhow.

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.

Rule of Thumb: a broadly accurate guide or principle, based on experience or practice rather than theory.

This is part of the accumulated wisdom of our profession. You should be aware of these, and part of developing your professional judgement is knowing when they apply.

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

¹ Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, 1975

² Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, 1975

CAP THEOREM³ It is impossible for a distributed system to provide all three of *consistency, availability, and partition tolerance*.

FALLACIES OF DISTRIBUTED COMPUTING were collected by engineers at Sun Microsystems over many years, including L. Peter Deutsch, Bill Joy, Tom Lyon, James Gosling, and others.

- The network is reliable;
- Latency is zero;
- Bandwidth is infinite;
- The network is secure;
- Topology doesn't change;
- There is one administrator;
- Transport cost is zero;
- The network is homogeneous.
- We all trust each other.

CONWAY'S LAW⁴ (1968) Any piece of software reflects the organizational structure that produced it.

HYRUM'S LAW (2018)⁵

With a sufficient number of users of an API,
it does not matter what you promise in the contract:
all observable behaviors of your system
will be depended on by somebody.

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.

HOARE'S RAZOR⁶ (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.

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

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

³ Wikipedia. CAP Theorem (Brewer's Theorem). URL https://en.wikipedia.org/wiki/CAP_theorem. Retrieved 2016-11-15 https://en.wikipedia.org/wiki/Fallacies_of_distributed_computing

⁴ Melvin E. Conway. How do committees invent? *Datamation*, 14(5):28–31, April 1968. URL <http://www.melconway.com/research/committees.html>

⁵ Hyrum Wright. Hyrum's Law, 2018. URL <https://www.hyrumslaw.com/>. This concept is much older than Hyrum. For example, IBM has been maintaining backwards compatibility for officially undocumented features/bugs on their mainframes for over half a century

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

⁶ 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

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*).

Metcalf was awarded the 2022 Turing Award for contributions to networking. https://en.wikipedia.org/wiki/Metcalfes_law https://en.wikipedia.org/wiki/Moore's_law

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 CRITERIA⁷ (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.

⁷ 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

This idea is the basis of design patterns,⁸ which mostly describe how to organize object-oriented software to manage certain kinds of anticipated future change.

⁸ 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 DESIGN⁹ (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. Significant update in 2021, including a new title *Hints and Principles for Computer System Design*.¹⁰

HOARE'S HINTS ON LANGUAGE DESIGN¹¹ (1973) A classic set of guidelines. Lampson recommends this as complementary reading to his hints. Lampson 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.

6.10 *Position in Normal vs Radical Design*

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

⁹ 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

¹⁰ Butler W. Lampson. Hints and principles for computer system design. URL <https://arxiv.org/abs/2011.02455>. Updated version of the 1983 classic

¹¹ 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 *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_tar_pit

Don't reinvent the wheel

In designing a new machine, an engineer employs familiar components, often in rearranged configurations and occasionally in radically modified ones.¹²

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.¹³

Normal design involves incremental improvement in a product class that is well understood, with known problems and solutions, and customary user interactions. By contrast:¹⁴

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 no 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 6.1 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*,¹⁵ and *design patterns*.¹⁶ 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.

DON'T REINVENT THE WHEEL. Most engineering practice involves normal design, and there are many practical benefits to working

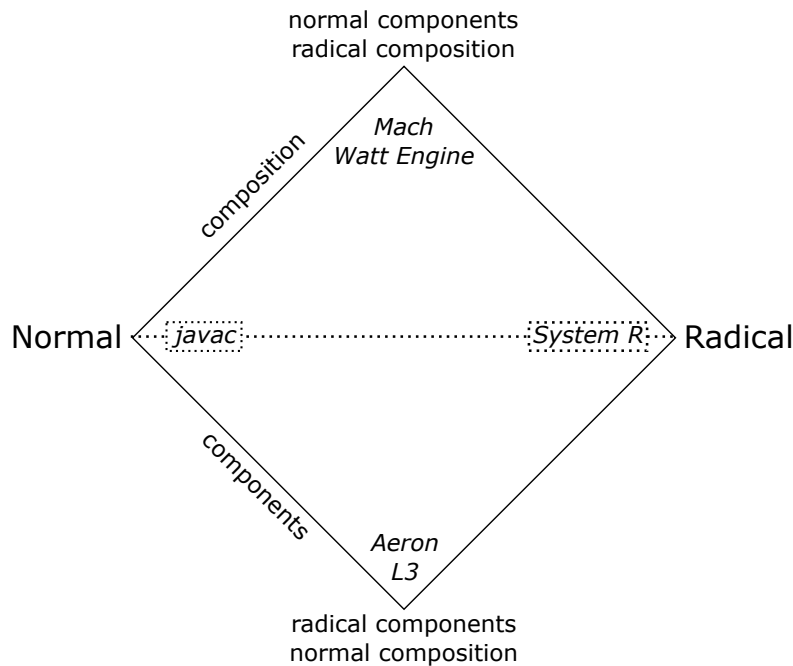
¹² Eugene S. Ferguson. *Engineering and the Mind's Eye*. The MIT Press, Cambridge, Mass., 1992

¹³ Eugene S. Ferguson. *Engineering and the Mind's Eye*. The MIT Press, Cambridge, Mass., 1992

¹⁴ Walter G. Vincenti. *What Engineers Know and How They Know It: Analytical Studies from Aeronautical History*. The Johns Hopkins University Press, 1993

¹⁵ David Garlan and Mary Shaw. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, Inc., 1996

¹⁶ Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995



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 Taylor¹⁷ software architecture textbook uses the term *unprecedented* for what we are calling *radical design*. Our terminology originates with Edward Constant¹⁸, and comes to us by way of Vincenti¹⁹ and Jackson²⁰.

Figure 6.1: 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 Newcomen 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).

¹⁷ Richard N. Taylor, Nenad Medvidović, and Eric M. Dashofy. *Software Architecture: Foundations, Theory and Practice*. John Wiley & Sons, 2009

¹⁸ Edward W. Constant. *The Origins of the Turbojet Revolution*. The Johns Hopkins University Press, 1980

¹⁹ Walter G. Vincenti. *What Engineers Know and How They Know It: Analytical Studies from Aeronautical History*. The Johns Hopkins University Press, 1993

²⁰ 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

6.11 Apply an Idea from the Project Domain

Find an idea from the project domain and apply it to the design of the project.

FOR EXAMPLE, consider a new product project to make an app for helping users towards their life goals. Searching on the internet for 'life goals' turns up several web pages. One of the top ten is written by someone who is both a psychologist and a MBA, starts with a citation, and is rather long.²¹ So probably a good source for project domain ideas. Let's skim to see what ideas are there:

1. 'Goal-setting theory draws on the concept that our conscious ideas guide our actions (Locke, 1968).' Ok, so there is a theory of goal setting. But this article doesn't explain that theory right away, so we'll put it on the back burner for now.
2. 'Goal-setting Can Promote Happiness.' This section mentions the PERMA model of happiness. Again, that idea is explained on another page, so we'll put it on the back burner.
3. Ah, the first figure is Maslow's hierarchy of human needs. That's classic PSYCH101 material, so possibly familiar. Let's go with that.

²¹ Christine Moore. How to set and achieve life goals the right way. Technical report, Positive Psychology, 2020. URL <https://positivepsychology.com/life-worth-living-setting-life-goals/>

<https://positivepsychology.com/goal-setting-theory/>

<https://positivepsychology.com/perma-model/>

<https://www.simplypsychology.org/maslow.html>

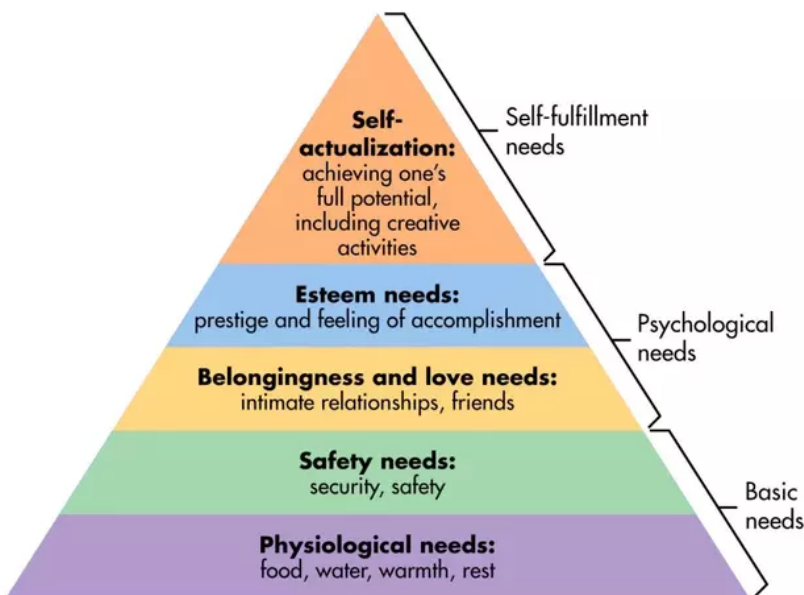


Figure 6.2: Maslow's hierarchy of human needs [image from <https://www.simplypsychology.org/maslow.html>]

APPLYING THE IDEA TO THE PROJECT SCOPE. There are many kinds of life goals. For example, writing a novel, losing weight, getting admitted to university, quitting smoking, getting married, having

kids, climbing a mountain, and so on. Just this small example set is quite diverse. The team could decide to focus on just one part of Maslow's hierarchy, for example: Self-Actualization. That might include goals like writing a novel (for users who are not writers), but would likely exclude the other examples, as in many cases they are more likely connected to other levels of Maslow's hierarchy.

It is important to realize that goals might be at different levels of the hierarchy for different people. For example, writing a novel is always a form of creative expression, but is probably more in the Esteem level for someone who is already a published author, and perhaps more in the Self-Actualization level for someone in a STEM line of work.

The team might decide that the project scope is supporting Self-Actualization life goals for users with successful STEM careers, who have always experienced intense external pressure at the Esteem level. Or the team might decide on a different scope. The point is that the a project domain idea (Maslow's hierarchy of human needs) is being applied to define a scope. Perhaps they could also define the project using the other ideas from this example model, such as Locke's theory of goal setting or the PERMA model of happiness.

APPLYING THE IDEA TO THE PROJECT DESIGN. Perhaps the team has an idea that users can give each other 'stickers' as a way to encourage each other towards their life goals. *Design question:* should everyone see what stickers have been given to which posts?

Answer from idea application: Stickers are a form of Esteem goal realization. But Esteem goals are out of scope for this project, and so might distract the users from their Self-Actualization goals (remember, the target user group are setting these Self-Actualization goals in order to try to surmount the constant external pressure to achieve Esteem goals). So making the stickers private might help the users still give and receive support with less temptation to turn this app into yet another social media Esteem project. In that vein, perhaps each post may receive at most one sticker, or at most three stickers, to put another limit on Esteem-seeking distraction within the app.

There are many other aspects of this example project design that Maslow's hierarchy could be applied to.

6.12 Apply Cognitive Bias Understanding

Learn about cognitive biases and apply that knowledge to some aspect of your project, such as: your understanding of the project; your user's use of the software, etc.

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

<https://uwaterloo.ca/knowledge-integration-exhibition/ki-x-2017/cognitive-casino>

<https://www.verywellmind.com/what-is-a-cognitive-bias-2794963>

7.5 Hypothesis Testing

From Prof Jo Atlee

Most of the information in your Lean Canvas are hypotheses. Among these hypotheses identify five (5) that reflect the riskiest parts of your plan regarding

- your target customer segments (who are your early adoptors, do they care about the identified problems)
- the customers' problems (what their most significant problems are, how important are they), and
- your competition (how do your customers solve their problems now, and is their current solution good enough)

For each hypothesis:

1. Construct a pass/fail test (falsifiable hypothesis).
2. Interview at least five members of the relevant customer segment – enough to suggest whether your hypothesis holds or not (no need to establish statistical significance). Provide anonymized data about the customers interviewed (e.g., type of stakeholder, demographics).
3. Provide a clear and professional summary of the results of the test (e.g., percentages of responses per test answer, insights learned)
4. If your initial hypothesis is false, provide a pivot hypothesis.

7.6 Identify User's Emotional Objectives

We are often focused on user's functional objectives. But in many circumstances emotional objectives drive decision-making.

- What is the user hoping to feel from use of the software?
- What relationships will be strengthened?
- What will use of the software tell the user about their self-concept?

The automobile industry, for example, has a strong messaging here: this car will make me feel rich; this car will make me feel like an adventurer; *etc.*

For example, consider a website to facilitate peer resume critiques. The functional objectives are around actually doing the resume critiques. But the emotional objective for the user is to *reduce anxiety* around the job search process. What else can the software do to help achieve this emotional objective?

7.7 Practice Decoding Analogies/Metaphors

Decoding analogies is an important skill for communicating with clients because they will often explain what they think they want using an analogy. For example, a client might say that they want

software 'like Garage Band' because Garage Band is something that they are familiar with. But their project domain might not be music.

You might have practiced decoding analogies for the SAT or SSAT tests. It might be a good idea to do some of those practice problems now.

Additionally, decoding client analogies in software engineering requires identifying places where the analogy doesn't hold. Literary reasoning tends to focus on the parts of the analogy that work, rather than the parts that don't work.

Practice decoding analogical descriptions of software requirements given by clients.

Design Activities

8.1 Describe Your Architecture

Describe your architecture using an appropriate technique, which might be a diagram. Justify why you take a *structure-oriented*, *decision-oriented*, or *communication-oriented* approach to describing your project's architecture.

The Software Engineering Institute at Carnegie Mellon University has catalogued over 20 different definitions for software architecture.

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

8.1.1 Structure-Oriented Definitions of Software Architecture

Architectural styles and design patterns are examples of structure-oriented descriptions of software architecture. Some important structure-oriented definitions of software architecture include:

- *Garlan & Perry*:¹ 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*:² 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, Clements, & Kazman*:³ 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.

¹ David Garlan and Dewayne E. Perry. Introduction to the special issue on software architecture. *IEEE Transactions on Software Engineering*, 2(1), April 1995

² 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/>

³ Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley, 2 edition, 2003

The industrial designer and architect Charles Eames⁴ made a clearer and more concise statement of this kind of definition several decades earlier:

⁴ Bill Moggridge. *Designing Interactions*. The MIT Press, Cambridge, Mass., 2007 p.648

- *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.

8.1.2 Decision-Oriented Definitions of Software Architecture

Decision-oriented definitions are gaining prominence in systems engineering (e.g., Koo & Simmons⁵) and are occasionally found in software. For example, the Taylor *et alia*⁶ software architecture textbook gives the following pleasingly concise definition:

- *Taylor et al. [58]:* A software system's *architecture* is the set of principal design decisions made about the system.

Why are the advantages of a decision-centric view of software architecture? Structure-oriented definitions are rooted in an analogy to the physical world, whereas decision-oriented definitions are inherently conceptual. Software is a conceptual, rather than physical, artifact. 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.

8.1.3 Communication-Oriented Definitions of Software Architecture

Conway's Law is an important idea in software engineering:

- *Conway⁷ 1968:* organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations

When computing was brand new there weren't software companies. Software was developed in companies that were organized around other concerns, such as manufacturing or services or regulation or historical practice. Conway's Law arose out of an observation of the software produced by these companies. But Conway's Law is still true today. What we have learned is to change how we organize our software engineering teams in order to produce the software we want.

8.2 Extract & Analyze Your Architecture

Use an architecture extraction tool to analyze your code and extract an architecture diagram. Check that the extracted architecture matches your expectations:⁸

- Are there nodes missing?
- Are there unexpected nodes?
- Are there edges missing?
- Are there unexpected edges?

⁵ 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

⁶ Richard N. Taylor, Nenad Medvidović, and Eric M. Dashofy. *Software Architecture: Foundations, Theory and Practice*. John Wiley & Sons, 2009

⁷ Melvin E. Conway. How do committees invent? *Datamation*, 14(5):28–31, April 1968. URL <http://www.melconway.com/research/committees.html>

Doxygen is a well-known open-source architecture extraction tool. There are many others.

⁸ Gail Murphy, David Notkin, and Kevin J. Sullivan. Software reflexion models. *IEEE Transactions on Software Engineering*, 27(4):364–380, 2001

8.3 *Apply Formal Methods*

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 systems, network protocols, or other projects requiring precise correctness criteria.

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>

8.4 *Apply UI Design Guidelines*

Select an appropriate set of user interface guidelines and apply them. There are user interface guidelines published by free/open-source organizations, corporations, and even government. Here are some examples:

- <https://www.microsoft.com/design>
- <https://www.microsoft.com/design/fluent/>
- <https://developer.apple.com/design/human-interface-guidelines/>
- <https://material.io/design>
- <https://developer.android.com/design>
- <https://www.ibm.com/design/language/>
- <https://polaris.shopify.com>
- <https://www.atlassian.design>
- <https://design.firefox.com/photon/>
- <https://hig.kde.org>
- <https://developer.gnome.org/hig/stable>
- <https://www.usability.gov>

8.5 *Incorporate Privacy by Design*

Privacy By Design is a systems engineering approach developed by Dr Ann Cavoukian, former Information & Privacy Commissioner of Ontario. It was adopted by the International Assembly of Privacy Commissioners and Data Protection Authorities in 2010, and is incorporated into the European GDPR (General Data Protection Regulation). It has seven foundational principles:

1. Proactive not Reactive; Preventative not Remedial
2. Privacy is the Default Setting
3. Privacy Embedded into Design
4. Full Functionality — Positive-Sum, not Zero-Sum
5. End-to-End Security — Full Lifecycle Protection
6. Visibility and Transparency — Keep it Open
7. Respect for User Privacy — Keep it User-Centric

Incorporate Privacy by Design into your project.

In September 2019 SE students organized CitizenHacks.com, a privacy-oriented hackathon with Dr Cavoukian as invited keynote speaker.
https://en.wikipedia.org/wiki/Privacy_by_design

<https://www.ipc.on.ca/wp-content/uploads/resources/7foundationalprinciples.pdf>

8.6 Peer Design Exploration

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.

8.7 Peer Design Review

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:

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

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

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

$\frac{1}{3}$ 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.

$\frac{1}{3}$ 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?

8.8 Select a Database Technology

Many projects use some kind of database technology. There are many kinds of database technology. Determine which one best meets the technical needs of your project. Database technologies vary on several dimensions, including:

- *Data Model:* relational, key/value, hierarchical, graph, object, time-series, *etc.*
- *Workload:* kinds, frequencies, and sizes of reads, writes, and queries.
- *Storage Model:* row-oriented, column-oriented, other
- *Concurrency Control:* Most database engines provide some kind of concurrency control, although not all provide full ACID support. Some, such as SQLite quite intentionally have no concurrency control.
- *Distributed vs. Centralized.*

TECHNICAL NEEDS are not the only technology selection criteria in engineering projects. Other factors might include licensing, availability of labour, alignment with other organizational standards, *etc.* However, you should be able to identify which database technology is most appropriate for your project's technical needs, even if you select something else for non-technical reasons.

8.9 Apply (or Reject) the UNIX Design Philosophy

The UNIX design philosophy has been summarized in the following three points:

- Write programs that do one thing and do it well.
- Write programs to work together.
- Write programs to handle text streams, because that is a universal interface.

The term NoSQL is new-ish, but the concept of hierarchical databases predates commercial SQL databases by about twenty years. One of the most successful software products of all time is IBM's IMS, which is still a widely used enterprise product, and started development in the 1960s for the Apollo space program. https://en.wikipedia.org/wiki/IBM_Information_Management_System

The 1973 Turing Award went to Charles Bachman for foundational work on networked/hierarchical databases (which is the way that many people use MongoDB today). https://amturing.acm.org/award_winners/bachman_9385610.cfm

SQL databases weren't acknowledged by the Turing Award until 1981, with Ted Codd's award. https://amturing.acm.org/award_winners/codd_1000892.cfm

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

Dan Luu has some interesting discussion and commentary <https://danluu.com/cli-complexity/>

There are contexts where it can make sense to explicitly reject aspects of the UNIX design philosophy, and certainly important thinkers in software engineering (such as Don Norman) have done so.

Think about your project, whether you are following the UNIX philosophy, and whether you should. Compare your project to other well-known example projects and whether they followed the UNIX philosophy or not.

8.10 Read a Book on Approaches to Software Design

Take a look at a book about software design, and discuss in the context of your project. One good approach is that each team member skims several chapters, so that collectively you have examined the entire book. The books listed here are a starting point; you might find other interesting books worth reading.

- *Clean Architecture: A Craftsman's Guide to Software Structure and Design* by Robert C. Martin
- *Design It!: From Programmer to Software Architect* by Michael Keeling
- *A Philosophy of Software Design* by John Ousterhout
- *Software Design Decoded: 66 Ways Experts Think* by Marian Petre et alia
- *Software Designers in Action: A Human-Centric Look at Design Work* by Marian Petre et alia
- *Designing Software* by Alex Baker et alia

Testing Activities

9.1 Assess Testing

Assess the current state of your project's testing. This might be as simple as saying 'needs improvement'. Or it might be more sophisticated. For example, measuring coverage (of lines, branches, methods, input space, *etc.*).

Assessment could also categorize tests by scope: unit tests, functional tests, system tests, integration tests, regression tests, *etc.*

It's pretty unsatisfying to read a report that says "OK, our state is 'needs improvement'." If your testing is not as good as you would like it to be, you should also think of the best next steps to improve your testing.

9.2 Create More Manual Tests

Create more tests, according to some identified goal(s).

9.3 Identify Invariants

Invariants are properties that should always be true (with the possible exception of intermediate states during what should be atomic updates). For example, that a particular pointer should never be null, or that some numeric value should always be greater than zero, *etc.* Data structures often have invariants. For example, in a red/black tree, the nodes must alternate between red and black.

Once invariants are identified, then assertions can be added to check them. This way one gets more value out of an existing library of test inputs.

9.4 Identify Mathematical Properties

Sometimes parts of a program should have mathematical properties. For example, an *equals* method should define a mathematical

equivalence class, which should be *reflexive*, *symmetric*, and *transitive*.

Two other general properties that can be useful in testing are *inverse functions* and *idempotence*. For example, parsing and pretty-printing are inverse functions. Idempotence means that if the function is applied repeatedly it produces the same result as if applied only once. For example, data synchronization should be idempotent: after the first synchronization, if nothing has changed, then the second synchronization shouldn't change any of the data.

9.5 Use Automated Test Input Generation Tools

There are now a variety of automated test input generation tools, which tend to fall in to three main categories:

Systematic: Generate all inputs that meet a general description. For example, all non-isomorphic trees up to size five.

e.g., Korat

Random: Generate random inputs, or random sequences of method calls. Systematic techniques tend to be bounded to small inputs, whereas random testing can create larger inputs.

e.g., Randoop

Fuzz: Generate garbage. Garbage inputs should not cause the program to crash.

9.6 Use A Linter

A linter is a tool that scans your source code for problems that are more sophisticated than what a compiler typically finds, but less sophisticated than what an automated test-generation tool might find. Most mature languages have linters available. For this (and analogously for other activities in this section), report what happens once you incorporate the linter (or other tool) into your CI workflow. Ideally, you incorporate the linter early and you use it for a few weeks. Then you report what you find after you've been using it. This only works if you add enough code during your linting period to make linting worthwhile. Do you find friction from using the linter, as a developer? Is using the linter tolerable? Do you have a huge pile of linter issues to fix when you initially add the linter?

SE2016 Team Sleekbyte wrote a linter for the Swift language.

9.7 Test Against an Alternative Implementation

Find, or create, an alternative implementation to test against. This technique is particularly applicable when the goal of your project is improved performance. The faster and more sophisticated algorithm should still compute the same result as the simpler and slower algorithm.

9.8 Set Up Continuous Integration

Set up a continuous integration system.

9.9 Set Up Deployment Environments

Separate development, testing, and production environments. Or Blue/Green deployment (two production environments, where only one handles live traffic), rolling deployments, etc.

<https://opensource.com/article/17/5/colorful-deployments>
<https://rollout.io/blog/rolling-deployment/>

9.10 Statistical Cross-Validation for Machine Learning

Statistical cross-validation is a group of techniques that can be used to evaluate how well a machine-learning system will be able to make correct conclusions on unseen data.

<https://machinelearningmastery.com/k-fold-cross-validation/>

A popular technique is *k-fold cross-validation*. Suppose you have 1000 labelled data-points, and you want to do a 5-fold validation ($k=5$). Randomly divide the data-points into 5 groups of 200 data-points each. Do 5 rounds of evaluation, one for each group. In each round of evaluation there will be 1 test group and 4 training groups. Train the model on the training groups, evaluate on the test group, then discard the model before the next round. Report the 5 results.

[https://en.wikipedia.org/wiki/Cross-validation_\(statistics\)](https://en.wikipedia.org/wiki/Cross-validation_(statistics))

9.11 Performance Profiling

Measure the performance of your program. Where are the real bottlenecks? If you don't measure it, you cannot focus your efforts appropriately. For example, tweaking the algorithm won't matter if the time is actually dominated by object serialization.

Michael A. Jackson has famously coined the first two rules of program optimization:

1. Don't do it.
2. (For experts only!): Don't do it yet.

9.12 Scalability Assessment and Planning

Programmers often like to think about the future, when there will be great demand for the programs they have written and scalability will be a concern. The first step is to do an assessment of the current design: how many users is it likely to support before having scalability concerns? Is there a load tester that can be used to simulate increased demand to provide empirical evidence for the estimate?

The next step is to make a plan for possible alternative designs that would be more scalable. Implementing one of these alternative plans should probably be deferred until there is a good engineering case that the demand exists, or will soon exist, or the technical debt of the current design will become a problem itself.

9.13 *Measure Precision and Recall*

Precision and *recall* are common efficacy measures for information retrieval and machine learning systems. A system with perfect precision returns no false positives. A system with perfect recall returns no false negatives (*i.e.*, finds everything of interest). The challenge is to do well on both of these measures. Simply returning everything will have perfect recall but terrible precision. By contrast, a system that returns nothing will have perfect precision but terrible recall.

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

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

This chapter is based on notes taken by Beverly Vaz SE2021 from the s20 offering of CS449 taught by Edith Law.

User-Centred Design (from CS449)

User-Centred Design is an approach to product design, including software product design, that considers the user's perspective at each step of the design process. Importantly, UCD involves validating the user experience through engaging with actual users. This chapter summarizes one approach to UCD, as taught in CS449.

TYPES OF OBSERVATIONS that a *researcher* might make of *users*:

Non-participatory: A detached observer is observed without them knowing they are under observation. Done in public spaces.

Passive participation: The researcher is known and participants know the research goals

Active Participation: The researcher is fully engaged with the participants as they are friends.

Complete participation: The researcher is fully embedded as a spy

10.1 Value Proposition

Ideally, the design and development of any idea should start with creating a *value proposition*.

A value proposition is a laser sharp description of the service you intend to provide and should ideally be summarised in a single sentence.

It should answer one of the following questions:

- What does your product do?
- How does it work? or,
- What does it feel like to use your product?

https://en.wikipedia.org/wiki/User-centered_design

Example CS449 project applying these techniques:
<https://medium.com/gomeet/cs-449-design-process-of-gomeet-d8ec35997f99>

Researcher here means someone who is studying (researching) the target user population.

10.2 Persona Empathy Map

With these defined, the next step would be to draw the *persona-empathy map*. (This is referred to in the handbook as **'Persona building'**).

First, you need to identify the product's user group, i.e, the general description of the target user based on certain characteristics.

Next, you should come up with specific descriptions of target users that you draw from your user group. This would be the **personas** you create. These will be fictional characters that represent what a real target user would be like.

This should be followed up with the creation of **empathy maps**. Empathy maps represent what the persona thinks, feels, experiences and wants.

Keep in mind: When you create personas and empathy maps, you are making assumptions about your end-users. These assumptions need to be tested in the field.

To give you an idea of what a persona and empathy map should look like, here's an example:

<https://www.nngroup.com/articles/empathy-mapping/>

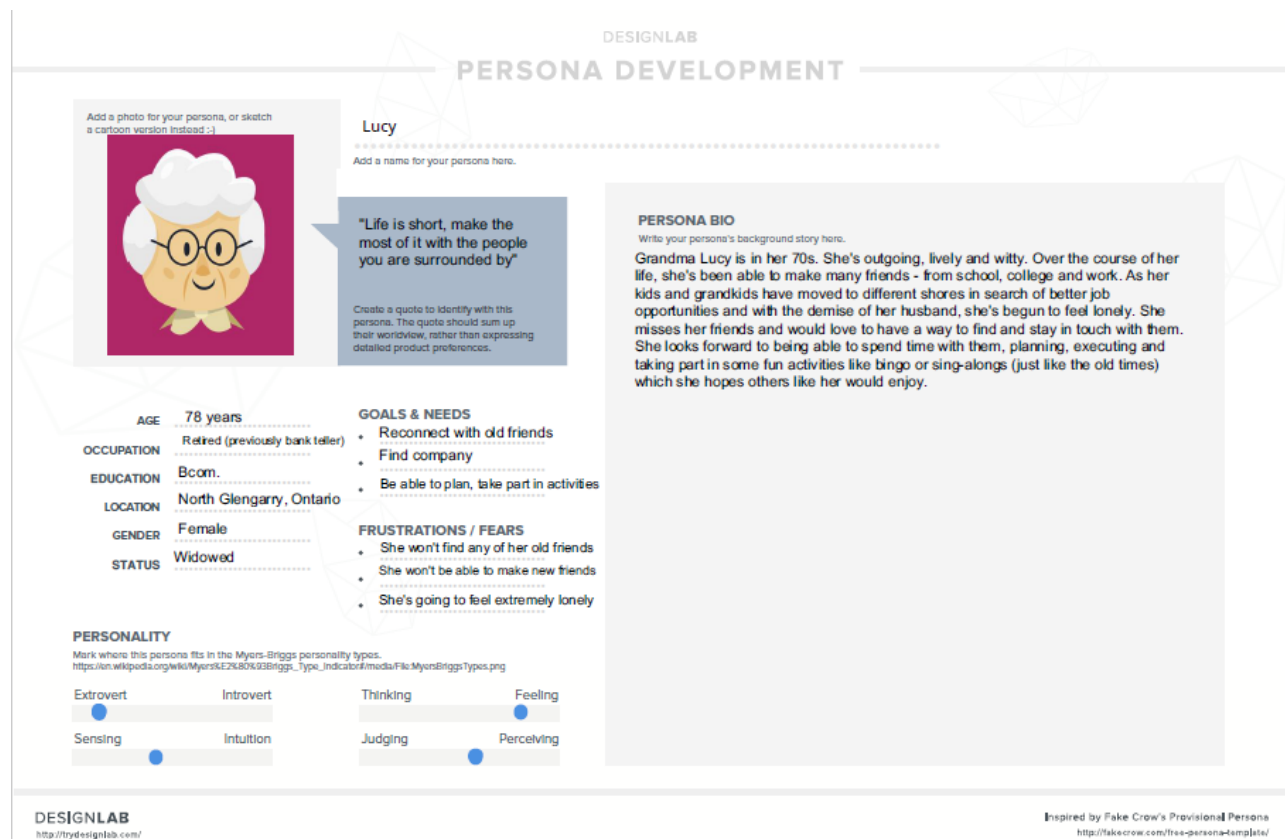


Figure 10.1: Example of what a persona looks like

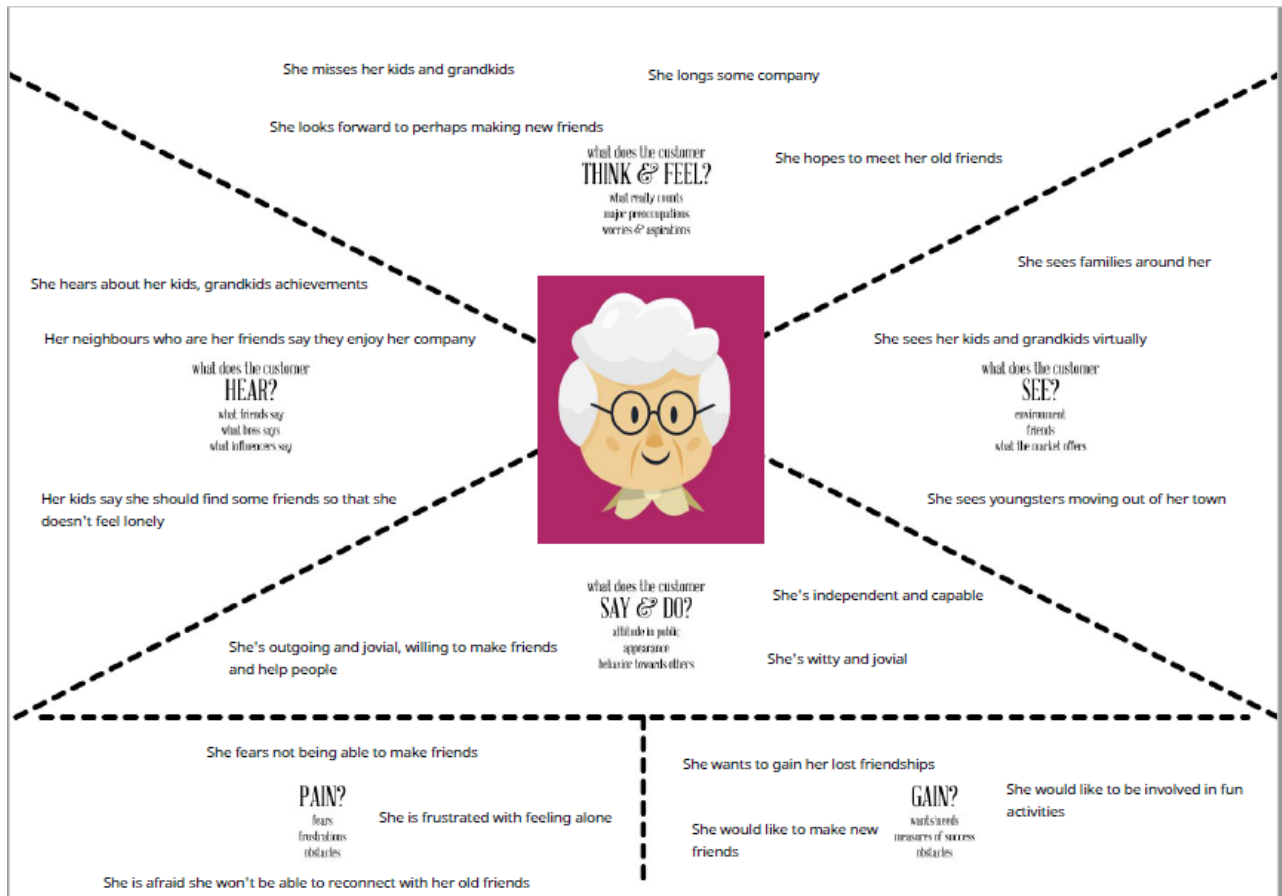


Figure 10.2: Example of what an empathy map looks like

10.3 *Gather Data*

In order to test your assumptions, you need to gather data. There are 2 ways to go about this.

1. Quantitative Methods

These are methods used to collect a large amount of data and are listed under 'Moran's 9 Quantitative User Activities' in the handbook.

2. Qualitative Methods

These are methods used to capture rich descriptions of people and phenomena.

There are 3 main ways this is done. Through:

- Observations

Can be:

- Controlled - performed in a lab
- Naturalistic - observed at home, on a bus

See **10** for types of observations.

- Interviews

(This is mentioned in the handbook as User Interviews).

An interview has 5 main parts:

(i) Introduction

Seek consent and inform the interviewee about what's coming up and the purpose of the interview

(ii) Kickoff

Ask non-threatening questions to get to know the interviewee

(iii) Building Rapport

Ask questions to get a detailed understanding of the interviewee

(iv) Grand Tour

Let people tell their story

(v) Reflection/Wrap up

Summarize the main things learnt and thank the interviewee

- Focus Groups

These are great for open questions and story sharing. It helps explore user's attitude, opinions, expectations and reactions.

10.4 *Analyze Data*

After collecting data, it's important that you analyze it, draw some findings and reiterate on your design. This can be done with the

help of *affinity diagramming*. The affinity diagram will help identify features your product should contain.

Method Go through the notes made while collecting data and write some key points on sticky notes. Arrange the notes in a hierarchy using a bottom-up process to reach a single overarching theme that covers all notes. The themes and sub-themes are initially unknown. This would reveal issues common across all users.

Here's an example of what an affinity diagram would look like.

Source: Payton, Talisha (February 8, 2016), *Learning UX, Affinity Diagrams: Tips and Tricks*; Retrieved from <https://medium.com/learning-ux/affinity-diagrams-tips-and-tricks-6225e8c1f0df>

For each feature that's been decided, a *design argument* must be written. A design argument is a testable hypothesis of why characteristics of a design will help overcome an obstacle to achieve a desired outcome.

Once you have the design arguments done, you may proceed with creation of *user stories*. (This is mentioned in the handbook as **User Studies**.) These describe functionality that will be valuable to the user.

10.5 Crazy 8s

Once you have the features planned out you can perform *Crazy 8*. (This is referred to in the handbook as **Crazy8** under Creative Activities). In this activity you draw 8 sketches of different possible layouts and screen organizations for the feature.

Here's an example of what a Crazy 8 activity output would look like.

Follow Crazy 8 with *storyboarding*. Storyboarding is where you illustrate the use of a feature. Display the background/context, emotion and accomplishments a user should experience with the feature.

Draw the screens required for the feature. These are called *sketches*. Illustrate the path a user would follow when they use the feature and the screens. This is called *user flow*.

10.6 Low-Fidelity Prototyping

Once you have the screens drawn out, you can conduct a *low-fidelity prototype evaluation*. Low-fidelity prototype evaluations are expected to be low on technology, have partial functionality and the interaction is simulated. This can be achieved through a paper prototype evaluation.

During the paper prototype evaluation, you want to get users to try out the screens by making them believe it is an actual application

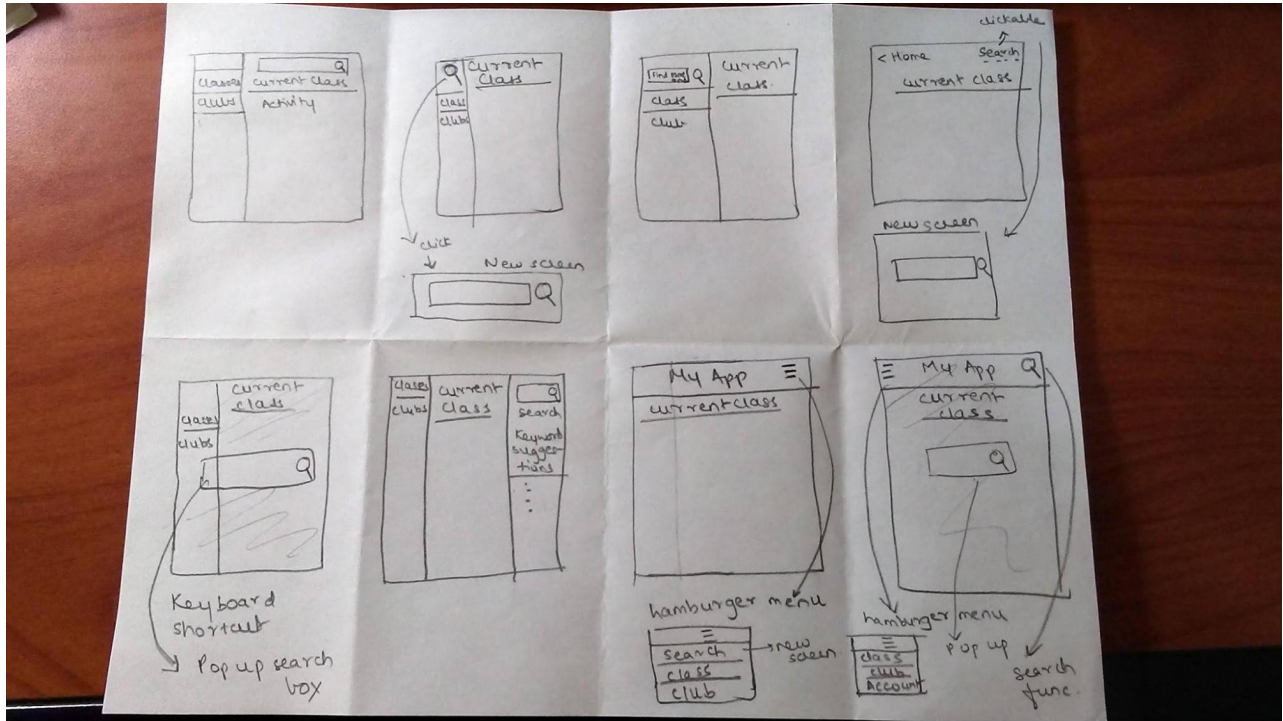


Figure 10.4: Image of Crazy 8



Figure 10.5: Image of sketch

on a phone. You manually switch between screens for the user when they click a button on the screen (this is the simulated interaction being referred to).

You need to perform the following steps:

- Identify testing goals
- Identify the items to test
- Choose testers
- Prepare the material - tasks, things to observe, questions to ask the evaluator
- Based on what needs to be done, assign roles - who observes, who simulates interaction
- Run the evaluation
- Analyze information obtained from the evaluation

Based on your findings, you might need to refine your feature.

10.7 *High-Fidelity Prototyping*

You can next move to *high-fidelity prototyping*. Typically a tool like Figma is used to create the screens and connections are made between the screens so you can make the interaction seem true and almost like what a user would experience if they actually interacted with your application. At this stage, color schemes and logos make a difference and have to be decided upon.

To evaluate the high-fidelity prototype, you can conduct *heuristic evaluations* and *cognitive walkthroughs*.

Both follow goal-based tasks. However, in a heuristic evaluation, the evaluators are experts and assess the tasks based on Nielsen's 10 Usability Rules of Thumb (mentioned in the handbook).

In a cognitive walkthrough, a typical user is picked and they are observed as they go about performing the tasks. An ideal way of performing the task is listed out by the designer and is referred to as the success story. The user's actions are compared against the success story and inferences are made. It's useful for judging things like 'Is adequate feedback provided to the user?'

User Activities

11.1 Take TCPS2 Training: Ethical Conduct with Users

The Canadian Tri-Councils have a framework and training program for *ethical conduct for research involving humans* called TCPS2. Your user activities are not necessarily ‘research,’ but you should still do this training if you wish to conduct serious user activities.

This TCPS2 training is mandatory for all students in CS449, and also for all researchers in Canada who work with human subjects — including software engineering researchers.

1. Take the training: <https://tcps2core.ca/welcome>
2. Identify relevant parts of the TCPS2 framework.
3. Apply the TCPS2 framework to your proposed user activities.

11.2 Do a User Activity

There are dozens of activities that you can do with users. The Nielsen/Norman Group is a leader in this area, and has several good overview articles to help you select activities that are appropriate to for each stage of your project. Most of the rest of this section provides you pointers to those Nielsen/Norman Group articles.

11.3 Ideate about Possible User Activities that You Might Do

Generate a list of 5–10 ideas of user activities that you might do at some future point. For each kind of activity, generate at least one (but preferably two or three) different ways in which you might apply that activity in your project. For example, there are multiple ways one might use A/B testing.

The Tri-Councils are: NSERC (National Science and Engineering Research Council), SSHRC (Social Sciences and Humanities Research Council), and CIHR (Canadian Institutes of Health Research).

https://ethics.gc.ca/eng/policy-politique_tcps2-eptc2_2018.html

Courses like CS449, MSCI343, SYDE548 teach you about these activities.

11.4 *Formative vs. Summative Evaluations*

STEM courses are often focused on *summative* evaluations, like exams. Summative evaluations are often quantitative or comparative, and they typically occur infrequently. Summative evaluations can also be done for designs:

“Summative evaluations describe how well a design performs.”¹

Formative evaluations are uncommon in STEM courses, but they are essential both for students in open-ended design courses and for designs:

“Formative evaluations focus on determining which aspects of the design work well or not, and why.”¹

Formative evaluations should occur more frequently than summative evaluations in open-ended design situations, to ensure that the problem definition is aligned with the user needs, and that the proposed solution strategy is aligned with the problem definition. Similarly, in design courses, students benefit from formative evaluations to get feedback to keep them on track.

¹ Alita Joyce. Formative vs. summative evaluations. Technical report, Nielsen/Norman Group, 2019. URL <https://www.nngroup.com/articles/formative-vs-summative-evaluations/>

The terms *formative evaluation* and *summative evaluation* actually originated in education theory, by Michael Scriven in 1967. They were then later adopted by designers to talk about evaluating designs.

11.5 Rohrer Survey of 20 Different User Activities

Rohrer² organizes 20 different user-research methods along 3 axes:

- Attitudinal vs. Behavioral
- Qualitative vs. Quantitative
- Context of Use

² Christian Rohrer. When to use which user-experience research methods. Technical report, Nielsen/Norman Group, 2014. URL <https://www.nngroup.com/articles/which-ux-research-methods/>

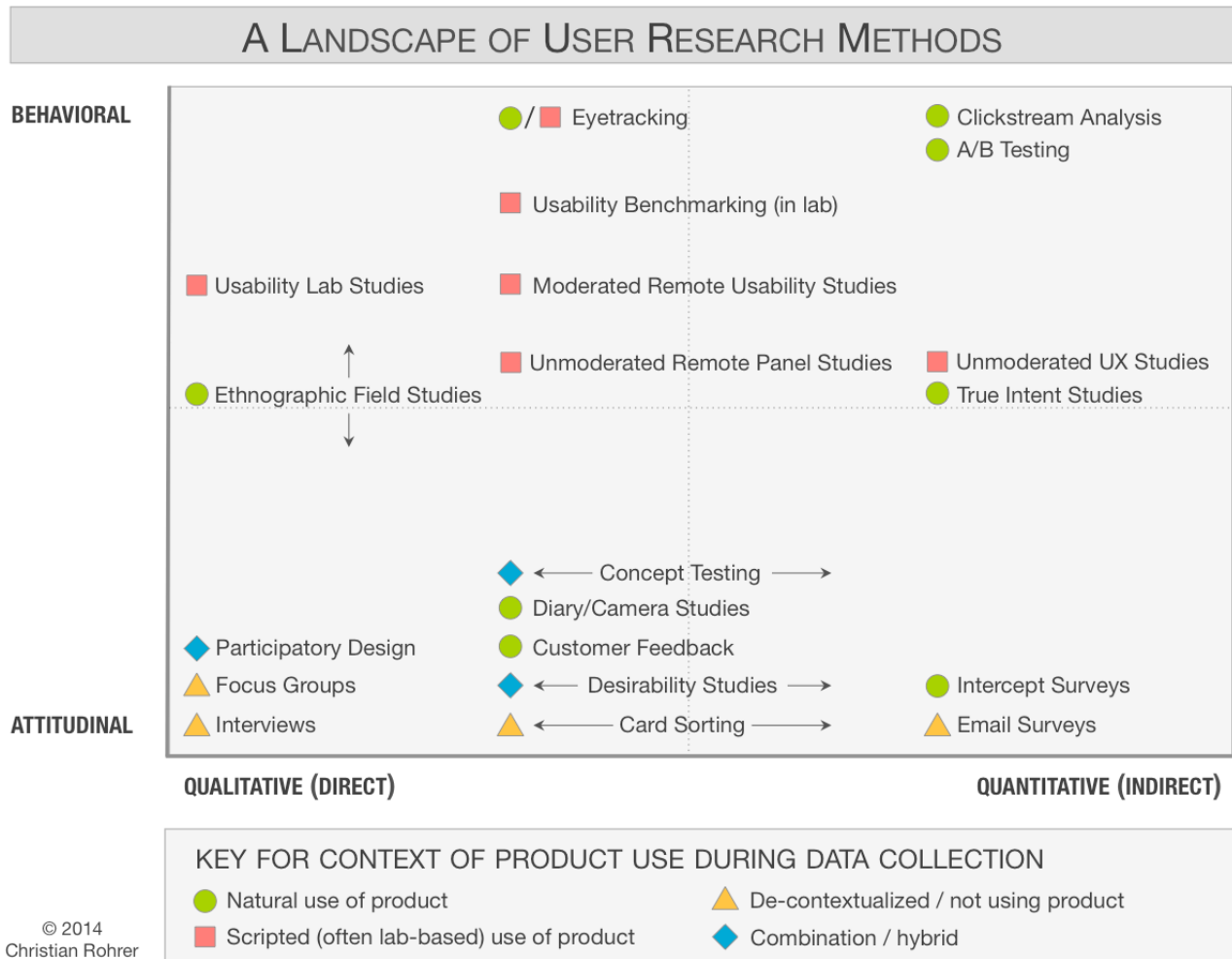


Figure 11.1: Central figure from Rohrer’s user-activity survey article for the Nielsen/Norman Group.

11.6 Farrell's Survey of 34 User Activity Methods

Farrell³ surveys 34 user activity methods, organizing them around which phase of the design cycle they are most appropriate for.

³ Susan Farrell. UX research cheat sheet. Technical report, Nielsen/Norman Group, 2017. URL <https://www.nngroup.com/articles/ux-research-cheat-sheet/>

UX ACTIVITIES IN THE PRODUCT & SERVICE DESIGN CYCLE

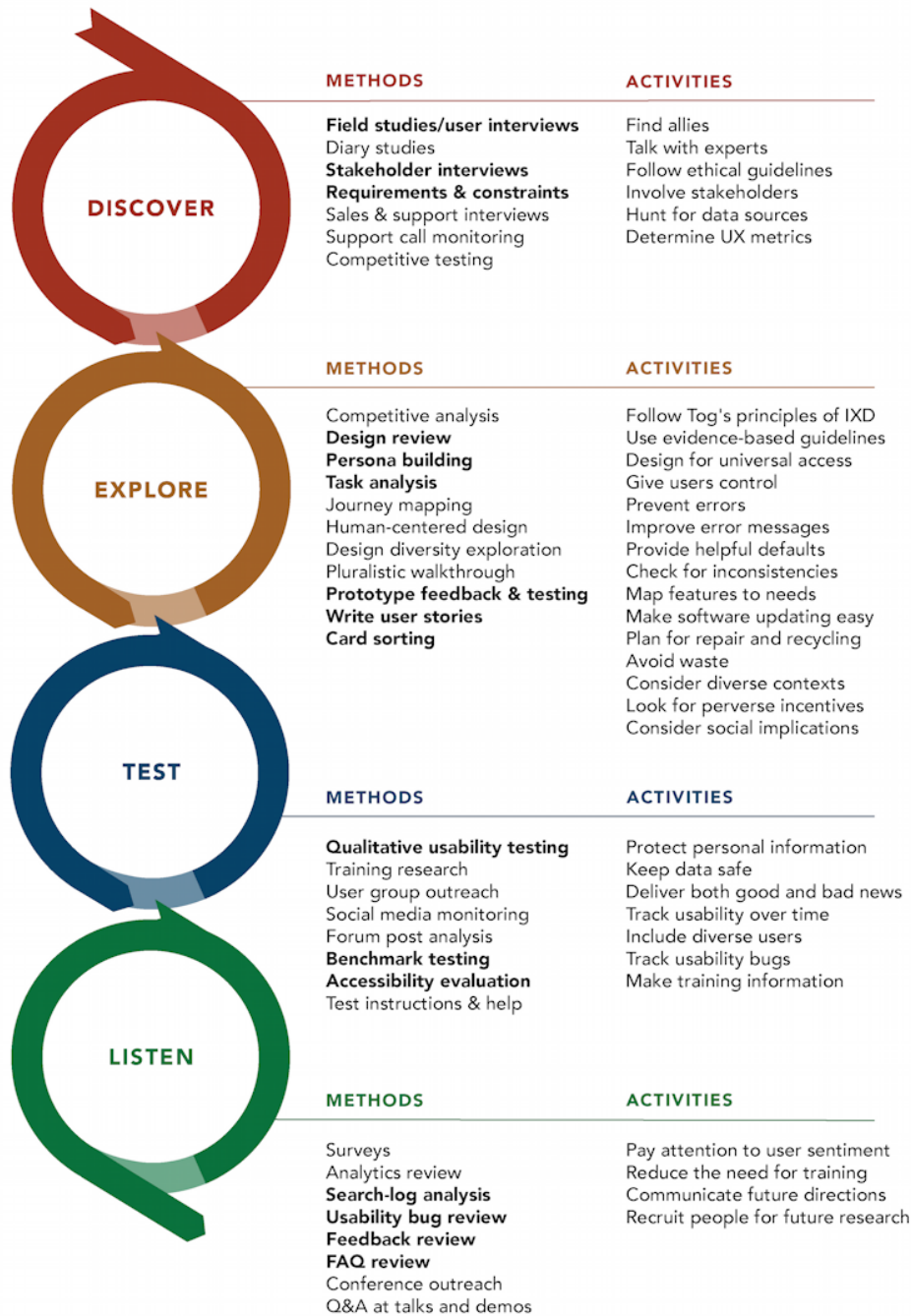


Figure 11.2: Central figure from Farrell's user-activity survey article for the Nielsen/Norman Group.

Bold methods are some of the most commonly used.

11.7 *Moran's Survey of 9 Quantitative User Activities*

Moran⁴ goes in depth with 9 different quantitative user activities:

- Quantitative Usability Testing (Benchmarking)
- Web Analytics (or App Analytics)
- A/B Testing or Multivariate Testing
- Card Sorting
- Tree Testing
- Surveys or Questionnaires
- Clustering Qualitative Comments
- Desirability Studies
- Eyetracking Testing

⁴ Kate Moran. Quantitative user-research methodologies: An overview. Technical report, Nielsen/Norman Group, 2018. URL <https://www.nngroup.com/articles/quantitative-user-research-methods/>

11.8 *Nielsen's 10 Usability Rules of Thumb*

A good learning activity is to apply Nielsen's 10 usability rules of thumb to your project:⁵

- Visibility of system status
- Match between system and the real world
- User control and freedom
- Consistency and standards
- Error prevention
- Recognition rather than recall
- Flexibility and efficiency of use
- Aesthetic and minimalist design
- Help users recognize, diagnose, and recover from errors
- Help and documentation

⁵ Jakob Nielsen. 10 usability heuristics for user interface design. Technical report, Nielsen/Norman Group, 1994. URL <https://www.nngroup.com/articles/ten-usability-heuristics/>

11.9 *Why Testing With 5 Users is Usually Enough*

Nielsen⁶ discusses why usability testing with 5 users is usually enough to discover the most prominent flaws in your current design and give you enough feedback to start refining the design.

⁶ Jakob Nielsen. Why you only need to test with 5 users. Technical report, Nielsen/Norman Group, 2000. URL <https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/>

11.10 *27 Tips for Conducting Successful User Research in the Field*

Farrell⁷ gives 27 tips for conducting user research 'in the field' — which means outside of the company's offices. Capstone design teams typically do not have their own dedicated office space, and typically have different resources available than a corporation. So not all of these tips are relevant to your context, but many of them are.

⁷ Susan Farrell. 27 tips and tricks for conducting successful user research in the field. Technical report, Nielsen/Norman Group, 2017. URL <https://www.nngroup.com/articles/tips-user-research-field/>

11.11 Levels of UX Design Maturity

Nielsen⁸ describes eight levels of team/corporate maturity with regards to UX maturity. Note that the levels are cumulative, so being at level 7 also means doing what is required for level 6 *etc.*

1. *Hostility Toward Usability.* You can do better than this.
2. *Developer-Centered User Experience.* The developer's care about usability, but they only rely on their own intuitions in making their designs. They do not due any user activities.
3. *Skunkworks User Experience.* No official budget or plan for user activities, but some manage to happen anyways.
4. *Dedicated UX Budget.* The team actively plans their milestones and goals around user activities, although typically focused just on user activities at the end of the design cycle.
5. *Managed Usability.* The organization dedicates a group to UX design.
6. *Systematic User-Centred Design (UCD) Process.* The team conducts early user research before they do any design. The team has an iterative design process that involves user activities in each iteration.
7. *Integrated User-Centered Design.* The project definition and the requirements are infused with user data.
8. *User-Driven Corporation.* The organization makes decisions about which projects to pursue based on user research. The concept of total user experience goes beyond the screen to also consider hardware selection and other aspects of user experience.

If your project involves user experience, and no other significant technical elements, then your team is expected to rise to at least UX maturity level 6: *systematic user-centred design process*. The team is expected to stretch towards maturity level 7 insofar as the project definition and requirements should at least incorporate user-oriented data from external sources, if not your own original user research.

For example, if the project was an app for reducing domestic food waste, that is a 100% user experience project: it likely has no other significant technical elements. Such a project might even stretch towards maturity level 8 by considering the hardware of the design, such as mounting an inexpensive tablet on the fridge or having a voice-interface device.

⁸ Jakob Nielsen. Corporate UX maturity: Stages 1–4. Technical report, Nielsen/Norman Group, 2006. URL <https://www.nngroup.com/articles/ux-maturity-stages-1-4/>; and Jakob Nielsen. Corporate UX maturity: Stages 5–8. Technical report, Nielsen/Norman Group, 2006. URL <https://www.nngroup.com/articles/ux-maturity-stages-5-8/>

This is analogous to the *Capability Maturity Model* described by the Software Engineering Institute at Carnegie Mellon University.

This stage doesn't have a lot of meaning in our capstone design team context.

The levels are cumulative, so truly achieving level 7 also means achieving level 6, *etc.*

11.12 Apply Universal Design for Accessibility

'Universal Design is the design and composition of an environment so that it can be accessed, understood and used to the greatest extent possible by all people regardless of their age, size, ability or disability.'⁹ While much of the literature around universal design focuses on the physical environment, it is also important in software. The seven principles of universal design are:¹⁰

1. Equitable Use
2. Flexibility in Use
3. Simple and Intuitive Use
4. Perceptible Information
5. Tolerance for Error
6. Low Physical Effort
7. Size and Space for Approach and Use

Learn about universal design and accessibility in software and apply your new knowledge to your project.

11.13 Peer Usability Review

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.

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.

⁹ <http://universaldesign.ie/What-is-Universal-Design/>

¹⁰ <http://universaldesign.ie/What-is-Universal-Design/The-7-Principles/>

A good book on universal design is:

Sara Hendren. *What Can a Body Do? How We Meet the Built World*. Riverhead Books, 2020. URL <https://www.penguinrandomhouse.com/books/561049/what-can-a-body-do-by-sara-hendren/>

<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

11.14 User Acquisition

Contributed by Yingning Gui SE2022

11.14.1 Define your user persona

Identify if the project is for enterprise or consumer users. If your product is best used in teams, then you are likely building for enterprise. Consider what type of teams would use it, what type of companies, what size of companies, and what industries. If your product is for consumers, consider what demographic you are targeting and where these people currently are online.

Next, flesh out the details of your user persona. What is your ideal user's current behaviour? What do they want to achieve? Define the demographic profiles, goals, and a scenario use case.

11.14.2 Find your users

Once you have created your user profile, you can now find places they are present. The internet has communities for every niche, try searching for communities relevant to your product on Reddit, Twitter, Hacker News and other forums. Post on these platforms and engage with potential users. There are also many modern platforms that product and technology enthusiasts browse to discover new solutions to their problems, such as Product Hunt, Beta List, and Indie Hackers.

Places you might find users:

- Reddit
- Twitter
- Facebook
- Hacker News
- Product Hunt
- Beta List
- Indie Hackers

11.14.3 Engage with your users

Be active on social media and keep an eye out for posts mentioning your product. It's also common for software products to set up Slack workspaces or Discord servers for eager users to provide feedback to the product.

Reflective Activities

12.1 Watch Past Project Presentations

Each member of your team watches a different past project video and writes a paragraph about what they learned from it.

12.2 Analyze Past Project Awards

Look at projects that have won awards and identify the qualities that the referees were celebrating. Express contrary opinions if you have them.

12.3 Index Past Projects

Pick a theme and build a list of past projects connected to that theme. Use the project abstracts and videos to learn about them. Contribute your list so that other students can benefit from it in the future.

12.4 Read Past Project Reports

Some past student research projects have produced written reports (and some of them have been published). Each member of your team reads a different report and writes a paragraph about what they learned from it.

<https://git.uwaterloo.ca/secapstone/abstracts/-/tree/master/publications>

<https://git.uwaterloo.ca/secapstone/abstracts/-/tree/master/reports>

12.5 Read Turing Award Speeches

Each member of your team reads a different Turing Award speech and writes a paragraph summarizing it, and another paragraph about how it might relate to your project.

<https://amturing.acm.org/>

12.6 *Read Video Game History: The Digital Antiquarian*

Jimmy Maher has been writing about the history of video games since 2011, covering primarily the period from the 1960s through to the end of the twentieth century. Each member of your team reads a different entry and writes a response (summary + project relevance).

<https://www.filfre.net/>
<https://www.filfre.net/hall-of-fame/>

12.7 *Watch ACM Tech Talks*

Each member of your team finds an ACM Tech Talk of some relevance to your project, watches it, and writes a paragraph summarizing the talk, and another paragraph discussing the content with respect to your project.

<https://learning.acm.org/techtalks>

12.8 *Read ACM Queue Articles*

ACM Queue is a magazine for practitioners (rather than researchers).

Each member of your team finds an ACM Queue article of some relevance to your project, reads it, and writes a paragraph summarizing the paper, and another paragraph discussing the content with respect to your project.

<https://queue.acm.org/>

12.9 *Read Classic SE Papers*

The Software Engineering Institute at Carnegie Mellon University have put together an annotated bibliography of classic research papers in software engineering.

Each member of your team finds a paper relevant for your project, and writes one paragraph summarizing the paper and one discussing its relevance for your project.

https://kilthub.cmu.edu/articles/journal_contribution/Seminal_Papers_in_Software_Engineering_The_Carnegie_Mellon_Canonical_Collection/6625733

12.10 *Watch a Documentary*

As a team, watch a documentary film related to your project. Each team member writes a paragraph about a different aspect of the film that is relevant to the project. Some possibilities might include:

- *Design Disruptors* A documentary about disruptive innovation, featuring interviews with a number of prominent designers.
- *The Social Dilemma* A docudrama about the impact of social networking software and the surveillance capital business model.
- *The Great Hack* A documentary on using targeted social media ads to sway elections.

<https://www.designdisruptors.com/>

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

There might also be great documentary films in the domain of your project that are worth watching and reflecting on.

12.11 Read a Book

There are many good books on software engineering. One classic, which is perhaps the most famous book in software engineering, is *The Mythical Man-Month* by Fred Brooks.¹ That was originally published in 1975, and so predates the invention of version control — the first research paper on version control was also published that year. A newer book, in which version control plays a prominent role, is *Software Engineering at Google*.² This book is a fairly comprehensive overview of what are considered best practices at one software-intensive company. There are also many other good books on software engineering.

¹ Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, 1975

² Titus Winters, Tom Manshreck, and Hyrum Wright, editors. *Software Engineering at Google*. O'Reilly Media, 2020

12.12 Assess Your Choice of Learning Activities

Part of being a professional engineer is knowing a wide range of techniques and ideas, and being able to select the appropriate ones. Reflect on the learning activities that your team has done to assess:

- which activities were worthwhile (and why);
- which activities were not (and why);
- which activities an outsider might think that you should have done, but which you rejected for good reason (near misses);
- and what better alternative activities might have been.

There is an old joke that is told in many cultures. Here is one retelling from Snopes.com:

“Nikola Tesla visited Henry Ford at his factory, which was having some kind of difficulty. Ford asked Tesla if he could help identify the problem area. Tesla walked up to a wall of boilerplate and made a small X in chalk on one of the plates. Ford was thrilled, and told him to send an invoice. The bill arrived, for \$10,000. Ford asked for a breakdown. Tesla sent another invoice, indicating a \$1 charge for marking the wall with an X, and \$9,999 for knowing where to put it.”

12.13 $n + 1$ 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:

- 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.

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

Communication Activities

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

– Aristotle, *Poetics*

Your communication should be:

- clear,
- correct,
- complete, and
- concise.

CREATE APPROPRIATE ABSTRACTIONS to satisfy these conflicting objectives — similar to programming. What distinguishes high-level languages from assembly are the abstraction mechanisms afforded to the programmer, which enable clearer and more concise exposition of the ideas. Effective technical communication is about finding and organizing the right natural language abstractions to express your project clearly, correctly, completely, and concisely.

MIND THE GAP between your words and deeds. This gap can consume all the oxygen in review/conversation, which creates frustration for everyone. Professional technical writing is different than first year resume inflation writing.

DON'T OVER-GENERALIZE. Make statements and that are precise and accurate. Clearly distinguish between what your prototype actually does versus what a fully developed system might do.

QUESTION EVERY ADJECTIVE. It's good if each adjective has an obviously observable empirical basis (*e.g.*, 'the ball is red'), or has a commonly accepted definition (*e.g.*, 'the race car is fast'). Avoid adjectives that express judgements, unless you have data or references to back those judgements. For example, do not say that you have done an 'extensive literature review': it's not your place to make that judgement. Each unsubstantiated adjective picks a fight with the reader.

13.1 Read Edward Tufte's Presentation Advice

Edward Tufte was a professor of computer science, statistics, and political science at Yale for twenty years, and at Princeton before that. He has a large volume of information available online and has also written five excellent books on STEM communication:

- *Visual Display of Quantitative Information*
- *Envisioning Information*
- *Visual Explanations*
- *Beautiful Evidence*
- *Seeing With Fresh Eyes*

Write a brief summary of what you learned and what you will apply to your presentation.

<https://www.edwardtufte.com/tufte/>

The format of this handbook is based on Tufte's books, using the Tufte-L^AT_EX style sheet.

13.2 Read Trees, Maps, and Theorems Presentation Advice

Jean-Luc Doumont completed his PhD in physics at Stanford, and now teaches STEM people how to organize and present their thoughts. He has written an excellent book on this topic called *Trees, Maps, and Theorems*, and also has some resources available online.

Write a brief summary of what you learned and what you will apply to your presentation.

<https://www.principiae.be/Xo300.php>

<https://www.principiae.be/book/Xo300.php>

New for 2020! How to give presentations remotely: <https://www.istem.illinois.edu/news/imrsec.remote.present.workshop.20.html>
<https://www.principiae.be/Xo800.php>

13.3 Watch Patrick Henry Winston's Presentation Advice

Patrick Henry Winston was a famous AI researcher at MIT, who gave a popular presentation to students every year on 'how to speak.' Write a brief summary of what you learned and what you will apply to your presentation.

<https://www.youtube.com/watch?v=Unzc731iCUY>

13.4 Learn from TED Presentation Advice

TED (Technology, Entertainment, Design) talks started in Silicon Valley in 1984, and are widely regarded as well-delivered talks from experts in their field ('ideas worth spreading'). There are several collections of advice on how to give a TED-style talk, some written and some video. Learn from them. Write a brief summary of what you learned and what you will apply to your presentation.

There are many sources of presentation advice from TED online. Here is one good article that appeared in *Harvard Business Review*: <https://hbr.org/2013/06/how-to-give-a-killer-presentation>

The sidebar of this article, *Find the perfect match of data and narrative* by Nancy Duarte, is particularly good.

13.5 *Learn from Nancy Duarte's Presentation Advice*

Nancy Duarte leads the largest communications firm in Silicon Valley. She has written or presented for TED, MIT, Stanford, Forbes, Harvard Business Review, and more. There are many samples of her teaching available online. Learn from them and write a brief summary of what you learned and what you will apply to your presentation.

<https://www.duarte.com/nancy-duarte/>
https://www.ted.com/talks/nancy_duarte_the_secret_structure_of_great_talks

13.6 *Conquer Your Fear of Public Speaking*

Many people in STEM careers have some fear of public speaking. Find, and try, some tips for becoming more comfortable with public speaking. Write a brief summary of what you learned and what you will apply to your presentation.

<https://hbr.org/2018/02/5-ways-to-get-over-your-fear-of-public-speaking>
There are many good TEDx talks on this topic.

13.7 Choose a Narrative Structure for Your Presentation

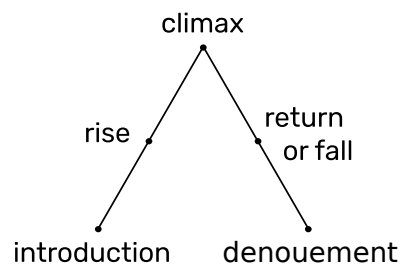
Narrative structure is an important abstraction mechanism that will help focus the audiences attention on the most interesting aspects of your complex project. Some options include:

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.

A variation on the historical structure is espoused by the creators of *South Park*:¹ 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. In literature this is often described by Freytag's pyramid:



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*.² 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.

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.

¹ <http://nathanbweller.com/creators-of-south-park-storytelling-advice-but-therefore-1>
Reference from Josh Kergen SE2018.

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

Figure 13.1: Freytag's Pyramid [image derived from Wikimedia]

² *Nature* 468:659–663, 2 Dec 2010

The old tv show *Columbo* used this structure: every episode begins by showing the crime taking place, so the audience knows whodunnit from the beginning. The drama is around how Detective Columbo is going to find evidence to conclude what the audience already knows.

13.8 Revise Your Writing

Writing good technical prose is a process.

1. *Brain Dump*: Write down everything in your mind. Start with what is easiest for you to write about (which might not be the introduction). Don't worry if it is messy or disorganized. Keep writing. New ideas will come to you as you write.
2. *Abstraction*: Identify the key concepts and ideas. Decide on consistent names for them. Rewrite the introduction and conclusion using this terminology.
3. *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.
4. *Target*: Who is your audience? 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?
5. *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.
6. *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, which is removed in the rewrite:

<p><i>Before:</i></p> <p>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.</p>	\mapsto	<p><i>After:</i></p> <p>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.</p>
--	-----------	--
 - Walk the reader through any inferences or reasoning that the text might require. For example:

<p><i>Before:</i></p> <p>The engineer rejects the discriminating example because a process P_1 can take an unintended lower-indexed mutex.</p>	\mapsto	<p><i>After:</i></p> <p>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.</p>
---	-----------	--
7. *Remix the Extras*: Add back any important points that had been relegated to the appendix previously.
8. *Spelling, Grammar & Consistency*: Polish.

Thanks to Matt Magni ECE 2018, and Carmen Celestini ECE290 TA for good suggestions on this process.

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

You do not have to write linearly, from the beginning to the end. It is often easier to write parts of the middle first.

Business people? Software people?
Other kinds of engineers?

Throw out the brain dump. Or lock it up somewhere so you cannot look at it during the rebuild step.

13.9 Revise Your Abstract

Your abstract is an important introduction to, and summary of, your project. It should be in the following form:

- about 300 words (one page of the booklet)
- mostly text (limited figures, tables, formulas, *etc.*; L^AT_EX permitted)
- primarily present tense
- limited jargon and acronyms
- good grammar and spelling

Your abstract should cover the following five issues:

Opportunity & Problem. What is the opportunity that your software will exploit? Who will use it? What is the technical problem it will solve? 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.”
- “In the US and Europe, 300,000 people die every year from cardiac arrest before they can reach medical care.”

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.

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.

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.”
- “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.”

Summary of (Expected) Results. What evidence have you gathered (will you gather) to demonstrate that the objective has been accomplished? See §14.2 for common approaches and targets.

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.

The *five W questions* can help you see the big picture: who, what, where, when, why.

ECE Group 2009.054

ECE Group 2012.035

Being open-ended is what makes this an *engineering design* project.

Do not mention course numbers here though. That’s on the Status Sheet §14.1.

ECE Group 2011.043

ECE Group 2012.031

This will evolve as your project progresses. Perhaps it will be written in future tense in SE490 and past tense for Symposium Day.

13.9.1 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.

This is a decent abstract: good, but not great. The revisions below clarify the context and expand the technical discussion.

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

Team Radiant did not have these results in January. This revised abstract was written in March. Your abstract for the Symposium Day booklet (due in January) might not yet have your final results. You could write (briefly) about your expected results or assessment methodology. For example, Team Radiant could have said, in January, that they will evaluate against the standard MIT/BIT dataset as well as a dataset provided by their industrial collaborator.

13.9.2 More Example Abstracts

There is an extensive collection of abstracts from past projects in the Handbook repository. This collection includes all past SE projects, and several years of ECE projects. It is quick and easy to read a sampling of them, and it will help you develop an appreciation for what makes a good project abstract.

<https://git.uwaterloo.ca/secapstone/abstracts>

13.9.3 *An entertaining example Abstract*

This is not a real project. But it shows you one example of a good structure for an abstract for a project that might be like yours.

Broccoli is a popular and nutritious vegetable that has been cultivated for over 2500 years. While broccoli originated in what is now Italy, it is eaten around the world, with 73% of the modern crop grown in India and China. There are billions of people around the world who enjoy broccoli on a regular basis, creating a huge market opportunity for our BB app.

Blanching is a fast, healthy, and delicious way to cook broccoli, either for immediate eating or as a preparatory step for freezing or sauteing. Broccoli that has been blanched before freezing retains up to 1300% more vitamin C than broccoli that is frozen directly. There are two distinct blanching techniques in common use, and both are supported by BB: boiling and steaming.

A major challenge in blanching is knowing how long to apply the heat. BB uses a feedback control system based on two forms of advanced image processing to ensure perfectly blanched broccoli every time. First, the initial time target is set by measuring the average floret size on the cutting board. Second, during the blanching process, the time target is continuously optimized based on the colour transformation of the vegetable in the pot.

Initial deployment of BB to the public was done during the fall harvest season, in collaboration with the Student Success Office and the St Jacob's Farmer's Market. Farmers reported that broccoli sales to younger adults increased by 57%. Interestingly, the app also drove broccoli sales to newly retired older people, a significant subset of whom are looking for new projects to engage with. App usage was not significant amongst adults of working age. The Student Success Office reported overall improved physical and mental health amongst students who improved their diet by using the BB app.

- Historical and cultural context.
- Market opportunity.
- These facts are real.

- User's motivation.
- Usage modalities.
- These facts are real.

- User's pain point.
- How application of advanced technical knowledge solves the problem.

- Initial public deployment during the last work term, between 4A and 4B.
- Results (fictitious).

13.10 *Read Authors You Want to Emulate*

When you read, start thinking about whether the author has a style that you admire or want to emulate. For example, many grad students (even native English speakers, even in England) read *The Economist* for this purpose. Write a brief summary of what you learned and what you will apply.

13.11 Write for Accessibility and ESL

If your project includes text for users, you might want to consider revising that text according to accessibility and ESL guidelines. There are accessibility writing guidelines at both the federal³ and provincial⁴ levels, as well as good online courses⁵ for you to learn from. UWaterloo also has web writing guidelines.⁶

SIMPLIFIED TECHNICAL ENGLISH is an international standard used in aviation to write instructions for pilots and aircraft mechanics, *etc.* It is the international language of aviation, and has also been adopted for procedures and maintenance manuals in other global industries. It has an active standards committee that revises the standard every three or four years. It is a good standard to consider when writing for an ESL audience (English as a Second Language).

METRICS TO ASSESS READING LEVEL. There are several standard metrics to assess reading level. Most of them are based on measures such as the number of syllables or characters in words, and the number of words in sentences. Some of these metrics are available in common word processors. Many of them are available on various web pages.

Some implementations of these metrics are very sensitive to where periods are placed. Do not assume that the implementation will infer a sentence ends at a carriage return. Also, some of the metrics involve a ratio of the number of big words per sentence. So you can sometimes bring the scored grade-level down by adding more small words to a sentence — even though that might actually decrease the readability of the sentence. Like all metrics, take them as a guideline.

ASSESSING READING LEVEL BY GRADE LEVEL OF INDIVIDUAL WORDS. The metrics just look at the sizes of words, but not their actual meaning or what grade level a student would be expected to know the word at. It's possible to build up some grade-level word lists from various sources, then write a script that assigns a grade level to each word in a text. This kind of analysis can help you identify which specific words might be challenging for readers, as well as helping you identify when you are using unnecessary synonyms and could reduce the overall vocabulary of the document by standardizing terminology.

³ <https://www.canada.ca/en/treasury-board-secretariat/services/government-communications/canada-content-style-guide.html>

⁴ <https://www.aoda.ca/accessible-writing-style/>

⁵ <http://www.humber.ca/makingaccessiblemedia/modules/05/02.html>

⁶ <https://uwaterloo.ca/web-resources/resources/usability/writing-web-tips>
<http://www.asd-ste100.org/>
https://en.wikipedia.org/wiki/Simplified_Technical_English

This website implements half a dozen readability metrics: <https://www.webfx.com/tools/read-able/check.php>

An example script with these word lists is in the source repository for this handbook: <https://git.uwaterloo.ca/secapstone/handbook/-/tree/master/activities/communication/accessibility>
https://en.wiktionary.org/wiki/Appendix:1000_basic_English_words
https://www.berkeleyschools.net/wp-content/uploads/2013/05/BUSD_Academic_Vocabulary.pdf
<https://www.flocabulary.com/1st-grade-vocabulary-word-list/>

Project Evaluation

Capstone projects are intended to show application of almost all of the learning objectives in an undergraduate degree. Consequently, the evaluation is broad:

see §1 *Learning Objectives*

<i>Facet</i>	<i>Begin</i> SE390	<i>Middle</i> SE490	<i>End</i> SE491
Reflection (learning outcomes, feedback, impact, IP, etc.)	15%	15%	15%
Requirements & Specifications	25%	20%	15%
Design, Implementation, & Deployment	15%	25%	25%
Verification & Validation (testing & results)	10%	10%	25%
Teamwork	15%	10%	10%
Communication (abstract, demo, presentation)	10%	10%	10%

FORMATIVE AND SUMMATIVE ASSESSMENTS: Final exams are the quintessential example of summative assessments. A summative assessment measures what has been learned or achieved.

The purpose of a formative assessment is to give feedback. Formative assessments can be more varied and exploratory in nature. Formative assessments might be graded on effort — the grade might not be intended as a measurement of the learning.

ASSESSMENT PROGRESSION: In the beginning of the capstone process there is an emphasis on formative assessments, as exploration and feedback and learning are the priorities at that time. Towards the end the focus shifts to summative assessment — both of the software and of the engineering efforts.

HOLISTIC EVALUATION: Capstone design projects differ significantly in their domains, technical challenges, trade-offs, and objectives. Holistic evaluation looks at a project as a whole, considering all factors and trade-offs. In contrast, the analytical or reductive grading approach examines facets or components in isolation.

The terms *formative* and *summative* assessment are also commonly used in HCI. For example, a formative usability assessment might involve a user thinking-aloud as they use the software, in order to give the designer insight into the user experience.

A summative assessment, by contrast, might measure the time taken to perform a task with a particular user interface. This summative assessment measures whether the user interface meets its objectives, but doesn't give insights in to how to improve it.

GRADING CRITERIA. Generally speaking, capstone projects should, in all relevant ways:

- exemplify the learning objectives;
- demonstrate the skills expected of a graduate;
- make appropriate trade-offs and judgments; and
- not suffer serious oversights.

see §1 *Learning Objectives*

A central premise is that the software should perform its intended function properly and in a unified way.

QUANTIZATION BY LETTER GRADES. In an open-ended design project, it is more reasonable to grade on a quantized scale.

A+	95	exceeds expectations on many learning objectives, with no oversights
A	90	demonstrates mastery on all learning objectives, with no oversights
A-	85	demonstrates mastery on most learning objectives, with some minor oversights
B+	80	adequacy on all learning objectives, mastery of some, with some minor oversights
B	75	adequacy on all learning objectives, with some minor oversights
B-	70	adequacy on all learning objectives, with some serious oversights
C	65	adequacy on most learning objectives, with some serious oversights
D	55	significant shortcomings in learning objectives, major oversights
F	45	unethical, unprofessional, incomplete, <i>etc.</i>

DEDUCTIONS. Deductions can be assessed for three reasons: oversights (*i.e.*, poor professional judgment); avoidable deficiencies (a list will be provided in advance), and missing deadlines.

Three main strategies for avoiding oversights are *peer interactions*, *learning activities*, and *formative assessments*.

The table above tries to characterize the role of oversights in grading. Some projects might not match these patterns, and in those cases oversights might be factored out and handled separately.

14.1 Project Status Sheets

The project status sheet (sample on following pages) may be used to characterize the current status of the project. These are *not* grading sheets. Their purpose is to give a broad overview of the project. Depth is provided in the presentation. The breadth of the status sheet is intended to help prevent oversights—to ensure that all of the bases are covered. For any given project, some parts of the status sheet will be more interesting than others.

Fillable PDF at https://ece.uwaterloo.ca/~se_capstone/status-sheets/
 L^AT_EX source for status sheets in handbook repo at <https://git.uwaterloo.ca/secapstone/handbook>

LINES OF CODE are reported on the status sheets, but are not the primary basis of assessment. Many factors may moderate the interpretation of this metric, including difficulty of the problem, nature of the problem, choice of language, *etc.*

To paraphrase Winston Churchill, lines of code is the worst metric we have except for all the others.

Lines of code is intended to include exploratory prototypes, tests, scripts, *etc.* The metric is intended to include just about everything written by hand and processed by machine.

Lines of code should be measured with CLOC: <https://www.tecmint.com/cloc-count-lines-of-code-in-linux/>
`sudo apt install cloc`
 LOC should not be measured by Git.

SE+CS Capstone Project Status Sheet

Contents

<i>1</i>	<i>Project & Team Identification</i>	<i>2</i>
<i>1.1</i>	<i>Opportunity Assessment</i>	<i>3</i>
<i>1.2</i>	<i>Curriculum Connections</i>	<i>4</i>
<i>2</i>	<i>Applying the Wisdom of the Past</i>	<i>5</i>
<i>2.1</i>	<i>Software Engineering Rules of Thumb</i>	<i>5</i>
<i>2.2</i>	<i>Learning from Prior Capstone Projects</i>	<i>5</i>
<i>3</i>	<i>Feedback Given and Received</i>	<i>6</i>
<i>3.1</i>	<i>Responses to Feedback Received</i>	<i>6</i>
<i>3.2</i>	<i>Summary of Feedback Given</i>	<i>6</i>
<i>4</i>	<i>Processes, Practices, and Tools</i>	<i>7</i>
<i>4.1</i>	<i>Tools</i>	<i>7</i>
<i>4.2</i>	<i>Process</i>	<i>7</i>
<i>4.3</i>	<i>Process Maturity Assessment</i>	<i>8</i>
<i>4.4</i>	<i>Additional Practices</i>	<i>8</i>
<i>5</i>	<i>Requirements & Specification</i>	<i>9</i>
<i>5.1</i>	<i>Exploratory Activity Summary</i>	<i>9</i>
<i>5.2</i>	<i>SRS Summary</i>	<i>9</i>
<i>5.3</i>	<i>Deviations from SRS</i>	<i>9</i>
<i>6</i>	<i>Design, Implementation, and Deployment</i>	<i>10</i>
<i>6.1</i>	<i>Applied Foundations</i>	<i>10</i>
<i>6.2</i>	<i>High-Level Design</i>	<i>10</i>
<i>6.3</i>	<i>Components/Libraries</i>	<i>10</i>
<i>6.4</i>	<i>Implementation</i>	<i>11</i>
<i>6.5</i>	<i>Deployment</i>	<i>11</i>
<i>7</i>	<i>Verification & Validation</i>	<i>12</i>
<i>7.1</i>	<i>Testing</i>	<i>12</i>
<i>7.2</i>	<i>Summative User Activities</i>	<i>12</i>
<i>7.3</i>	<i>Results Target(s)</i>	<i>12</i>
<i>8</i>	<i>Graduate Attributes Self Assessment</i>	<i>13</i>

This form attempts to give a broad project overview. It is about breadth, not depth. For any given project, there will be parts of this form that are more and less important.

Your presentation is where you will go in depth on the things that are of particular interest for your project. This form is to identify interesting issues, but does not have space to explain them thoroughly.

1 *Project & Team Identification*

TeamName : Project

Abstract: See Handbook for instructions.

Students

1.1 Opportunity Assessment

WHAT DO YOU WANT TO LEARN?

What are you hoping to learn?

What you want to learn might include Technical Elective courses that complement your project. You might be learning the content of those courses through capstone rather than by taking the course.

WHAT IS THE OPPORTUNITY IN THE WORLD?

Why does the world need your project? Why is now the right time to do it?

Why does the world need your project? Why is now the right time to do it? Has something changed in the world to create this opportunity now? Is this a longstanding opportunity?

<u>Advantages</u>	<u>Challenges</u>
<u>External Partner</u>	<u>Access to Data</u>
<u>New Technology</u>	<u>Against Vested Interests</u>
<u>New Data Source</u>	<u>Dense Market Space</u>
<u>Clear Opportunity/Prob.</u>	<u>Significant Marketing</u>
<u>Clear User Population</u>	<u>Scope Too Large</u>
<u>Domain Knowledge</u>	<u>Scope Too Small</u>
<u>Novel Idea</u>	<u>Scope Too Foggy</u>
<u>Grounded Idea</u>	<u>Free Engineering Labour</u>
<u>Awesomeness</u>	<u>Legal/Regulatory Hurdles</u>
<u>Other</u>	<u>Other</u>

These checklists have some common advantages and challenges that have been faced by projects in the past.

Free or undervalued engineering labour is a strategic blunder if it is the basis of a proposed commercial competitive advantage: *i.e.*, we can undercut the competition because they pay their engineers and we do not value our own work. This violates P.E.O. Code of Ethics 77(7)v. Note that *pro bono* work in the public interest, including patches to established FOSS projects, does not suffer this blunder.

SUMMARY OF ADVANTAGES

Summary of advantages.

STRATEGIES FOR MITIGATING CHALLENGES

Strategies for mitigating identified challenges.

1.2 Curriculum Connections

Course	Who?	Why?
SE463	everyone	project needs a spec
SE464	everyone	project needs a design
SE465	everyone	project needs testing

Identify courses with some connection to your project. It might be the case that your project goes beyond what is actually taught in the course — list the course here anyways.

It's ok if nobody on your team is taking the course. Capstone gives you a change to learn ideas from other courses on your own. Still list the course here, as there is a connection.

CRUD projects need to list CS349 user interfaces and CS449 / MSCI343 / SYDE548 user-centred design. CRUD projects often have no algorithmic technical depth: their technical depth comes from human-computer interaction / user-centred design.

CRUD is an old industry acronym for Create/Read/Update/Delete. In the modern context, a database-backed website or app. CRUD is a very popular technical category.

2 *Applying the Wisdom of the Past*

2.1 *Software Engineering Rules of Thumb*

Lehman Type:

Discuss Lehman Type of your project

Perlis Epigram:

Discuss application of Perlis epigram to your project

Another Law:

Discuss application of another law / rule of thumb to your project

2.2 *Learning from Prior Capstone Projects*

Project 1:

Discuss what you learned from a past capstone project 1

Project 2:

Discuss what you learned from a past capstone project 2

Project 3:

Discuss what you learned from a past capstone project 3

3 *Feedback Given and Received*

3.1 *Responses to Feedback Received*

<i>Party</i>	<i>Date</i>	<i>Focus</i>	<i>Outcome</i>

3.2 *Summary of Feedback Given*

<i>Team</i>	<i>Date</i>	<i>Comment</i>

4 Processes, Practices, and Tools

4.1 Tools

Tools you used for version control, testing, prototyping, etc.

4.2 Process

Briefly describe your team's process

Briefly describe your team's process.

Process Alternatives

Describe alternative processes from your co-op work terms and why your team isn't using them.

Briefly describe some alternative processes you have used on co-op work terms and why your team isn't using them.

4.3 *Process Maturity Assessment*

Joel Test:

CMMI Capability:

CMMI Maturity:

UX Maturity:

Other:

Discussion of Process Maturity

Describe your team's process.

There are several process assessments in Software Engineering, including those listed here. Perhaps you know of another interesting one.

The process assessments listed here often apply to entire organizations, and not just individual projects, but nevertheless many aspects are applicable to individual projects.

The Joel Test was defined by Joel Spolsky as a lightweight assessment.

CMMI Capability and Maturity levels are defined by the Software Engineering Institute at Carnegie Melon University (CMU). For individual project teams it only makes sense to consider CMMI levels 0-2.

UX Maturity levels are defined by the Nielsen/Norman Group.

4.4 *Additional Practices*

Brief discussion of your team's professional practices of interest, such as code review, pair programming, *etc.*

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

14.2 Results (An Aspect of Validation)

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.

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.

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.

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.

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.

14.2.1 *New Product*

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.

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

TEAM RED COCONUT (2014) created the UWFlow.com website for 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 demonstrate the flexibility of their framework (or at least they did not say so in their talk).

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 14.1 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).

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>

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

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.

Grade: A+.

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 interestingly different games to demonstrate the flexibility and value of the framework.



Figure 14.1: 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.

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

14.2.2 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.

<i>Grade</i>	<i>Criteria</i>
A+	Novel results or context <i>or</i> paper published, <i>etc.</i>
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.

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.

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

14.2.3 Consulting

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.

<i>Grade</i>	<i>Criteria</i>
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.

14.2.4 Free/Open-Source Software Patch

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

<i>Grade</i>	<i>Criteria</i>
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 head-quarters. 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.

14.2.5 Application of Advanced Technology

Some teams choose to focus on solving a particular problem by applying advanced technical knowledge in a specific domain. Rather than focus on having an organization or individual users adopt a product, these projects typically involve finding a problem and solving it in a way that is better than existing solutions. The main criteria here is that the project demonstrates excellence in several measures that have to do with the importance of the problem, the (technical) difficulty of the problem/solution, novelty, *etc.*. This may require gaining significant in-depth knowledge either by combining concepts from several ATEs or by doing extensive self-study.

Grade Criteria

<i>Grade</i>	<i>Criteria</i>
A+	Demonstrates that a problem was solved convincingly, meets two or more of the measures below with excellence, <i>and</i> positive mention from an industry professional. <i>Measures</i> · Importance of the problem · (Technical) difficulty and scope · Novelty of the solution · Applies knowledge from multiple upper-year courses
A	Demonstrates that a problem was solved convincingly and meets two or more of the measures above with excellence.
B	Demonstrates that a problem was solved convincingly and meets one of the measures above with excellence.
C	Demonstrates that a problem was solved satisfactorily. Scope and depth similar to a 499 project.
F	Demonstrates that a problem was solved marginally. Scope and depth similar to a course project.

TEAM DYNALIST (2017) created dynalist.io, an outlining app. They used a novel algorithm for managing collaborative, hierarchical lists. They received positive mention from industry professionals in the form of testimonials from its 10k users.

Grade: A+. This project received an A+ in New Product rubric [§14.2.1], but they would have done well under this rubric as well.

TEAM RED COCONUT (2014) created UWFlow.com website for students to critique courses and share their schedules. Course critique websites have been done multiple times over the years and is not a novel solution. At its core UWFlow is a C.R.U.D project that would not meet the difficulty measure with excellence.

Grade: B. This project received an A+ in New Product rubric [§14.2.1] with their large user base but would *not* have done as well using this rubric.

APPLICATIONS OF ADVANCED TECHNOLOGY might benefit from working with the appropriate stakeholders to ensure that the problem being solved is worthwhile and that the solution is appropriate for the audience being targeted. Finding someone in industry or academia who is an expert in the domain being focused on can help a project prove the importance, novelty, or difficulty of the work involved.

When choosing a project, keep in mind the difficulty and scope of the project. It would be inappropriate to choose a project whose content is only at a second year level. It would also be inappropriate to choose a project that could be completed in one term by a single fourth year student. It is expected that groups will apply their engineering judgement and try to come up with a reasonable scope and timeline for what can be completed by symposium day.

A possible way to demonstrate that the problem was solved convincingly is by comparing the solution to the alternatives. Aim to show a concrete example of the major improvement enabled by the product. Keep in mind that a “major improvement” does not necessarily imply that the problem was solved in its entirety. A project can still make a big impact even if the problem domain is such that there is no complete solution.

For example, a degree auditor that can automate 80% of cases is still a major improvement over no automation.

14.3 *Teamwork Assessment*

TBD — will include peer evaluation

14.4 *Communication Assessment*

Generally speaking, all communication should be:

- clear,
- complete,
- concise, and
- correct.

There is some inherent tension between these criteria — especially between being *complete* and *concise*. Learning to navigate these waters is an important communication learning objective.

§13 *Communication Activities* contains specific advice for your abstract, and also refers you to a variety of sources of advice for your presentation.

In your presentation, it is advisable to briefly indicate which presentation advice you are following.

14.5 *How Referee Grades Get Combined*

On Symposium Day, referees submit grading sheets to the instructor, who is then responsible for merging those grades — or making exceptions to them. First we’ll discuss merging grades.

14.5.1 Merging Referee Reports

The simplest case is when all of the referees agree on what the grades should be. For example, for team eCryptFS in SE2014, the referees got together after the presentation and then submitted a single grading sheet with their consensus assessment. This level of unanimity is rare.

Usually, the referees have different opinions. There are a number of strategies that the instructor might use to merge the grades. The instructor might use different merge strategies for different parts of the grade for the same group. For example, team HAKS in SE2018 had two external referees: one self-identified as business/non-technical, and the other self-identified as technical. The business referee's technical assessment was given less weight, whereas his communication assessment was given more weight.

- *Average.* Take the mean of the referee assessments.
- *Weighted Average.* Take the mean of the referee assessments, but placing more importance on some referees and less on others.
- *Mode.* Take the most common grade. For example, if two referees give A, and one gives B, just go with A.
- *Discard Outliers.* The instructor might choose to disregard grades from a particular referee if they are significantly out of line with the other referees.
- *Max/Min.* Take one of the outliers.

14.5.2 Making Exceptions to Referee Reports

In rare cases, the instructor (or the students) might completely disagree with the referee's Symposium Day assessments. There are several ways in which grades can diverge from referee reports:

- *Instructor Assessment.* Final grades are at the sole discretion of the instructor. The instructor can choose to disregard the referee reports. This is a rare occurrence, but does happen sometimes. As a rule of thumb, the instructor tends to restrict exercising discretion in this way to assigning grades that are higher than what the referees assessed. The instructor, as a matter of practice, is unlikely to disregard all referee reports and assign a lower grade.
- *The Risk Reward Bonus* §14.7 is one mechanism for adjusting the final grade outside of the ordinary Symposium Day process. In this case, the referee assessments from Symposium Day stand, but bonus points are awarded for factors that are not captured in the Symposium Day marking rubrics.

This can happen for lots of reasons. For example, there are some referees that are known to grade to extremes: A or C, but never give out B's. If a referee like that is also known to give out A's rarely, then a C from them might be weighted less — at the instructor's discretion.

For example, a team in SE2016 asked a grad student, who happened to be their roommate, to be a referee. He gave them A+ across the board, which was not the judgement of the other referees.

On the other side, Team Amalgam from SE2014 had an external referee from ETH Zurich in Switzerland. He graded according to European norms, which are lower than North American norms. Remember that the important part of the Symposium Day experience is the formative feedback from the referees, not their summative assessments.

14.6 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. We are open-minded about high-quality nominees. The following categories of candidates are encouraged:

- Faculty or staff (from any university).
- SE alumnus.
- Graduate students in a related area of research. For example, Amalgam SE2014 had a researcher from ETH Zürich (Switzerland).
- The ‘Customer’ for the project. This is especially relevant for the Custom Software rubric.
- The creator or maintainer of some technology used in the project. For example, Team Sleekbyte SE2016 had a senior engineer from the Swift language team at Apple. Team Parallax SE2016 had the creator of WebGL.

The Instructor retains final discretion on grades. This oversight is 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.

14.6.1 Referee Scheduling

There are two kinds of referees from a scheduling perspective:

1. *Referees who are unique to your team.*
2. *Referees who might also be requested by other teams.*

14.6.2 Inviting Team-Specific Referees

You can directly invite referees that are unique to your team: referees that will not be invited by other teams, and that your team has a deep connection with. Your steps to invitation are:

1. Select two videos of past presentations for your referee to view in advance of Symposium Day. Your selections should be using the same Results Rubric as your team. Your invitation to your referee will include your assessment of those past projects.
2. Send a draft of your invitation email to the course instructor for suggestions and approval of the referee. Also include a brief description of the referee’s suitability, their involvement in your project, and disclose if there is any relationship with the referee outside of your project.
3. Send your invitation email to your referee; copy the instructor.

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.

Any personal connections to nominated referees should be declared to the Instructor. A personal connection does not necessarily disqualify the referee — that judgement is for the instructor to make.

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.

You can find links to videos of past presentations in [\\$?? Awards](#).

See below for an example template.

For example, if the referee is your old boss or uncle *etc.* Previous relationships do not disqualify referees, but are important to disclose.

Here is an example template for your referee invitation email. You should customise this template to fit the particulars of your situation.

Dear X,

Thank you for your mentorship/involvement/*etc.* in our Software Engineering Capstone Design Project over the last year. Your efforts have enriched our educational experience.

We would like to invite you to judge our project on our Symposium Day, Tuesday, March 28, 2017. Symposium Day is a public event where all teams in our cohort will present their projects. Judging our project will involve three steps that will take about an hour in total:

1. *Preparation:* Watch a 20 minute video of a presentation from a previous team in a previous year, to become familiar with the grading sheet and the expected calibre of the projects.
2. *Demonstration:* We will give you a 10 minute interactive demonstration of our software at our booth on the Symposium floor, supported by our poster.
3. *Presentation:* We will give a 20 minute presentation to you and the general public, followed by 10 minutes of audience questions.

After the presentation you will submit your grading sheet to the course instructor, Prof P <p@uwaterloo.ca>.

If you accept our invitation to judge our project, please reply in the affirmative and let us know what times you are available on Symposium Day (March 28, 2017). Presentations typically run from 8:30AM to 3PM. The course instructor will make every effort to schedule our presentation at a time that is convenient for you.

Thank you,
Team T

Next communication from course instructor ...
referee package
links to videos
sample grading sheet

14.7 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.

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

Pick one word here.

Say something nice.

If the referee is remote, then do this online the day before.

Specify how, as appropriate: on paper if local, via email if remote.

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.

1. Clay Mathematics Institute Millenium Prize Problem.
2. Other researchers/companies have tried and failed.
3. Requires inventing a new idea in a new area.
4. Applying an old idea in a new area.
5. 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.

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.

15.1 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>

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.

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.

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.

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.

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.

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.

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.

15.2 *Collaborating on Free / Open Source Software*

Collaborating on free or open source software is usually uncontroversial: 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.

15.3 *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.

15.4 Collaborating with a Startup

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.

Velocity has good resources online:
<http://wiki.velocity.uwaterloo.ca/Legal>

15.4.1 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.

15.4.2 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.

1. I agree to treat information from *collaborator* as confidential, and to not disclose it beyond what is necessary for my academic requirements.
2. 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-disclosure, to pacify collaborator

Non-compete, to pacify collaborator

3. 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.

15.4.3 *Co-Founding a Startup with a Collaborator*

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.

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.

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 graduation, 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.

15.5 *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.

15.6 *Intellectual Property & Standards Report*

The following pages contain a template report covering various aspects of intellectual property law: copyright (including software licensing), patent, trademark, and privacy. There is also a section on industry standards.

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.

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.

Intellectual Property and Standards Report

Team Name

Student1, Student2, Student3

Date

Project abstract (uncomment \LaTeX input{../abstract})

Contents

1	Intellectual Property	2
1.1	Copyright	2
1.2	Patent	3
1.3	Trademark	3
1.4	Trade Secrets	3
1.5	Export Controls	3
1.6	End User License Agreement or Terms of Service	3
2	Privacy	4
3	Industry Standards, Regulations, Norms	4

Instructions: Complete the sections of the report relevant to your project.

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.

\LaTeX Instructions:

- Edit main.tex
- Do not rename main.tex
- Set your text editor to use plain ASCII — not UNICODE. The most common problem here is ‘smart quotes’.
- Ampersand is a special character in \LaTeX that must be escaped with a preceding backslash.
- Use the `URL` macro to wrap URLs
- main.tex will not build on its own — it reads in your meta.tex file. So if you copy just main.tex into an online \LaTeX system, it won’t build properly. You need to build it on your local machine (if you want to build it at all).

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 73² 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
- Does your project involve analysis of third-party datasets? Who owns the copyright of them? What is the licence?
TODO
- Who will retain ownership of the copyrights on your software after you graduate? You? Your customer? Someone else?
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.softwrefreedom.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.*

1.2 Patent

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

TODO

- 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.

TODO

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

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

There are some online tools for generating these, such as TermsFeed.com

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

- USA: *Health Insurance Portability and Accountability Act (HIPAA)*

TODO

- Europe: *Data Protection Directive*

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.

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 below. Please delete those that are not relevant to your project, and expand those that are — plus, of course, add other relevant items not on this list.

- 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 Consortium)
- CAP Theorem
- Language standards: SQL, C, Java, *etc.*
- Google Material Design UI/UX Standard
- Apple Design UI/UX Standard

Bibliography

- [1] M. Balsom, R. Barrass, J. Michela, and A. Zdaniuk. Processes and attributes of highly effective teams. Technical Report The WORC Group, University of Waterloo, 2009.
- [2] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley, 2 edition, 2003.
- [3] Sharon Anthony Bower and Gordon H. Bower. *Asserting Yourself*. Da Capo Lifelong Books, 2004.
- [4] Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, 1975.
- [5] Avi Bryant. Web Heresies: The Seaside Framework. OSCON, 2006. URL http://conferences.oreillynet.com/cs/os2006/view/e_sess/8942.
- [6] 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.
- [7] 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.
- [8] 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.
- [9] Edward W. Constant. *The Origins of the Turbojet Revolution*. The Johns Hopkins University Press, 1980.
- [10] Melvin E. Conway. How do committees invent? *Datamation*, 14(5):28–31, April 1968. URL <http://www.melconway.com/research/committees.html>.

- [11] 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.
- [12] 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.
- [13] Alan M. Davis. Operational prototyping: A new development approach. *IEEE Software*, 9(5), September 1992.
- [14] Edward de Bono. *Six Thinking Hats: An Essential Approach to Business Management*. Little, Brown, & Company, 1985.
- [15] Clive L. Dym and Patrick Little. *Engineering Design: A Project Based Introduction*. John Wiley & Sons, 3 edition, 2008.
- [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] Susan Farrell. 27 tips and tricks for conducting successful user research in the field. Technical report, Nielsen/Norman Group, 2017. URL <https://www.nngroup.com/articles/tips-user-research-field/>.
- [18] Susan Farrell. UX research cheat sheet. Technical report, Nielsen/Norman Group, 2017. URL <https://www.nngroup.com/articles/ux-research-cheat-sheet/>.
- [19] Eugene S. Ferguson. *Engineering and the Mind's Eye*. The MIT Press, Cambridge, Mass., 1992.
- [20] Anthony Finkelstein. *SIGSOFT Software Engineering Notes*, 1992. URL <http://wwwo.cs.ucl.ac.uk/staff/A.Finkelstein/papers/immaturity.pdf>.
- [21] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.
- [22] 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’.
- [23] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
 - [24] David Garlan and Dewayne E. Perry. Introduction to the special issue on software architecture. *IEEE Transactions on Software Engineering*, 2(1), April 1995.
 - [25] 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.
 - [26] David Garlan and Mary Shaw. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, Inc., 1996.
 - [27] H. Goldstine. *The computer from Pascal to Von Neumann*. Princeton University Press, 1972.
 - [28] William J.J. Gordon. *Synectics: The Development of Creative Capacity*. Harper & Row, New York, 1961.
 - [29] Sara Hendren. *What Can a Body Do? How We Meet the Built World*. Riverhead Books, 2020. URL <https://www.penguinrandomhouse.com/books/561049/what-can-a-body-do-by-sara-hendren/>.
 - [30] 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.
 - [31] 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.
 - [32] 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/>.
 - [33] 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.

- [34] Alita Joyce. Formative vs. summative evaluations. Technical report, Nielsen/Norman Group, 2019. URL <https://www.nngroup.com/articles/formative-vs-summative-evaluations/>.
- [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.
- [36] H.-Y. Benjamin Koo. *A Meta-language for Systems Architecting*. PhD thesis, Engineering Systems Design, Massachusetts Institute of Technology, 2005.
- [37] Butler W. Lampson. Hints and principles for computer system design. URL <https://arxiv.org/abs/2011.02455>. Updated version of the 1983 classic.
- [38] 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.
- [39] P. Lencioni. *Five Dysfunctions of a Team*. John Wiley and Sons Inc., New York, NY, 2002.
- [40] Bill Moggridge. *Designing Interactions*. The MIT Press, Cambridge, Mass., 2007.
- [41] Christine Moore. How to set and achieve life goals the right way. Technical report, Positive Psychology, 2020. URL <https://positivepsychology.com/life-worth-living-setting-life-goals/>.
- [42] Kate Moran. Quantitative user-research methodologies: An overview. Technical report, Nielsen/Norman Group, 2018. URL <https://www.nngroup.com/articles/quantitative-user-research-methods/>.
- [43] Gail Murphy, David Notkin, and Kevin J. Sullivan. Software reflexion models. *IEEE Transactions on Software Engineering*, 27(4):364–380, 2001.
- [44] Jakob Nielsen. 10 usability heuristics for user interface design. Technical report, Nielsen/Norman Group, 1994. URL <https://www.nngroup.com/articles/ten-usability-heuristics/>.

- [45] Jakob Nielsen. Why you only need to test with 5 users. Technical report, Nielsen/Norman Group, 2000. URL <https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/>.
- [46] Jakob Nielsen. Corporate ux maturity: Stages 1–4. Technical report, Nielsen/Norman Group, 2006. URL <https://www.nngroup.com/articles/ux-maturity-stages-1-4/>.
- [47] Jakob Nielsen. Corporate ux maturity: Stages 5–8. Technical report, Nielsen/Norman Group, 2006. URL <https://www.nngroup.com/articles/ux-maturity-stages-5-8/>.
- [48] Alex F. Osborn. *Applied Imagination: Principles and Procedures of Creative Problem Solving*. Scribner & Sons, New York, 1953.
- [49] David Lorge Parnas. On the Criteria to be Used in Decomposing Systems into Modules. *Communications of the ACM*, 15(12):1053–1058, December 1972.
- [50] 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/>.
- [51] 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>.
- [52] Christian Rohrer. When to use which user-experience research methods. Technical report, Nielsen/Norman Group, 2014. URL <https://www.nngroup.com/articles/which-ux-research-methods/>.
- [53] G. Rolfe, D. Freshwater, and M. Jasper. *Critical reflection in nursing and the helping professions: a user's guide*. Palgrave Macmillan, Basingstoke, 2001.
- [54] 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.
- [55] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects*. Addison-Wesley, 2000.
- [56] Tom Schorsch. The Capability Im-Maturity Model (CIMM). *CrossTalk Magazine*, 1995.

- [57] Willard Simmons. *A Framework for Decision Support in Systems Architecting*. PhD thesis, Aeronautics & Astronautics, Massachusetts Institute of Technology, 2008.
- [58] Richard N. Taylor, Nenad Medvidović, and Eric M. Dashofy. *Software Architecture: Foundations, Theory and Practice*. John Wiley & Sons, 2009.
- [59] K. W. Thomas and R. H. Kilmann. *Thomas–Kilmann Conflict Mode Instrument*. Xicom, Tuxedo NY, 1974.
- [60] Walter G. Vincenti. *What Engineers Know and How They Know It: Analytical Studies from Aeronautical History*. The Johns Hopkins University Press, 1993.
- [61] Eugene K. Von Fange. *Professional Creativity*. Prentice Hall, Englewood Cliffs, N.J., 1959.
- [62] Wikipedia. CAP Theorem (Brewer’s Theorem). URL https://en.wikipedia.org/wiki/CAP_theorem. Retrieved 2016-11-15.
- [63] Titus Winters, Tom Manshreck, and Hyrum Wright, editors. *Software Engineering at Google*. O’Reilly Media, 2020.
- [64] Hyrum Wright. Hyrum’s Law, 2018. URL <https://www.hyrumslaw.com/>. This concept is much older than Hyrum. For example, IBM has been maintaining backwards compatibility for officially undocumented features/bugs on their mainframes for over half a century.
- [65] Fritz Zwicky. *Discovery, Invention, Research — Through the Morphological Approach*. The Macmillian Company, Toronto, 1969.