ARIZONA STATE UNIVERSITY

GENERAL STUDIES COURSE PROPOSAL COVER FORM

Course information:

*Copy and paste **current** course information from Class Search/Course Catalog.*

College/School	Ira A. Fulton Schools of Engineering	Department/School	**CIDSE**

Prefix:	**SER**	Number:	**416**	Title:	Software Enterprise: Project and Process Management	Units:	3

Course description: **Project-centric course focusing on applying software process, project management, and technical leadership. Final course in the software enterprise sequence.**

Is this a cross-listed course? No If yes, please identify course(s):

Is this a shared course? No If so, list all academic units offering this course:

*Note- For courses that are crosslisted and/or shared, a letter of support from the chair/director of **each** department that offers the course is required for **each** designation requested. By submitting this letter of support, the chair/director agrees to ensure that all faculty teaching the course are aware of the General Studies designation(s) and will teach the course in a manner that meets the criteria for each approved designation.*

Is this a **permanent-numbered** course with topics? Yes

If **yes**, each topic requires **an individual submission**, separate from other topics.

Requested designation: Literacy and Critical Inquiry-L **Mandatory Review:** Yes

*Note- a **separate** proposal is required for each designation.*

Eligibility: Permanent numbered courses **must** have completed the university's review and approval process. For the rules governing approval of omnibus courses, contact Phyllis.Lucie@asu.edu.

Submission deadlines dates are as follow:

For Fall 2020 Effective Date: October 10, 2019 For Spring 2021 Effective Date: March 5, 2020

Area proposed course will serve:

A single course may be proposed for more than one core or awareness area. A course may satisfy a core area requirement and more than one awareness area requirements concurrently, but may not satisfy requirements in two core areas simultaneously, even if approved for those areas. With departmental consent, an approved General Studies course may be counted toward both the General Studies requirement and the major program of study. It is the responsibility of the chair/director to ensure that all faculty teaching the course are aware of the General Studies designation(s) and adhere to the above guidelines.

Checklists for general studies designations:

Complete and attach the appropriate checklist

Literacy and Critical Inquiry core courses (L)
Mathematics core courses (MA)
Computer/statistics/quantitative applications core courses (CS)
Humanities, Arts and Design core courses (HU)
Social-Behavioral Sciences core courses (SB)
Natural Sciences core courses (SQ/SG)
Cultural Diversity in the United States courses (C)
Global Awareness courses (G)
Historical Awareness courses (H)

A complete proposal should include:

- Signed course proposal cover form
- Criteria checklist for General Studies designation being requested
- Course catalog description
- Sample syllabus for the course
- Copy of table of contents from the textbook and list of required readings/books

It is respectfully requested that proposals are submitted electronically with all files compiled into one PDF.

Contact information:

Name	Srividya Bansal	E-mail	srividya.bansal@asu.edu	Phone	7-5107

Department Chair/Director approval: ***(Required)***

Chair/Director name (Typed): Srividya Bansal Date: 02-17-2020

Chair/Director (Signature): [illegible]

Arizona State University Criteria Checklist for

LITERACY AND CRITICAL INQUIRY - [L]

Rationale and Objectives

Literacy is here defined broadly as communicative competence—that is, competence in written and oral discourse. **Critical inquiry** involves the gathering, interpretation, and evaluation of evidence. Any field of university study may require unique critical skills that have little to do with language in the usual sense (words), but the analysis of written and spoken evidence pervades university study and everyday life. Thus, the General Studies requirements assume that all undergraduates should develop the ability to reason critically and communicate using the medium of language.

The requirement in Literacy and Critical Inquiry presumes, first, that training in literacy and critical inquiry must be sustained beyond traditional First Year English in order to create a habitual skill in every student; and, second, that the skill levels become more advanced, as well as more secure, as the student learns challenging subject matter. Thus, two courses beyond First Year English are required in order for students to meet the Literacy and Critical Inquiry requirement.

Most lower-level [L] courses are devoted primarily to the further development of critical skills in reading, writing, listening, speaking, or analysis of discourse. Upper-division [L] courses generally are courses in a particular discipline into which writing and critical thinking have been fully integrated as means of learning the content and, in most cases, demonstrating that it has been learned.
Notes:

1. ENG 101, 107 or ENG 105 must be prerequisites
2. Honors theses, XXX 493 meet [L] requirements
3. The list of criteria that must be satisfied for designation as a Literacy and Critical Inquiry [L] course is presented on the following page. This list will help you determine whether the current version of your course meets all of these requirements. If you decide to apply, please attach a current syllabus, or handouts, or other documentation that will provide sufficient information for the General Studies Council to make an informed decision regarding the status of your proposal.

Revised April 2014

Proposer: Please complete the following section and attach appropriate documentation.

ASU - [L] CRITERIA

TO QUALIFY FOR **[L]** DESIGNATION,THE COURSE DESIGN MUST PLACE A MAJOR EMPHASIS ON COMPLETING CRITICAL DISCOURSE--**AS EVIDENCED BY THE FOLLOWING CRITERIA:**

YES	NO		Identify Documentation Submitted
X		**CRITERION 1:** At least 50 percent of the grade in the course should depend upon writing assignments (see Criterion 3). Group projects are acceptable only if each student gathers, interprets, and evaluates evidence, and prepares a summary report. *In-class essay exams may not be used for [L] designation.*	Course Syllabus, Project Description, Lab Descriptions, example writing assignments (student work)

1. Please describe the assignments that are considered in the computation of course grades--and indicate the proportion of the final grade that is determined by each assignment.
2. **Also:**

Please **circle, underline, or otherwise mark** the information presented in the most recent course syllabus (or other material you have submitted) that verifies **this description** of the grading process--and label this information **"C-1".**

C-1

YES	NO		
X		**CRITERION 2:** The writing assignments should involve gathering, interpreting, and evaluating evidence. They should reflect critical inquiry, extending beyond opinion and/or reflection.	All lab assignments involve reflection questions that have to be answered by individual students (Lab descriptions included)

1. Please describe the way(s) in which this criterion is addressed in the course design.
2. **Also:**

Please **circle, underline,** or **otherwise mark** the information presented in the most recent course syllabus (or other material you have submitted) that verifies **this description** of the grading process--and label this information **"C-2".**

C-2

ASU - [L] CRITERIA			
X		**CRITERION 3:** The syllabus should include a minimum of two writing and/or speaking assignments that are substantial in depth, quality, and quantity. Substantial writing assignments entail sustained in-depth engagement with the material. Examples include research papers, reports, articles, essays, or speeches that reflect critical inquiry and evaluation. Assignments such as brief reaction papers, opinion pieces, reflections, discussion posts, and impromptu presentations are not considered substantial writing/speaking assignments.	1. Course Project involves substantial writing (project description and sample student work included). It involves writing as well as in-class presentation. 2. Lab 4 on Risk Management involves substative writing for last 2 activities involving at least 300 and at least 400 words respectively. 3. Lab 3 involves discussing student perspectives on Software Process Improvement with at least 500 words. 4. In a similar manner all other labs also have substantial writing involved.
1. Please provide relatively detailed descriptions of two or more substantial writing or speaking tasks that are included in the course requirements			
2. **Also:** Please **circle, underline,** or **otherwise mark** the information presented in the most recent course syllabus (or other material you have submitted) that verifies **this description** of the grading process--and label this information **"C-3".** **C-3**			

ASU - [L] CRITERIA

YES	NO		Identify Documentation Submitted
X		**CRITERION 4:** These substantial writing or speaking assignments should be arranged so that the students will get timely feedback from the instructor on each assignment in time to help them do better on subsequent assignments. *Intervention at earlier stages in the writing process is especially welcomed*.	Clear grading rubric for all written work is provided within the Project and Lab descriptions.

1. Please describe the sequence of course assignments--and the nature of the feedback the current (or most recent) course instructor provides to help students do better on subsequent assignments

2. **Also:**

Please **circle, underline,** or **otherwise mark** the information presented in the most recent course syllabus (or other material you have submitted) that verifies **this description** of the grading process--and label this information **"C-4".**

C-4

Course Prefix	Number	Title	General Studies Designation
SER	416	Software Enterprise: Project and Process Management	L

Explain in detail which student activities correspond to the specific designation criteria.
Please use the following organizer to explain how the criteria are being met.

Criteria (from checksheet)	How course meets spirit (contextualize specific examples in next column)	Please provide detailed evidence of how course meets criteria (i.e., where in syllabus)
C-1	The Exercises and Project categories of assessments on the syllabus consist primarily for outcomes directly connected to the L requirements, such as writing project management plans, defect estimation, defect tracking, risk management, software process improvement according to software engineering standards, and using evidence-based techniques to evaluate progress	In the attached syllabus the areas related are highlighted in yellow. Project and Lab descriptions (Labs 1 through 4) are attached with written/presentation work highlighted in yellow.
C-2	The project and exercises include activities to critically evaluate evidence gathered on a project and apply decision-making procedures across alternatives justified by that analysis	In the attached syllabus the areas related are highlighted in yellow. Project and Lab descriptions are attached with written/presentation work highlighted in yellow.
C-3	The writing and oral activities in the project context are examples of this. Example project activity descriptions and student work are included.	In the attached syllabus the areas related are highlighted in yellow. Further, the semester project description, sample student written work, lab descriptions with grading rubrics are attached outline.
C-4	The course is part of the Software Enterprise project-based learning sequence. In Enterprise courses students apply Kolb's active learning approach to get rapid feedback on project activities, and include metacognition (individual and team reflections) at frequent intervals. The Software Enterprise forms the core curricular design structure of the BS in SE, has received NSF and other funding, and has been published (sample references provided) in several	Lab descriptions are attached which include grading rubric highlighted in yellow. An Enterprise project description is included, and publications in the software engineering education research literature are available, including: 1. Gary, K. “The Software Enterprise: Practicing Best Practices in Software Engineering Education”, The International Journal of Engineering Education Special Issue on Trends in Software Engineering Education, Volume 24, Number 4, July 2008, pp. 705-716. 2. Gary, K., Lindquist, T., Bansal, S., and Ghazarian, A. “A Project Spine for Software Engineering Curricular Design”, Proceedings of the 26th Conference on Software Engineering Education & Training (CSEET 2013), Co-

	venues.	located with ICSE 2013, San Francisco, CA, May 2013. 3. Gary, K., “The Software Enterprise: Preparing Industry-ready Software Engineers” Software Engineering: Effective Teaching and Learning Approaches, (book chapter) Ellis, H., Demurjian, S., and Naveda, J.F., (eds.), Idea Group Publishing. October 2008. 4. Gary, K., Koehnemann, H., and Gannod, B. “The Software Enterprise: Facilitating the Industry Preparedness of Software Engineers” National Conference of the American Society for Engineering Education (ASEE 2006), Chicago, IL, June 2006.

Course Title and Catalog Description

SER 416 - Project and Process Management

Course description: Project-centric course focusing on applying software process, project management, and technical leadership. Final course in the software enterprise sequence.

Course Syllabus

SER 416 Project and Process Management, Spring 2019

Catalog Description

Project-centric course focusing on applying software process, project management, and technical leadership. Final course in the software enterprise sequence.

1. Contact Information

Instructors:	Dr. Kevin Gary, kgary@email.asu.edu, Zoom: https://zoom.us/my/drgary, (480)727-1373 Dr. Tyler Baron, tjbaron@asu.edu
Office:	Peralta 230C (Gary), Peralta 230X (Baron)
Class Meeting Time:	12:15-1:30 Mondays and Wednesdays
Classroom:	AGBC154
Schedule Line Number:	21404
Class Website:	Canvas
Teaching Assistant:	Mr. Aditya Vikram, avikram4@asu.edu

2. Office Hours

Dr. Gary Office Hours:	TTH 10:30pm-12:30pm, other days/times by appointment
Dr. Baron Office Hours:	TBA
TA Office Hours:	W 2-4pm and TH 1-3pm in Peralta 208

3. Course Objectives and Expected Learning Outcomes

After successfully completing SER416, the student will (Program Outcomes in parentheses):

CO-1. (ABET-3) Communicate, in oral and written form, project and quality management plans.

CO-2. (ABET-4) Conduct tradeoff analyses to determine the best course of action in scalable projects considering a range of contextual (global, economic, environmental, societal, etc.) attributes.

CO-3. (ABET-5) Demonstrate competency in establishing and tracking project goals, plans, and objectives.

CO-4. (ABET-6) Analyze and interpret project, process, and quality data to make informed judgements about a software project's success trajectory.

CO-5. (SER1) Apply modern project and quality management tools and techniques common to software engineering best practices.

4. Grading Policies

Assessment	Percent of Grade
Labs	35%
Project	25%
Exam 1	20%
Exam 2	20%

Labs: There will be assigned small team labs counted for the course lab grade. Specific instructions on the requirements, grading criteria, and submission process will be given when the assignments are handed out. Lab assignments will be started in-class in small teams and completed outside of class. After completing as a small team, there will be a short individual reflection exercise.

Project: Software Enterprise courses are project-based. A class period will be reserved for project kickoff in mid-February, and one week mid-April is reserved for final class presentations. The project handout will have specific rubrics, but in general the project grade will be 5% class presentation, 5% peer review, 7.5% written deliverables (based on course modules), and 7.5% critical analysis of scalable software project and quality data. **→ C2 and C3**

Exams: 2 exams will be given. These exams are scheduled for Wednesday March 13th, and Wednesday May 1 (the scheduled day for our class on the ASU final exam schedule). The second exam is not cumulative, though understanding of prior material will increase chances for success in the second exam.

Grade Appeals: Students have the right to appeal a grade in writing. Submit your typed appeal (email is fine) with the graded lab or exam, stating the reason for the appeal. All appeals must be turned in no later than one week after the grade has been posted.

5. Absence and Make-Up Policies

There is not a mandatory attendance policy for this course. However, assessments under "Labs" are not excused (if you miss the lab class you cannot turn it in), nor are absences from exams. Students unable to attend class, take exams, or start lab assignments due to a medical condition must present a doctor's signed excuse and notify the instructor *as soon as the condition affects the student's work*.

Accommodations will be made for religious observances provided that students *notify the instructor at the beginning of the semester concerning those dates*. Students who expect to miss class due to officially university-sanctioned activities should *inform the instructor early in the semester*. Alternative arrangements will generally be made for any examinations and other graded in-class work affected by such absences. Please see ACD 304–04, "Accommodation for Religious Practices" and ACD 304–02, "Missed Classes Due to University-Sanctioned Activities" for more information.

6. Readings, Special Materials, and Required Activities

Readings:

- *(Required) Software Engineering (10th ed)*, Ian Sommerville. Addison-Wesley (2016); ISBN-10: 0-13-394303-8, ISBN-13: 978-0-13-394303-0
- *(Required)* Additional readings will be posted on the class website.
- *(Recommended)* The Software Engineering Body of Knowledge (SWEBOK) version 3 (P. Bourque and R.E. Fairley, eds., *Guide to the Software Engineering Body of Knowledge, Version 3.0*, IEEE Computer Society, 2014; www.swebok.org.

Class Schedule:

Wk	Topic	Lab
1	Quality Mgmt	Quality Mgmt
2	Defect Tracking	DT/QM
3	Defect Mgmt	Defect Estimation
4	Testing	PROJECT KICKOFF
5	Open Source	OpenSource TechEval
6	Risk Management	Risk lab
	SPRING BREAK	
7	PM Overview	MIDTERM
8	WBS	WBS
9	Task Sequencing	TANs/Critical Paths
10	Task Scheduling	PERT/Gantt charts
11	Task Tracking	Earned Value Analysis
12	Project presentations this week (6 per class)	
13	Postmortem	Postmortem analysis

7. Classroom Behavior

Cell phones and pagers must be turned off during class to avoid causing distractions. Exceptions may be accommodated for personal reasons with advance approval of the instructor. The use of recording devices is permitted during class only with prior approval of the instructor. Any violent or threatening conduct by an ASU student in this class will be reported to the ASU Police Department and the Office of the Dean of Students.

8. Academic Integrity

All students in this class are subject to ASU's Academic Integrity Policy (available at http://provost.asu.edu/academicintegrity) and should acquaint themselves with its content and requirements, including a strict prohibition against plagiarism. All violations will be reported to the Dean's office, who maintain records of all offenses. Students are expected to abide by the FSE Honor Code (http://engineering.asu.edu/integrity/).

The Student Academic Integrity Policy of Arizona State University requires each student to act with honesty and integrity and to respect the rights of others in carrying out all academic assignments. There are a number of actions that constitute a violation of the policy. These actions in this course include, but are not limited to:

1) practicing any form of academic deceit;
2) referring to materials or sources or employing devices (e.g., audio recorders, crib sheets, calculators, solution manuals, or commercial services) not specifically authorized by the instructor for use during tests, quizzes, homework, and class activities;
3) acting as a substitute for another person in any academic evaluation or using a substitute in any academic evaluation;
4) possessing, buying, selling, or otherwise obtaining or using, without appropriate authorization, a copy of any materials intended to be used for academic evaluation in advance of its administration;
5) depending on the aid of others to the extent that the work is not representative of the student's abilities, knowing or having good reason to believe that this aid is not authorized by the instructor;
6) providing inappropriate aid to another person, knowing or having good reason to believe the aid is not authorized by the instructor;
7) submitting the ideas or work of another person or persons without customary and proper acknowledgment of sources (i.e., engaging in plagiarism);
8) permitting one's own ideas or work to be submitted by another person without the instructor's authorization; or attempting to influence or change any academic evaluation or record for reasons having no relevance to class achievement.
9) turning in work done by someone else or another pair/group
10)copying work done by someone else or another pair/group

A common question is the use of material that is "googled" or found on popular sites such as StackOverflow. Items 5 and 7 pertain to this situation. Most software engineers use reference examples, found in print or online. This is fine as a practice but is not acceptable in situations where you are using the examples to proxy *your understanding of the concepts* applied in that assessment (i.e. lab or project activity). First, if you are uncertain if it is allowable or not, verify directly with the instructor before submitting the assignment. Second, if it is allowable, you are still required to a) adhere to all originating author's constraints on the use and licensing of the material, and b) provide proper attribution (full bibliographic reference and proper citation). Failure to do so constitutes a violation of this Academic Integrity Policy.

Students will be asked to work in small teams on lab and a class project. You are to work with your partners and only your partners as directed by the instructor; receiving assistance from anyone else other than your partners, teaching assistants, approved tutors, or the instructor is considered a violation of this Academic Integrity Policy. Further, on any paired/group assessments you remain individually responsible for the entire solution – you must understand it fully, and there will be no differentiated grades awarded between the individuals in the pair/group. From an ethics standpoint, you have a professional responsibility to your partner to give your best effort on each assignment and project deliverable. Failure to do so will be considered an ethics violation.

The penalty for an Academic Integrity Violation (cheating) on a lab or project deliverable will be a reduction of a course letter grade for the first offense, and failure of the course for a second offense. The penalty for an Academic Integrity Violation (cheating) on an exam is immediate failure of the course. The penalty for an ethics violation will be a zero for the lab or project deliverable grade. All violations will be referred to the Dean's Office of the Ira A. Fulton Schools of Engineering.

9. Disability Accommodations

Suitable accommodations will be made for students having disabilities and students should notify the instructor as early as possible if they will require same. Such students must be registered with the Disability Resource Center and provide documentation to that effect.

10. Additional Notices

Notice: Any information in this syllabus may be subject to change with reasonable advance notice.
Notice: All contents of these lectures, including written materials distributed to the class and recorded videos, are under copyright protection. These materials may not be sold, distributed, or commercialized without the express permission of the instructor, ASU, and the Arizona Board of Regents. See ACD 304-06 for more info.

11. Sexual Discrimination

Title IX is a federal law that provides that no person be excluded on the basis of sex from participation in, be denied benefits of, or be subjected to discrimination under any education program or activity. Both Title IX and university policy make clear that sexual violence and harassment based on sex is prohibited. An individual who believes they have been subjected to sexual violence or harassed on the basis of sex can seek support, including counseling and academic support, from the university. If you or someone you know has been harassed on the basis of sex or sexually assaulted, you can find information and resources at https://sexualviolenceprevention.asu.edu/faqs.

As a mandated reporter, I am obligated to report any information I become aware of regarding alleged acts of sexual discrimination, including sexual violence and dating violence. ASU Counseling Services, https://eoss.asu.edu/counseling, is available if you wish discuss any concerns confidentially and privately.

ADDITIONAL COURSE INFORMATION

Prerequisites

- Formal prerequisite: SER316
- It is helpful to have taken SER415 but it is not a strict prerequisite.
- **Command of the English language and an attention to detail. Most of this course grade is assessed through communication-oriented deliverables in the form of oral and written documents and analyses. These are expected to be prepared (as taught) in the best practices of the software engineering discipline, and with a level of professionalism befitting senior students about to complete an engineering degree and perhaps enter the professional workplace.**
- A desire to learn and participate in class.

Additional Course Description

Welcome to the Software Enterprise! In this last course of a 5-6 course sequence, you will learn the basic principles of software process, project, and quality management. The intent is not to make you ready to manage scalable software projects, but to gain an awareness of the broader contexts of complex engineering projects that involve a multitude of facets (people, processes, techniques, and tools) and require a methodological approach to informed decision making. It is hoped that this final Software Enterprise experience will be that last step you need to make the transition from student to contributing software professional!

Contents

SER416 Software Enterprise IV **Spring 2019**
Project Topics and Description **due April 14 2019**

Objectives:

- Develop deep knowledge of an advanced software engineering topic not covered in class
- Demonstrate communication proficiency (written and verbal).
- Broaden understanding of the role and impact of software engineering in society.

Overview:
As software engineering students in the last (or close to last) semester of study, it is important that you have a broad understanding of the software engineering profession. To date the majority of your curriculum has focused on technical skills, mainly programming, software engineering specific skills (the phases of the software lifecycle process), and "soft skills" including teamwork and communication (through your Enterprise classes). The intent of this project is to broaden your understanding of the profession through an in-depth exploration of SE, either through study of an advanced SE topic, or the application of SE in a vertical or horizontal domain. No matter which choice your team makes, you will follow the same set of activities, deliverables, and grading rubrics.

Project Topics:
As mentioned above in the Overview you can select from advanced SE, vertical, or horizontal domains. Advanced SE provides for in-depth exploration in-discipline but not limited to a specific domain. Vertical domains are domains, or markets, of applications (see https://en.wikipedia.org/wiki/Vertical_market). Horizontal domains (see same Wikipedia page) support a wide variety of markets.

Advanced SE Topics
The set of advanced SE topics comes mostly from your textbook chapters, see table below. I also added two special current topics at the bottom.

Topic	References	Description
Maintenance & Evolution	Sommerville Ch. 9, SWEBOKv3 Ch. 5	There should be a specific focus on large-scale systems.
Dependability & Reliability	Sommerville Ch. 10,11	A project should consider technical approaches and non-technical concerns (e.g. regulatory compliance as described in Sommerville).
Software Safety	Sommerville Ch. 12+27 (web)	The safety of software refers to its ability to operate when failures may lead to a loss of life. The techniques one employs in this domain (such as formal methods) tend to be specialized based on this requirement.
Security	Sommerville Ch. 13, 14	Cybersecurity is a hot career path right now. A project should consider it from an SE perspective, including not just technical approaches (static analysis) but process approaches (risk management, audits, etc.)
SE Economics	SWEBOKv3 Ch. 12	The economics of software engineering has the difficult task of attempting to define specific project costs, down to the $ per SLOC, for engineered software. This includes all process overhead, most notably quality improvement factors. I have Capers Jones' book on the topic.
Scaled Agility	SAFe, LeSS, DAD	Scaling up Agile. New process models are being defined to help agile projects scale up and out across organizations. These are 3 of the latest models, and of course there is "standard" Agile practices like "Scrum of Scrums" to consider.

The advantage of most of these topics is you have ready-made starting points in your textbook, so finding technical material, case studies, and additional references will be easier.

Vertical SE Applications
As you enter your career you may find you work for a software product or services company in a specific vertical market, or on a category of software (horizontal). It will behoove your career path to understand as much as possible about the business of that domain, and the intersection of that domain and your chosen profession (software engineering).

The list of vertical domains, a brief description, and some starting point suggestions are provided below:

Topic	Domain	Description
Defense Systems	Vertical	SE applied to US Department of Defense Systems, including but not limited to logistics, C4I, weapons, etc. A project should explore the availability and reliability aspects of such systems.
Healthcare Systems	Vertical	SE applied to clinical life-critical systems such as surgical systems. A project should explore the safety and reliability aspects of such systems.
Avionics	Vertical	SE applied to aviation (specifically, commercial airplanes). A project should explore the safety and reliability aspects of such systems.
Space	Vertical	SE applied to Satellite and spaceship systems (NASA). A project should explore the availability and reliability aspects of such systems.
Energy	Vertical	SE applied to Solar, Hydro, Oil, or Gas management systems. A project should explore the safety and reliability aspects of such systems.

You will find that chapters 19-21 in your textbook will be helpful to a number of these as I have purposefully selected system-related domains.

Horizontal SE Applications
As mentioned above, horizontal domains mean you design and create solutions centered around a type of technology or engineering that may be applied across several verticals (in fact you often see horizontal companies describe their vertical offerings on their websites). Some interesting horizontal domains are below:

Topic	Domain	Description
SE in Robotics	Horizontal	SE in Robotics-based systems, such as industrial robots or medical device robots
SE in IoT	Horizontal	SE Internet-of-Things systems.
Evolution of Open Source Software	Horizontal	How has OSS changed Software Engineering? How do OSS communities differ in their processes from commercial entities? What is the Total Cost of Ownership (TCO) of OSS? How does one evaluate the suitability of OSS?
Adaptive & Self-Managed or Autonomic Systems	Horizontal	SE for Adaptive and Self-Managing Systems is relatively new and considers systems that adapt their behaviors or are dynamically resilient to operational changes. This fundamentally changes the way we think about software maintenance and evolution (Sommerville Ch. 9, SWEBOKv3 Ch. 5). SEAMS is a symposium on this topic.

For all projects I have some additional SE textbooks that you may find helpful as well when getting your starting point of references and background material.

Deliverables:

1. Project report: your team will submit a project report that is at least 20 pages and no more than 30 pages (not including title page, TOC, or Appendices). The report must be in Microsoft Word .docx format, used 1’’ margins on all 4 sides, Letter (8.5’’x11’’) paper, 12 point Times New Roman font, and 2’’ spacing. Please see the format specifications at https://advances.asee.org/guide-for-authors/
2. Presentation: your team will present its project in class during the week of April 15. Presentations must be at least 8 minutes long and no more than 10 minutes long, with 2 minutes reserved for questions.

Due Dates:

- The project report is due Friday April 12th via Canvas submission at 11:59pm
  - Teams may submit a draft by Friday April 5th to get feedback on their writing.
- The project presentation materials are due Sunday April 14th via Canvas submission at 11:59pm.
  - Presentations will take place in class Monday 4/15 and Wednesday 4/17. The order of presentations will NOT be specified ahead of time. Presenters will not be allowed to use materials not turned in by the April 14th submission deadline.

Grading Rubrics:

Area	Pts	Description	Rubric
Paper (76% or 190 points)			
Writing (77)	10	Proper spelling, punctuation, and grammar	1 pt deducted for every 3 errors (rounded up)
	10	Meets required length	1 pt deducted for every ½ page below page threshold
	5	Meets format specifications	1 pt deducted every 2 errors (rounded up). May impact length.
	7	Adequate number of references. Minimum references = 8, At least 4 must be non-Web URLs, at least 6 must be peer-reviewed / edited content.	1 pt deduction for each reference that does not meet criteria
	5	Use of proper citation format according to template.	1 pt deducted for every 2 citation errors (rounded up).
	10	Appropriate citations and quotations.	1 pt deducted per error including overuse of direct quotations & missing citations. Multiple missing citations will be considered plagiarism and subject to an AIV and 0 grade for the paper.
	30	Quality. The paper must use clear and concise phrasing, proper section and paragraph organization including topic sentences and transition language.	1 pt deduction for each error.
Case studies (23)	6	Case study 1 is relevant to the paper topic.	Case study is either highly relevant, moderately relevant -2 (discusses topic but is not central theme of the study), slightly relevant (mentions topic) -4 or not relevant at all -6
	4	Case study 1 is summarized in sufficient depth	Not a length requirement but length can play a role. The case study must be described in enough depth to make a connection to the paper topic. Point deductions: Excellent 0, Sufficient -1, Insufficient but discussed -2, Deficient -3, Absent or minimal -4
	6	Case study 2 is relevant to the paper topic.	Case study is either highly relevant, moderately relevant -2 (discusses topic but is not central theme of the study), slightly relevant (mentions topic) -4 or not relevant at all -6
	4	Case study 2 is summarized in sufficient depth	Not a length requirement but length can play a role. The case study must be described in enough depth to make a connection to the paper topic. Point deductions: Excellent 0, Sufficient -1, Insufficient but discussed -2, Deficient -3, Absent or minimal -4
	3	One case study must be from 2010 or later	-1 2005-2009, -2 2000-2004, -3 if pre-2000. The year refers to the case itself, not the year of publication of the case.
Technical Content (90)	40	Technical summary. The paper should provide a detailed technical summary of the area that cites at least 3 different references. This is expected to be at least 5 pages long.	2 point deduction for each ½ page below the expected 5 pages (max 12). -3 for each missing reference, -1 or -2 if reference present but inadequate or deficient (max 9). Technical accuracy: 0 for correct [-1,-3] for inadequate (lacking depth), [-4,-6] for deficient (aspects missing or incorrect), [-7,-9] for gross incorrectness or missing.
	20	Discussion of relevant standards. Standards will vary by topic, but all topics have multiple relevant standards (past and present).	Minimum expected standards referenced: 2. -5 for only 1. -2 if standard moderately related to topic, -4 if standard slightly relevant to topic, -5 if no relevance/incorrect. 5 points for connecting standard to topic and explaining its significance (does industry/government/academia/etc. adheres to the standard?)
	10	Current and future impact discussion. Papers must include a section of the popularity or other measure of significance / importance of this topic.	Current impact explained with evidence: 5 points. Adequate [-1,-2], Deficient [-3,-4], Missing/Incorrect -5. Future impact explained with evidence: 5 points. Adequate [-1,-2], Deficient [-3,-4], Missing/Incorrect -5.
	20	Discussion of the impact on the phases of the Software Development Lifecycle (SDLC). Papers should explain how the topic impacts or emphasizes two or more phases of the SDLC.	SDLC phase identified and evidence-based discussion presented: 0. Adequate discussion (phase identified but impact lacks depth) [-1,-5], Deficient discussion (phase identified but cursory presentation of impact) [-6,-10], Incorrect (aspects of the discussion suffer from technical deficiencies) [-11,-15], Missing (minimal with phase poorly identified and significant issues with technical discussion) [-16,-20]. The point breakdowns will be spread over the number of phases discussed. If only one phase is discussed, halve these breakdowns and -10.
Presentation (24%, or 60 points)			
	7	2 or more presenters, no presenter more than 60%	1 presenter -4, 1 presenter > 60% -2, Transition(s) between presenters 1 pt
	10	Time management and presentation cadence	1 pt deducted for each minute over 10 minutes or under 8 minutes. -1 for hard stop ordered (12 minute, with deduction for no questions). -2 for significantly rushed presentation, -1 for slightly rushed. -2 for significant deficiency in apportionment of time, -1 for slight deficiency. [-1,-3] for not speaking clearly and coherently.
	9	Handling questions. 2 minutes at the end to handle questions from the audience	Excellent: 0. Adequate (questions answered to mostly acceptable level) [-1,-3], Deficient (poor answering of questions, lack of understanding or answering the wrong questions) [-4, -6], Missing -9. 1 pt deduction per insufficient or over time.
	10	Technical summary. Ability to summarize the technical topic via presentation modality (definition, example, relation to existing). Expect 3-4 minutes	Excellent: 0. Adequate (conveys sufficient technical content but not in format consumable in brief presentation) [-1,-3], Deficient (missing technical aspects) [-4, -6], Incorrect (serious issues with technical correctness) [-7,-9], Missing -10.
	10	Case study. Team need only present one of the case studies from the paper, though both are allowed if the team chooses (but then both must meet rubric and team must manage time). Expect 2-3 minutes	Excellent: 0. Adequate (summarizes case and sufficiently connects it to the topic) [-1,-3], Deficient (summarizes case and insufficiently connects it to the topic) [-4,-6], Incorrect (significant technical errors in case presentation and/or lack of connection to the topic) [-7,-9], Missing -10.
	7	Current and future impacts summarized. The presentation should list current and future impacts and select 1 current and 1 future impact to discuss in a modest amount of depth. Expect 1-2 minutes	Excellent: 0. Adequate (List of impacts given w/ 1 missing current/future discussion) [-1,-3], Deficient (incomplete list of impacts w/ no current/future discussion) [-4, -6], Missing -7
	7	Identify & connect standards to topic. This may be in the context of the technical summary if that makes the presentation flow better. Expect 1-2 minutes.	Excellent: 0. Adequate (Standards presented w/ 1 missing discussion on connection to topic) [-1,-3], Deficient (incomplete list of standards w/ no connection discussion) [-4, -6], Missing -7

LAB 1 Description

SER416 Software Enterprise IV: Process and Project Management **Spring 2019**
Lab: Quality Management and Defect Tracking

Instructions:
Activities 1 and 2 must be done in a team of 3. Activities 3 and 4 I expect you to do individually. Each student should make an individual submission by the due date (Tuesday February 5th at 11:59pm) with their copy of all Activities. Please indicate at the top your team partners. Submission done electronically to Canvas with a file type of .docx, .pptx, .pdf, or OpenOffice (.odt, .odp, .ott, .otp, .odg).

General Grading Rubrics: → **C4**

1. I expect written answers to demonstrate proper use of spelling, grammar, punctuation, and active voice.
2. I expect the writing style to be persuasive and evidence-based. It should make a clear coherent argument, report findings based on evidence, and/or
3. Provide proper attribution in all places you refer to an outside reference. Specifically, include a reference list and cite in IEEE style (see https://en.wikipedia.org/wiki/IEEE_style).
4. All students are responsible for a team's shared answers. If the answer is plagiarized all students on the team will receive the AIV penalty. No grade appeals are allowed based on "I didn't do that Activity".
5. Individuals are responsible for Individual Activities. Sharing of answers of these Activities is an AIV.

As quality of writing will be taken into account, I strongly suggest the team work in iterations – at least one iteration to get the evidence down for each shared Activity, one to discuss the questions requiring discussion (taking a position on something), one to draft the polished language, and one to polish the presentation (final review). Keep in mind as well that sometimes Individual Activities rely on team Activities so do not wait until Tuesday night!

Activity 1: Quality Management Standards (12 points)
I posted the IEEE-730 standard on the class website. Review this document, looking for cross-references to other IEEE standards, or other standards in general. For each major phase of the software lifecycle, find two places in IEEE-730 that pertain to the phase, and find cross-references to other standards and a description of the QA activities pertaining to the phase. You may find it useful to Google some of the IEEE standards for summaries, or refer to their descriptions on www.ieee.org. Specifically fill in the following table:

Software Process Lifecycle Phase	IEEE-730 section & page reference	Other (IEEE) standard references	Description of other activities performed in this phase as discussed in IEEE-730
Requirements			•
Analysis & Design			
Implementation			
Deployment/ Operation			
Maintenance			
Cross-cutting			

"Cross-cutting" here refers to QA activities described in IEEE-730 that apply to more than one, and possibly all, software lifecycle phases. It is my hope that mapping these activities will give you some guidance on mapping your project deliverables to IEEE-730 areas, and therefore give you a head start on doing your QA plan.

Activity 2: Defect Tracking (22 points)

A. (1) Explain why a software defect is called a *bug*.
B. (2) In class we discussed *Failures – Faults – Errors (Defects)*. The SWEBOK version 3 identifies 5 categories, not 3. What are they? How do they overlap with these 3?
C. (9) This blog post: https://blog.capterra.com/top-free-bug-tracking-software/ summarizes 10 open source defect tracking tools. Pick any 3 and i) provide a key feature summary, ii) define the default roles available in the system (and what features they support), and iii) create a graph of the default defect-handling states and transitions (like your notes). For (i) and (ii) I am fine if you provide a table, though make sure you define what the key features are, not just label them.
D. (4) GitHub includes an Issues feature where one can create an issue and label it a *bug*. GitHub and the larger GitHub community also post a number of workflows to handle various development patterns. Find 2 GitHub development workflows related to bug issue handling – describe them and provide a state diagram for each.
E. (6) Agile methods such as Scrum and XP favor practices such as Continuous Integration and Testing through automated unit tests and automated static analysis. Answer the following:
    a. Should all defects found in CI&Test be entered in a defect tracker? Why or why not? Explain. If your answer is "only some should" then provide a threshold definition.

b. Agile methods, XP in particular, also promotes Pair Programming and informal code reviews. Should all defects found in Pair Programming / informal code reviews be entered in a defect tracker? Why or why not? Explain. If your answer is “only some should” then provide a threshold definition.

c. In SER216 you used a “lite” version of the PSP by Watts Humphrey. Humphrey believed in a broad definition of a *defect* (see your notes) and in logging all defects, even those found by code review *before the code is even compiled*. This is counter to the Agile mindset, however provide a justification as to why this may be a good idea. It may help to review Humphrey’s motivations in this technical report: https://resources.sei.cmu.edu/asset_files/TechnicalReport/2000_005_001_13751.pdf

Activity 3 (6 points, INDIVIDUAL):

A. In Activity 2 C and D you should have looked at 4 tools. Pick the best and worst tools from amongst these 4 and explain what specific features of the tool lead you to these conclusions.

B. You also should have a total of 5 state diagrams from Activity 2 C and D, and you have 2 in your notes. Pick the best and worst state models from these 7 models and explain why you chose the models you did.

C. Draw a state model of your own that takes the best from these models. Explain how you derived this model (what ideas you got from what other models).

Activity 4 Reflection (10 points, INDIVIDUAL):

A. Describe what defect tracking tool or process your Capstone team is using if you are in the Capstone course. If your team does not yet have one, indicate what you will recommend to them and why. List your Capstone team members. If you are not on a Capstone team, then use your SER316 team.

B. Discuss (at least 200 words) your perspective on defect tracking. Is it an important quality process? Is it more or less important than other quality activities (testing, code reviews, static analysis, etc.)? What do you think the pros and cons are, and how would you determine the best defect tracking process for a given project? That is, what project attributes will influence what type of defect tracking you think that project should do?

LAB 2 Description

SER416 Software Enterprise IV: Process and Project Management **Spring 2019**
Lab: Defect Estimation **due February 12, 2019**

Objectives:

1. Understand how to estimate the total number of defects for a project using the Capture-Recapture method.
2. Know the importance of defect containment by understanding the play between defect injection and removal rates.

Activity 1: Capture-Recapture Method

Part 1: Compute a defect estimate as per the method in the Schofield paper using the defect data in the table below:

	Huey	Dewey	Louie
Defect 1	X	X	
Defect 2		X	
Defect 3			X
Defect 4		X	
Defect 5	X	X	X
Defect 6		X	
Defect 7		X	
Defect 8	X		
Defect 9		X	X

Add the needed columns to the table to show your work.
Answer: What is the defect discovery rate for the Huey/Dewey/Louie team? Explain.

Part 2: Compute a defect estimate again, this time from reviews from 4 developers:

	Paul	John	George	Ringo
Memory Leak			X	X
DB connection not closed	X		X	
Does not close file		X		
Incorrect parameter type	X		X	
Improper indentation	X	X	X	X
Lack of code comments			X	
Null pointer Exception	X	X	X	
Type mismatch error				X
Incorrect logic condition		X	X	X
Incorrect computation			X	
Improper use of synch block		X		
Uses goto	X	X		
Incorrect initialization of var			X	X
Improper access level	X			X

Add the needed columns to the table to show your work.
Answer: What is the defect discovery rate for the Beatles team? Explain.

Activity 2: Defect Containment Derivation

Consider the following information about a hypothetical application for PestSoftware Inc.:

- The application has 180,000 lines of code (LOC)
- PestSoftware uses RUP, so it has phases (I)nception, (E)laboration, (C)onstruction, and (T)ransition.
- There is a defect injected for every 15 LOC.
- The *defect injection distribution rates* per phase are: I = 10%, E = 25%, C = 45%, and T = 20%
- The *defect removal rates* per phase are: I = 60%, E = 80%, C = 67%, and T = 50%
- The *cost ratios* of fixing defects are: I:E:C:T:F → 1:5:10:50:100, where F means the defect has made it to "(F)ield operation". Assume a defect always costs 1 "unit" to fix if found in phase.

Use this information to answers the following questions.

1. How many total defects exist in the application?
2. How many defects exist per phase?
3. Fill in the matrix below. In this matrix, each entry has 2 cells. Fill in the left one with the number of defects from the column phase, and the right one with the number of defects removed in this phase.

	Inception		**Elaboration**		**Construction**		**Transition**		**Total**	
Injected/Removed	Inj	Rem	Inj	Remo	Inj	Rem	Inj	Rem	Inj	Rem
Inception			▨	▨	▨	▨	▨	▨		
Elaboration					▨	▨	▨	▨		
Construction							▨	▨		
Transition										
Field operation										

As an example: Suppose there are 100 defects in a system. The defect injection rate says that 10%, or 10 of them, were injected during Inception. Then the left cell of the 1st row/column (Inception/Inception) would be 10. The right cell would be the defects removed *in-phase*, which would be 6 since the defect removal rate for Inception is 60%. That would leave 4 defects carried over to Elaboration, entered in row Elaboration and column Inception – and so on.

4. What are the total costs per phase for injected defects? (i.e. how much did all of the defects for Inception cost you? Elaboration? etc. Keep in mind that a defect fixed in phase costs 1 unit, and ratio from there).
5. What is the cost-per-defect for each phase? You can derive this from your answer to #4.
6. What is the overall project defect removal rate?

Activity 3: Defect Depletion Activity

Some time ago a software engineer named Ed Weller put together a simple Defect Deflection worksheet in Excel. Please download this sheet. Defect depletion is somewhat like your previous problem, except it examines your specific defect removal processes and costs between *inspection* activities in requirements, analysis, and design (high-level and low-level) and *test* activities (unit, integration, and system testing). This model does assume a waterfall-like development process (so when you read the formulas going top-down and left-right, you can see how they are derived from previous activities). Weller's model is not based on any established standard or study, just a personal model he posted to a software engineering quality site. But spreadsheets like these are not uncommon (we had a capstone project with General Dynamics once where the team had to replace over 15K of Excel macros!).

The spreadsheet has a tab already filled in for Aardvark.org. Create 2 copies of the model on 2 separate tabs (name each tab for the organization name below. Enter the following data on each yellow highlighted line:

BewareBugs Inc:

Size in KLOC: 1	Req	Ana	HLD	LLD	Code	Unit Test	Integ Test	Sys Test
Injected	2.5	3.75	5	15	20	0.5	0.25	0.1
Removal Rate	0.8	0.7	0.5	0.4	0.7	0.3	0.7	0.7
Cost	5	3	3	2	1	1	16	40

Cantankerous Coders, LLC:

Size in KLOC: 1	Req	Ana	HLD	LLD	Code	Unit Test	Integ Test	Sys Test
Injected	1.5	2.5	10	45	50	1	0.5	0.5
Removal Rate	0.5	0.5	0.8	0.7	0.9	0.8	0.4	0.4
Cost	2	1	5	3	3	2	12	32

Answer the following:

1. What is Aardvark.org good at? What are they bad at?
2. What is BewareBugs good at? What are they bad at?
3. What is CantankerousCoders good at? What are they bad at?
4. What is the most costly quality problem for Aardvark.org? For BewareBugs? For Cantankerous Coders? Explain.
5. Identify ONE thing you think each organization (including Aardvark.org) needs to do to improve?

Activity 4: INDIVIDUAL Thought Questions

1. Suppose in Part 1 of CRM, that you found out that a) Huey seems to only find coding style errors while Louie only finds memory-related defects, and b) Dewey was kind of tired so he only skimmed the last 25% of the code. Would this information shake your faith that the CRM-based estimate you derived was appropriate? Explain why or why not for both (a) and (b).
2. If you asked to pick only one phase of the software process for PestControl Inc. to improve, what phase would it be, and what activity would you focus on (injection, discovery, or removal)? Explain your answer to both.
3. Is there any additional information that is missing so far that would have helped you answer #2 better?
4. What are the challenges in applying *defect containment* models in Agile software development processes? Is there a way to adapt them to this purpose? I'd like you to answer this question by yourself first, just thinking about it. Then, using Google as your friend, see what the community says.

Activity 5: Individual Reflection

This activity is to be completed individually by each student separate from their team! Answer the following.

Reflection questions:

1. Do you think *CRM* is a useful technique to do on real software projects?
2. Do you think *CRM* will benefit your capstone (316) project? How will it, how will it not? Do you think a different defect estimation method will work better?
3. In general, what do you think the main benefits and challenges are in *defect estimation*?
4. In general, what do you think the main benefits and challenges are in *defect containment*?

LAB 3 Description

SER416 Software Enterprise IV: Process and Project Management **Spring 2019**
Lab: Software Process Improvement

Instructions: ***Note: Please do Activity 3.1 in class on February 13th!***
Activities 1-3 must be done with your Capstone team. Activities 4 and 5 I expect you to do individually. Each student should make an individual submission by the due date (Tuesday February 19th at 11:59pm) with their copy of all Activities. Please indicate at the top your team partners. Submission done electronically to Canvas with a file type of .docx, .pptx, .pdf, or OpenOffice.

General Grading Rubrics: → C4
1. I expect written answers to demonstrate proper use of spelling, grammar, punctuation, and active voice.
2. I expect the writing style to be persuasive and evidence-based. It should make a clear coherent argument, report findings based on evidence, and/or
3. Provide proper attribution in all places you refer to an outside reference. Specifically, include a reference list and cite in IEEE style (see https://en.wikipedia.org/wiki/IEEE_style).
4. All students are responsible for a team's shared answers. If the answer is plagiarized all students on the team will receive the AIV penalty. No grade appeals are allowed based on "I didn't do that Activity".
5. Individuals are responsible for Individual Activities. Sharing of answers of these Activities is an AIV.

As quality of writing will be taken into account, I strongly suggest the team work in iterations – at least one iteration to get the evidence down for each shared Activity, one to discuss the questions requiring discussion (taking a position on something), one to draft the polished language, and one to polish the presentation (final review). Keep in mind as well that sometimes Individual Activities rely on team Activities so do not wait until Tuesday night!

Activity 1: Software Process Improvement Standards (11 points)
On Canvas you will find a copy of the IDEAL software process improvement model.
1. On slide 4 of your notes, it talks about process measurement, analysis and change. Identify sections of the IDEAL standard that discuss each of these aspects and summarize what it says about them. Include references to specific sections and page numbers.
2. On slide 8 of your notes, it describes, at a high level, 5 stages to conducting a software process improvement change. To what extent do these stages appear in IDEAL (Note: Don't expect it to be as simple as finding a section named for one of the stages, like "Process change introduction" – you will have to read and understand to map a section to a stage).

I expect these discussions to be significant and show some depth. If it helps, I expect at least 300 words per each part 1 & 2 or at least 600 words total. Refer to the general grading rubrics on writing above!

Activity 2: Software Process Improvement Examples (12 points)
On Canvas you will find two reports, one by Raytheon and one by Nationwide, written as recipients of the 2016 Watts S. Humphrey Software Process Achievement Awards. For each report, select any three of the process attributes in Figure 26.2 of Chapter 26 of Sommerville (also available on Canvas) and discuss how that organization provided a high level of that process attribute. You do not have to use the same process attributes for each organization. I expect at least 125 words per process attribute per organization, or at least 750 words total. Refer to the general grading rubrics on writing above!

Activity 3: Software Process Improvement and Agile methods (12 points)
1. **DO THIS ONE IN CLASS!** Organize yourself in your capstone teams. Execute the "6 hats retrospective" as described here: http://retrospectivewiki.org/index.php?title=6_Thinking_Hats_Retrospective. For the hats that are labeled 10 minutes, change them to 5 minutes as well. For the time period, instead of the last sprint, simply use the project-to-date. Have a scribe document the outcome for each step or "hat". Then write down your conclusions and outcomes as described at the bottom.
2. There are LOTS of websites out there on how to conduct an Agile sprint or project retrospective. Consider this page: https://blog.lucidmeetings.com/blog/how-to-lead-a-successful-project-retrospective-meeting. Read through the full description, including the "Step-by-Step" section. Write a persuasive argument for or against retrospectives as defined here as adhering to the Process measurement, analysis, and change stages discussed in your notes. Specifically, attempt to map what the retrospective does to each of these areas (measurement, analysis, change), and then critically review whether you think this form of review is a valid software process improvement technique given this framework. I expect at least a one page writeup. Refer to the general grading rubrics on writing above!

Activity 4 (5 points, INDIVIDUAL):
In your lab on defect estimation, you had Individual Thought Questions 2 and 3 related to PestSoftware Inc. from Activity 2 of the same lab. Revisit your answer to 2 and 3 and rewrite it in terms of the GQM paradigm on slides 8 and 9 of your notes.

Activity 5 Reflection (10 points, INDIVIDUAL):
Discuss your perspective on software process improvement. For many students this is a very difficult thing to do as you may not have a long track record of work experience and/or worked on projects at scale. Drawing from any personal experience you have (work, school, outside club, any activity you have been involved in for any length of time), discuss what kind of process improvements you would have like to have seen, and whether you would recommend a GQM-oriented or Agile (human-oriented) process improvement approach. Then map this back to your opinion on software process improvement. At least 500 words.

LAB 4 Description

SER416 Software Enterprise IV **Spring 2019**
Risk Management **due March 12 2019**

Objectives:

- Exercise the steps of the risk management process
- Understand how to use risk analysis techniques to prioritize requirements.
- Gain an awareness of individual personality types and how these play a role in teamwork.

Activity 1: (15 points)
In this activity you will prioritize alternatives to delivering project requirements using the general decision tree approach presented in Boehm's paper. Below you are provided data on two hypothetical requirements, "Req1" and "Req2". It does not matter what these requirements actually are. The project management team has identified a risk associated with each requirement (Table 1), and also identified risk mitigation and contingency planning possibilities with respect to the risks and requirements (Table 2). Using this data, you are to identify the sequence of activities your project should undertake. Specifically, apply Boehm's model to Req1 and determine whether it is worthwhile to build a Prototype or not, and to Req2 to determine whether Training Courses are worthwhile.

	Risk	*Req1*	*Req2*
1	Inadequate understanding of user needs	0.5 / 15	
2	Poor understanding of needed technology		0.3 / 20

Table 1. P(UO) / L(UO) per risk and requirement. As an example, read the entry in row 1, Req1 as "there is a 50% chance that Risk 1 will cause a loss of 15 business value points"

	Risk Mitigation – Risk Reduction/Cost	**Risk1**	**Risk2**
1	Prototype	0.4 / 5	
2	Training courses		0.1 / 5

Table 2. Risk Mitigation actions with risk reduction and cost values.

The 1st value in each cell is the probability by which the risk is reduced, and the 2nd value is the cost of implementing the risk mitigation strategy. For example, prototyping (Mitigation Plan 1) may reduce Risk1 for Req1 from 0.5 to 0.1 (0.5 – 0.4 = 0.1), but will come at a cost of 5 story points.

1. For each requirement, create a tree formulating the requirement's RE using the data in Table 1 and whether it is worthwhile to apply the mitigation plan in Table 2. State your conclusion for each Risk/Req.
2. Now add a new wrinkle, add a row (risk) to table 1 and a column to table 2 as such:

	Risk	*Req1*	*Req2*
1	Inadequate understanding of user needs	0.5 / 15	
2	Poor understanding of needed technology		0.3 / 20
3	Decaying architecture	0.2 / 30	0.1 / 40

Table 3. P(UO) / L(UO) per risk and requirement with new row 3.

Now we have a risk that applies to both requirements. Here is how our mitigation plans help with this new risk:

	Risk Mitigation – Risk Reduction/Cost	**Risk1**	**Risk2**	**Risk3**
1	Prototype	0.4 / 5		0.1 / 3
2	Training courses		0.2 / 5	0.1 / 2

Table 4. Risk Mitigation actions with risk reduction and cost values, risk 3 added.

Now we have a situation where a single risk impacts more than one requirement, and each mitigation strategy impacts more than one risk. Revisit your RE and decision trees for each of the mitigation plans to see if your answers change.

Activity 2: (15 points)
Activity 2 expects you to use a risk matrix type approach to determine which Mitigation and Contingency plans are worth applying to different Risk Categories in a hypothetical organization. It is expected you will use the matrix on slide 9, and you may also choose to construct your own matrix so you may decorate it with counts and costs of dealing with the Risk Events of each Risk Category.

Risk Category /Event	Severity	Probability	Mitigation Action	Contingency Action	Outcome of Actions
PERSONNEL					
Lack of knowledge in this hw/sw	1	A	1, 2		Probability is reduced 3 levels
Insufficient resources available	3	C	2	7	Probability reduced 1 level
EQUIPMENT					
Delivery date slip	2	D		7, 8	Severity reduced to Negligible (4)
Insufficient configuration	3	B	5, 6	3, 4	Probability reduced 2 levels, Severity reduced 1 level
LOGISTICS					
Multiple customer sites	4	A	9, 11		Probability is reduced 1 level
Physical separation of team & customer	2	E	10, 11		Probability is now zero

Mitigation / Contingency Plan Actions (Preventative Measures and Contingency Measures columns). Relative costs of each action are given as HIGH, MEDIUM, or LOW

1.	Provide appropriate training.	HIGH
2.	Hire trained specialists.	HIGH
3.	Install temporary hardware.	MEDIUM
4.	Utilize internal hardware temporarily.	LOW
5.	Purchase additional equipment.	HIGH
6.	Implement product functionality in a phased manner.	LOW
7.	Adjust deadline and get customer buy-off.	MEDIUM
8.	Increase estimates for the related tasks	HIGH
9.	Hold regular meetings with customer.	LOW
10.	Visit remote sites as needed.	MEDIUM
11.	Demonstrate incremental results.	MEDIUM

The Severity and Probability columns are coded to match the numbers and letters in the Risk Assessment Matrix show on slide 9 of your notes.

Answer the following 3 questions based on this table:

Q1: Using the leftmost 3 columns only, indicate which Risk Category (Personnel, Equipment, or Logistics) presents the most serious problem for the organization and why? Use the risk matrix on slide 9 to help you explain your answer.

Q2: Now add to the consideration of Q1 the rightmost column (Outcome of Actions) in the table. Do not consider the Mitigation or Contingency columns yet. Applying the outcomes of the actions, reconsider your answer to Q1. That is, presuming the application of the Mitigation and/or Contingency plans, the Severity and/or Probability is impacted in different way. Consider how the risk matrix changes with these new values.

Q3: Now factor in the actual Mitigation and Contingency plans. Specifically, the cost of implementing these plans is given in the list with each action. Considering how often an action appears in the table, how much it costs, and the Outcome of Actions (Q2), indicate a) which will be the most expensive category to apply actions to, and b) whether it is worthwhile to apply the actions given the cost information.

Activity 3: Individual Task (10 points)

Individually take the DISC personality test. You should then map the DISC type to one of the 3 personality types under Motivation on slide 13 of your PM overview notes. Optionally (only if you are comfortable), I'd like you to work with a partner (even out of class) and have them take the DISC test pretending that person is you, and see what result that person gets for you. Finally, and again only if you are comfortable, I'd like you to work with your capstone team to compose all DISC personality types and see if they are "compatible". Please submit a summary of your personality results and of the optional discussions in a brief (min 300 words) well-written summary that adheres to the general grading rubric.

Activity 4: Individual Reflections (10 points)

1. Describe (400 words) how your Capstone (or 316 if you are not in Capstone) team has gone through the “formin’, stormin’, normin’, and performin” stages of the Tuckman model that is popular in Agile team practices (ignore “adjourning”). Provide a timeline and the events that exemplify each stage (and I know it may not be the case that you have gone through all 4 stages yet). Please then share with at least one of your Capstone team members and discuss similarities and differences in your summaries. Jointly write a 200 word comparison (in can go in each of your individual submissions, but identify the team member).
2. Provide a paragraph min (300 words) that describes how your Capstone team deals with risk, even if you do not have a formal risk management process. First, is there any discussion of risk in the project management deliverables you have been asked to produce? Second if yes, do you follow these procedures, and if so, are they helpful? If you do not have them defined or you do not follow them, then describe how you *informally* deal with risk – risk is a reality whether you have a formal plan or not, and it comes in many forms – late on deadlines, team members not participating, lack of stability in your incremental sprint releases. How does your team presently deal with these and other risks you can identify?

****** All written deliverables should adhere to the general grading rubrics presented on previous labs for proper writing and critical/persuasive style! *** → C4***

LAB 1 Key provided to students as feedback

SER416 Software Enterprise IV: Process and Project Management **Spring 2019**
Lab: Quality Management and Defect Tracking

Instructions:
Activities 1 and 2 must be done in a team of 3. Activities 3 and 4 I expect you to do individually. Each student should make an individual submission by the due date (Tuesday February 5th at 11:59pm) with their copy of all Activities. Please indicate at the top your team partners. Submission done electronically to Canvas with a file type of .docx, .pptx, .pdf, or OpenOffice (.odt, .odp, .ott, .otp, .odg).

General Grading Rubrics:
1. I expect written answers to demonstrate proper use of spelling, grammar, punctuation, and active voice.
2. I expect the writing style to be persuasive and evidence-based. It should make a clear coherent argument, report findings based on evidence, and/or
3. Provide proper attribution in all places you refer to an outside reference. Specifically, include a reference list and cite in IEEE style (see https://en.wikipedia.org/wiki/IEEE_style).
4. All students are responsible for a team's shared answers. If the answer is plagiarized all students on the team will receive the AIV penalty. No grade appeals are allowed based on "I didn't do that Activity".
5. Individuals are responsible for Individual Activities. Sharing of answers of these Activities is an AIV.

As quality of writing will be taken into account, I strongly suggest the team work in iterations – at least one iteration to get the evidence down for each shared Activity, one to discuss the questions requiring discussion (taking a position on something), one to draft the polished language, and one to polish the presentation (final review). Keep in mind as well that sometimes Individual Activities rely on team Activities so do not wait until Tuesday night!

Activity 1: Quality Management Standards (12 points)
I posted the IEEE-730 standard on the class website. Review this document, looking for cross-references to other IEEE standards, or other standards in general. For each major phase of the software lifecycle, find two places in IEEE-730 that pertain to the phase, and find cross-references to other standards and a description of the QA activities pertaining to the phase. You may find it useful to Google some of the IEEE standards for summaries, or refer to their descriptions on www.ieee.org. Specifically fill in the following table:

Software Process Lifecycle Phase	IEEE-730 section & page reference	Other (IEEE) standard references	Description of other activities performed in this phase as discussed in IEEE-730
Requirements			•
Analysis & Design			
Implementation			
Deployment/ Operation			
Maintenance			
Cross-cutting			

"Cross-cutting" here refers to QA activities described in IEEE-730 that apply to more than one, and possibly all, software lifecycle phases. It is my hope that mapping these activities will give you some guidance on mapping your project deliverables to IEEE-730 areas, and therefore give you a head start on doing your QA plan.

I do not have a specific set of entries for the table as there are multiple possibilities. They are to provide 12 entries (2 per phase) for 1 pt each. You may score with ½ pt increments. Please check the section and page number for the reference. Assuming the reference is there, check for references to any other standards (not just IEEE), and if they have a description that is at least mentioned in that part of the IEEE-730 spec (which is on Canvas).

Activity 2: Defect Tracking (22 points)

A. (1) Explain why a software defect is called a *bug*. *They should tell the story of a bug getting caught in a vacuum tube (1/2 pt). They should use complete sentences and provide a reference (1/2 pt)*
B. (2) In class we discussed *Failures – Faults – Errors (Defects)*. The SWEBOK version 3 identifies 5 categories, not 3. What are they? How do they overlap with these 3?

   The SWEBOK in section 3.2 of chapter 10 describes Computational Error, Error, Defect, Fault, and Failure. While 3 of the terms are in the notes, only Failure really has the same definition. Error is slightly modified to mean "human action", while Fault is a bug, which is a completely different definition than the notes. Computational Error and Defect together mean what the notes definition of what Error means.

In terms of grading, ½ pt for identifying the 2 new terms, ½ pt for identifying that Failure is basically the same, ½ pt for identifying that Fault means something completely different (what notes meant by Error), and ½ pt for noting Error is restricted to human action. You may get some answers that you have to read carefully as they may not say it the same way I am, so take the time to understand. Be lenient in your interpretation; if it loosely connected to my criteria give them the credit but provide written feedback.

C. (9) This blog post: https://blog.capterra.com/top-free-bug-tracking-software/ summarizes 10 open source defect tracking tools. Pick any 3 and i) provide a key feature summary, ii) define the default roles available in the system (and what features they support), and iii) create a graph of the default defect-handling states and transitions (like your notes). For (i) and (ii) I am fine if you provide a table, though make sure you define what the key features are, not just label them.
3 defect tracking tools are required, and 3 criteria per tool = 9 points (1 pt each). I am looking more for completeness than for you to go check each tool. In fact what I would do is sort by the tools selected and then compare answers between teams for consistency. The teams that gave answers for a tool much different than others will be the ones you may need to go to the tool's documentation to check. Also I am allowing them to use a cut-and-pasted graph for (iii), but make sure what they give you is a state diagram, not some other kind of diagram. Some may not even know what they were looking for!

D. (4) GitHub includes an Issues feature where one can create an issue and label it a *bug*. GitHub and the larger GitHub community also post a number of workflows to handle various development patterns. Find 2 GitHub development workflows related to bug issue handling – describe them and provide a state diagram for each.
There are a LOT of GitHub workflows out there, so I can't give a specific criteria. You have to check that they identified 2 workflows (1/2 pt each), provided references for each workflow found (1/2 pt each), describe them and how they could tie in some way to issue handling (1/2 pt each), and provide a state or flow diagram (I allowed flow-style diagrams like an Activity diagram) for each (1/2 pt each)

E. (6) Agile methods such as Scrum and XP favor practices such as Continuous Integration and Testing through automated unit tests and automated static analysis. Answer the following:
   a. Should all defects found in CI&Test be entered in a defect tracker? Why or why not? Explain. If your answer is "only some should" then provide a threshold definition.
   b. Agile methods, XP in particular, also promotes Pair Programming and informal code reviews. Should all defects found in Pair Programming / informal code reviews be entered in a defect tracker? Why or why not? Explain. If your answer is "only some should" then provide a threshold definition.
   c. In SER216 you used a "lite" version of the PSP by Watts Humphrey. Humphrey believed in a broad definition of a *defect* (see your notes) and in logging all defects, even those found by code review *before the code is even compiled*. This is counter to the Agile mindset, however provide a justification as to why this may be a good idea. It may help to review Humphrey's motivations in this technical report: https://resources.sei.cmu.edu/asset_files/TechnicalReport/2000_005_001_13751.pdf

Question E is really free-form writing. Please score 3 of the 6 points based on the General Grading Criteria 1-3. Specifically:
- *2 spelling mistakes = ½ pt, 1 point max for 4 or more*
- *2 grammar mistakes = ½ pt, 1 point max for 4 or more*
- *2 missing references (where it is clear they took from somewhere but did not cite) ½ point, 1 point max for 4 or more*
- *If you find the answer is plagiarized in any way, deduct all 6 points. You can usually detect this by seeing wording that reads out of place (it doesn't sound like what a student would write). If it is quoted and not cited, ½ pt. If it is cited but not quoted (which they need to do if anything is copied verbatim), ½ pt. If they do not use quotations AND do not cite, it is plagiarism.*
- *The other 3 points are 1 point each for a, b, c. In a and b, we are looking for persuasive writing, meaning they should state an answer and then defend with some line of justifiable reasoning (and hopefully evidence of some kind). We'll accept whatever answer and reasoning (as long as it is not nonsense) if it is presented persuasively. Part c) needs to be graded like this as well, but they will probably give you Humphrey's reasoning which is basically you cannot estimate what you do not measure first, so if you do not log everything you cannot derive good future estimates.*

Activity 3 (6 points, INDIVIDUAL):

A. In Activity 2 C and D you should have looked at 4 tools. Pick the best and worst tools from amongst these 4 and explain what specific features of the tool lead you to these conclusions.
B. You also should have a total of 5 state diagrams from Activity 2 C and D, and you have 2 in your notes. Pick the best and worst state models from these 7 models and explain why you chose the models you did.
C. Draw a state model of your own that takes the best from these models. Explain how you derived this model (what ideas you got from what other models).

Another open-ended question, 2 points each part. A and B are 1 point for the "Pick" and 1 point for the explanation. As in the previous question the explanation must be properly written (spelling and grammar ½ pt) and use a persuasive style (1/2 pt). For part C, 1 point for having a valid state model (they may give you a non-state diagram), and 1 point for an explanation with the same persuasive style rubric.

Activity 4 Reflection (10 points, INDIVIDUAL):

A. Describe what defect tracking tool or process your Capstone team is using if you are in the Capstone course. If your team does not yet have one, indicate what you will recommend to them and why. List your Capstone team members. If you are not on a Capstone team, then use your SER316 team.
B. Discuss (at least 200 words) your perspective on defect tracking. Is it an important quality process? Is it more or less important than other quality activities (testing, code reviews, static analysis, etc.)? What do you think the pros and cons are, and how would you determine the best defect tracking process for a given project? That is, what project attributes will influence what type of defect tracking you think that project should do?

For part A (4 pts), only one person is in SER316 (Mitchell Roberts) so I believe the rest should have a capstone reference in the response. 3 points for either describing an existing process or indicating what you will recommend and why. Again, open-ended, accept anything that is coherent and sounds reasonably justified. 1 additional point for proper spelling and grammar.

For part B (6 pts):

- *½ pt deduction for < 180 words (I'll accept 180-199 as OK – note Word has a count tools built in if you highlight the answer).*
- *Additional ½ point for < 140, and another ½ for < 100, and another ½ pt for < 60 (2 pts max)*
- *1 pt for indicating pros and cons, ½ point deduction for skipping pros, ½ point deduction for skipping cons*
- *1 point for answering "Is it more or less important than other quality activities (testing, code reviews, static analysis, etc.)?" – again they can form an opinion but they have to answer the question somewhere.*
- *1 point for "how would you determine…That is, what project attributes…?". Again I know this is very subjective, and I do expect you to be lenient, but basically all I am asking are what are the distinct descriptors of a project that would influence their decision? It could be anything from what operating system they are on to the experience of developers to whether the team is distributed – almost anything SE-related that could impact a project team I will accept as long as the explanation is coherent. You may grade at ½ pt intervals subjectively.*
- *The final point is for proper spelling and grammar (and citations if relevant). Use the rubric above for 2E.*

Lab 2 Key provided to students as feedback

SER416 Software Enterprise IV: Process and Project Management **Spring 2019**
Lab: Defect Estimation **due February 12, 2019**

Objectives:

1. Understand how to estimate the total number of defects for a project using the Capture-Recapture method.
2. Know the importance of defect containment by understanding the play between defect injection and removal rates.

You may do this lab with 1 other partner who is NOT on your project team.

Activity 1: Capture-Recapture Method (9 points)

Part 1 (4): Compute a defect estimate ala the Schofield paper using the defect data in the table below:

	Huey	Dewey	Louie	A	B	C
Defect 1	X	X		X	X	X
Defect 2		X		X		
Defect 3			X		X	
Defect 4		X		X		
Defect 5	X	X	X	X	X	X
Defect 6		X		X		
Defect 7		X		X		
Defect 8	X				X	
Defect 9		X	X	X	X	X

|A| = 7, |B| = 5, |C| = 3, so 7/X = 3/5, 3X = 35, or X = 11.67 (round up to 12).
So, the team found 9, estimate there are 12, for a defect discovery rate of 75%
1 point for "9", 1 point for "75%" defect discovery rate, 2 points for the added columns A/B/C. Deduct ½ pt for each X missing or in the wrong place up to max of 2 points

Add the needed columns to the table to show your work.
Answer: What is the defect discovery rate for the Huey/Dewey/Louie team? Explain.

Part 2 (6): Compute a defect estimate again, this time from reviews from 4 developers:

	Paul	John	George	Ringo	A	B	C
Memory Leak			X	X	X	X	X
DB connection not closed	X		X		X	X	X
Does not close file		X				X	
Incorrect parameter type	X		X		X	X	X
Improper indentation	X	X	X	X	X	X	X
Lack of code comments			X		X		
Null pointer Exception	X	X	X		X	X	X
Type mismatch error				X		X	
Incorrect logic condition		X	X	X	X	X	X
Incorrect computation			X		X		
Improper use of synch block		X				X	
Uses goto	X	X				X	
Incorrect initialization of var			X	X	X	X	X
Improper access level	X			X		X	

|A| = 9, |B| = 12, |C| = 7. 9/X = 7/12, 7x = 108, X = 15.429, so round to 15. The team found 14 of 15 possible defects for a defect discovery rate of 93.3%.
1 point for 15, 1 point for defect discovery rate of 93%, 3 points for columns in table. Deduct ½ pt for each X missing or in the wrong place up to max of 3 points. Note if they created 4 columns they did it wrong!

Add the needed columns to the table to show your work.
Answer: What is the defect discovery rate for the Beatles team? Explain.

Activity 2: Defect Containment Derivation (13 points)

Consider the following information about a hypothetical application for PestSoftware Inc.:

- The application has 180,000 lines of code (LOC)
- PestSoftware uses RUP, so it has phases (I)nception, (E)laboration, (C)onstruction, and (T)ransition.
- There is a defect injected for every 15 LOC.
- The *defect injection distribution rates* per phase are: I = 10%, E = 25%, C = 45%, and T = 20%
- The *defect removal rates* per phase are: I = 60%, E = 80%, C = 67%, and T = 50%
- The *cost ratios* of fixing defects are: I:E:C:T:F → 1:5:10:50:100, where F means the defect has made it to "(F)ield operation". Assume a defect always costs 1 "unit" to fix if found in phase.

Use this information to answers the following questions.

1. (1) How many total defects exist in the application? *2216 1 pt*
2. (2) How many defects exist per phase? *2 pts. I = 16, E = 100, C = 900, T = 1200. ½ point off for each incorrect.*
3. (5) Fill in the matrix below. In this matrix, each entry has 2 cells. Fill in the left one with the number of defects from the column phase, and the right one with the number of defects removed in this phase. *5 pts. There are 10 cells to check really, highlight below. ½ point for each correct. Look for a pattern if they make a mistake. Often students will incorrectly apply a ratio or not compute the next row starting point correctly (for example, not understanding Elaboration starts with 480 defects as that is 1200-720)If it is a single mistake repeated, only deduct 1 pt total.*

	Inception		**Elaboration**		**Construction**		**Transition**		**Total**	
Injected/Removed	Inj	Rem	Inj	Rem	Inj	Rem	Inj	Rem	Inj	Rem
Inception	1200	720							1200	720
Elaboration	480	384	3000	2400					3480	2784
Construction	96	64	600	400	5400	3600			6096	4064
Transition	32	16	200	100	1800	900	2400	1200	4432	2216
Field operation	16		100		900		1200		2216	

As an example: Suppose there are 100 defects in a system. The defect injection rate says that 10%, or 10 of them, were injected during Inception. Then the left cell of the 1st row/column (Inception/Inception) would be 10. The right cell would be the defects removed *in-phase*, which would be 6 since the defect removal rate for Inception is 60%. That would leave 4 defects carried over to Elaboration, entered in row Elaboration and column Inception – and so on.

4. (2) What are the total costs per phase for injected defects? (i.e. how much did all of the defects for Inception cost you? Elaboration? etc. Keep in mind that a defect fixed in phase costs 1 unit, and ratio from there).
5. (2) What is the cost-per-defect for each phase?

Inception = 720 + 384*5 + 64*10 + 16*50 + 16*100 = **5680**, or 4.73 per defect
Elaboration = 2.4k + 400*2 + 100*10 + 100*20 = **6200**, or 2.07 per defect
Construction = 3.6k + 900*5 + 900*10 = **17,100**, or 3.17 per defect
Transition = 1.2k + 1.2k*2 = **3600** = 1.5 per defect

*Note that the ratios between phases are used. So for example, when doing Elaboration, 2400 is the base cost, so the next up (400 in Construction) is * 2 because the ratio is 5:10. Note that many folks may have misinterpreted this specific part of the costs. If they applied the 1:5:10:50:100 straight down the table (one per row) accept it anyway.* ***Bold*** *is #4 (2 pts, ½ per each), decimal value is per defect #5, also ½ each for 2 points.*

6. What is the overall project defect removal rate? *1 point, should be expressed as a percentage.*

Overall: 12,000 defects injected, 9784 removed (720+2784+4064+2216), 71. 1%

Activity 3: Defect Depletion Activity (12 points)

Some time ago a software engineer named Ed Weller put together a simple Defect Deflection worksheet in Excel. Please download this sheet. Defect depletion is somewhat like your previous problem, except it examines your specific defect removal processes and costs between *inspection* activities in requirements, analysis, and design (high-level and low-level) and *test* activities (unit, integration, and system testing). This model does assume a waterfall-like development process (so when you read the formulas going top-down and left-right, you can see how they are derived from previous activities). Weller's model is not based on any established standard or study, just a personal model he posted to a software engineering quality site. But spreadsheets like these are not uncommon (we had a capstone project with General Dynamics once where the team had to replace over 15K of Excel macros!).

The spreadsheet has a tab already filled in for Aardvark.org. Create 2 copies of the model on 2 separate tabs (name each tab for the organization name below. Enter the following data on each yellow highlighted line:

BewareBugs Inc:

Size in KLOC: 1	Req	Ana	HLD	LLD	Code	Unit Test	Integ Test	Sys Test
Injected	2.5	3.75	5	15	20	0.5	0.25	0.1
Removal Rate	0.8	0.7	0.5	0.4	0.7	0.3	0.7	0.7
Cost	5	3	3	2	1	1	16	40

Cantankerous Coders, LLC:

Size in KLOC: 1	Req	Ana	HLD	LLD	Code	Unit Test	Integ Test	Sys Test
Injected	1.5	2.5	10	45	50	1	0.5	0.5
Removal Rate	0.5	0.5	0.8	0.7	0.9	0.8	0.4	0.4
Cost	2	1	5	3	3	2	12	32

1 point for each tab (2 pts total) in the submitted spreadsheet for BewareBugs and Cantankerous Coders Really they just had to transcribe this table data into copies of the sheet I gave them.

Answer the following: *(2 points each) I wasn't clear enough, they could have answered this 2 ways – which Development Phases, or which defect process – Inspections or Testing. So accept either.*

1. What is Aardvark.org good at? What are they bad at? *Aardvark is good at Requirements, Analysis, and Unit Testing. These all have low costs. They are poor at LLD and System Test, and to a certain extent HLD and Code. In general good at Testing and bad at Inspections.*
2. What is BewareBugs good at? What are they bad at? *BB is good at unit test and to some degree Analysis and HLD. They are very bad at Integration and System Test. In general good at Inspections and bad at Testing.*
3. What is CantankerousCoders good at? What are they bad at? *CC is good at requirements, analysis, and all forms of testing. They are bad at Code and LLD. They are poor at Inspections, much stronger in Testing.*
4. What is the most costly quality problem for Aardvark.org? For BewareBugs? For Cantankerous Coders? Explain. *For Aardvark it is Code (71) though one could argue these values are not normalized against the amount of work invested in each phase. Considering that and the objectives of the phase, System Test look pretty poor. For BB, Integration Test stands out as most costly. For CC, Coding stands out and so does LLD, but also relatively speaking HLD is much higher earlier in the process that the other 2 organizations*.
5. Identify ONE thing you think each organization (including Aardvark.org) needs to do to improve? *This is really just the other side of the coin for question 4. They should name a practice that impacts the phase they said was the most costly.*

Activity 4: INDIVIDUAL Thought Questions (8 points)

As these are individual thought questions, I will award a point on each for any well-phrased presentation (according to the general writing rubrics). The other point is for whether that presentation has merit – particularly for 2-4 where their answer doesn't have to agree with my answer, but it does have to be evidence-based.

1. (2) Suppose in Part 1 of CRM, that you found out that a) Huey seems to only find coding style errors while Louie only finds memory-related defects, and b) Dewey was kind of tired so he only skimmed the last 25% of the code. Would this information shake your faith that the CRM-based estimate you derived was appropriate? Explain why or why not for both (a) and (b).

These changes do affect the CRM process. Recall that CRM is based on a few assumptions, such as the "marked" population being evenly distributed throughout the entire population. Here this means that the defects you are discovering are "representative" of the entire population of defects. But, these assumptions skew what defects are discovered, and so may skew the results.

2. (2) If you asked to pick only one phase of the software process for PestControl Inc. to improve, what phase would it be, and what activity would you focus on (injection, discovery, or removal)? Explain your answer to both.

There could be a variety of correct answers for this, I am looking for how you justified the answer. You could argue that a lower defect injection rate for a phase is best, or a better defect discovery rate, or lowering the cost-per-defect ratio. To support your argument, you should "run some numbers". While one could get elaborate with a sim model or a spreadsheet calculation to show what is best, I am merely expecting you to try a few alternatives and weight the results.

3. (2) Is there any additional information that is missing so far that would have helped you answer #2 better?

The main information that is missing is the cost of improving each of the above activities (injection, discovery, removal). You only know if the improvement is worth it if you can amortize the cost of the process improvement w.r.t. your data. At some point, extra investment may reach a point of diminishing returns.

4. (2) What are the challenges in applying *defect containment* models in Agile software development processes? Is there a way to adapt them to this purpose? I'd like you to answer this question by yourself first, just thinking about it. Then, using Google as your friend, see what the community says.

This is a persuasive argument question. For the 1st question, there are several challenges, most notably that phases are gone. Yet one could argue that you could map user story development, construction tasks, or unit tests to different phases. So for 2 points we'll accept either position, but they must make a coherent argument.

<u>Activity 5: Individual Reflection (8 points)</u>

This activity is to be completed individually by each student separate from their team! Answer the following.

Reflection questions:

1. Do you think *CRM* is a useful technique to do on real software projects?
2. Do you think *CRM* will benefit your capstone (316) project? How will it, how will it not? Do you think a different defect estimation method will work better?
3. In general, what do you think the main benefits and challenges are in *defect estimation*?
4. In general, what do you think the main benefits and challenges are in *defect containment*?

2 points per question, 1 point on general writing rubrics. The other point really for just answering the question in some coherent way. If you are not sure ask.

Lab 3 Key provided to students as feedback

SER416 Software Enterprise IV: Process and Project Management **Spring 2019**
Lab: Software Process Improvement

Instructions: ***Note: Please do Activity 3.1 in class on February 13th!***
Activities 1-3 must be done with your Capstone team. Activities 4 and 5 I expect you to do individually. Each student should make an individual submission by the due date (Tuesday February 19th at 11:59pm) with their copy of all Activities. Please indicate at the top your team partners. Submission done electronically to Canvas with a file type of .docx, .pptx, .pdf, or OpenOffice.

General Grading Rubrics:

1. I expect written answers to demonstrate proper use of spelling, grammar, punctuation, and active voice.
2. I expect the writing style to be persuasive and evidence-based. It should make a clear coherent argument, report findings based on evidence, and/or
3. Provide proper attribution in all places you refer to an outside reference. Specifically, include a reference list and cite in IEEE style (see https://en.wikipedia.org/wiki/IEEE_style).
4. All students are responsible for a team's shared answers. If the answer is plagiarized all students on the team will receive the AIV penalty. No grade appeals are allowed based on "I didn't do that Activity".
5. Individuals are responsible for Individual Activities. Sharing of answers of these Activities is an AIV.

As quality of writing will be taken into account, I strongly suggest the team work in iterations – at least one iteration to get the evidence down for each shared Activity, one to discuss the questions requiring discussion (taking a position on something), one to draft the polished language, and one to polish the presentation (final review). Keep in mind as well that sometimes Individual Activities rely on team Activities so do not wait until Tuesday night!

Activity 1: Software Process Improvement Standards (11 points)
On Canvas you will find a copy of the IDEAL software process improvement model.

1. On slide 4 of your notes, it talks about process measurement, analysis and change. Identify sections of the IDEAL standard that discuss each of these aspects and summarize what it says about them. Include references to specific sections and page numbers.
2. On slide 8 of your notes, it describes, at a high level, 5 stages to conducting a software process improvement change. To what extent do these stages appear in IDEAL (Note: Don't expect it to be as simple as finding a section named for one of the stages, like "Process change introduction" – you will have to read and understand to map a section to a stage).

I expect these discussions to be significant and show some depth. If it helps, I expect at least 300 words per each part 1 & 2 or at least 600 words total. Refer to the general grading rubrics on writing above!

300 words Q1 & Q2 1 pt each (2), 2 pt deduction if < 200. Proper spelling punctuation and grammar 3 pts – deduct ½ point per each 3 such errors you find with Word's checker tool up to a max of 18 errors (3 pts).

For Q1 (3 pts), there are many places in the IDEAL standard that talk about process measurement, analysis, and change – however a simple keyword search is not the way to reveal them. Instead, just look at the Table of Contents – Chapter 2 is mostly Measurement, Chapter 3 mostly Analysis, Chapter 4 mostly Change – these are not hard and fast mappings, but I would expect to see this pattern in student responses. I cannot provide a full mapping of all the places these appear as the standard is too long, so instead look for this pattern and selectively check some answers by going to the document.

For Q2, the 5 stages are

1. *Improvement identification*
2. *Improvement prioritization*
3. *Process change introduction*
4. *Process change training*
5. *Change tuning*

1 pt for each stage. If they can make a reasonable connection give them the point for the stage. Basically we are asking them to map much like I did for Q1 – the fastest way to do it would simply go to the ToC. Chapter 2 is mostly stage 1, as is first ½ of chapter 3. Chapter 3 2nd half gets into stage 2. Stages 3 and 4 are in Chapter 4, and Stage 5 (loosely interpreted) could be found in Chapters 5 and 6. That is a mapping pattern I would look for, but again you will have to read each stage answer to see if a plausible connection is there as it is not possible to write out all the possible connections.

Activity 2: Software Process Improvement Examples (12 points)
On Canvas you will find two reports, one by Raytheon and one by Nationwide, written as recipients of the 2016 Watts S. Humphrey Software Process Achievement Awards. For each report, select any three of the process attributes in Figure 26.2 of Chapter 26 of Sommerville (also available on Canvas) and discuss how that organization provided a high level of that process attribute. You do not have to use the same process attributes for each organization. I expect at least 125 words per process attribute per organization, or at least 750 words total. Refer to the general grading rubrics on writing above!

Again, open-ended grading, and it is not possible for me to enumerate all the attributes and all the places possibly discussed in the 2 reports. So grading this is more about checking what they claim and seeing if it is plausible.

(3) 125 words per process attribute – ½ point deduction for each that fails to get 100 words (we will give some slack room)
(3) Again check for spelling, grammar, and punctuation, deduct ½ point for every 3 defects up to a max of 3 points.
(6) 3 process attributes x 2 reports = 6 points, 1 per process attribute. Read the explanation and verify plausibility of the connection. The bar remains low – "plausibility", which again means they wrote an argument for a mapping of an attribute that one can see a possible connection based on the cited areas of the report, even if we would not agree with it. Deductions are really meant for places where it was clear the team also makes a cursory connection by keyword without any effort.

Activity 3: Software Process Improvement and Agile methods (12 points)

1. **DO THIS ONE IN CLASS!** Organize yourself in your capstone teams. Execute the "6 hats retrospective" as described here: http://retrospectivewiki.org/index.php?title=6_Thinking_Hats_Retrospective. For the hats that are labeled 10 minutes, change them to 5 minutes as well. For the time period, instead of the last sprint, simply use the project-to-date. Have a scribe document the outcome for each step or "hat". Then write down your conclusions and outcomes as described at the bottom.
2. There are LOTS of websites out there on how to conduct an Agile sprint or project retrospective. Consider this page: https://blog.lucidmeetings.com/blog/how-to-lead-a-successful-project-retrospective-meeting. Read through the full description, including the "Step-by-Step" section. Write a persuasive argument for or against retrospectives as defined here as adhering to the Process measurement, analysis, and change stages discussed in your notes. Specifically, attempt to map what the retrospective does to each of these areas (measurement, analysis, change), and then critically review whether you think this form of review is a valid software process improvement technique given this framework. I expect at least a one page writeup. Refer to the general grading rubrics on writing above!

Q1 – 7 points. 1 point for capturing some for each "hat (6 hats), and 1 point for the conclusions and outcomes. Really every team that was there should get the 6 points as they were doing the steps, unless they poorly documented things. For conclusions and outcomes, they should have a reasonable summary of what they found out and what they will take back to their capstone team. I did not put a word limit on it, but I would expect more than 2 sentences. They should at least have a short paragraph tied back to the process.
Q2 – 5 points. 1 point for 1 page, 2 points for writing rubrics above (spelling grammar punctuation again but only up to 2 points off). 2 points for the answer. The key here is persuasive argument. This means that have to write in a style that tries to convince the reader (you). If they just summarize or do not take a position, deduct 1 point. I chose this blog post because it is the opposite of the SPI methods and processes like CMM, IDEAL, and what the textbook discusses – the process uses more human anecdotal input and an informal structure and treats the process more as a social exercise that as a tradition process improvement process (1 point).

Activity 4 (5 points, INDIVIDUAL):
In your lab on defect estimation, you had Individual Thought Questions 2 and 3 related to PestSoftware Inc. from Activity 2 of the same lab. Revisit your answer to 2 and 3 and rewrite it in terms of the GQM paradigm on slides 8 and 9 of your notes.

Check they went back to their lab 2, as they may not have worked with the same team. Rewriting in terms of the GQM paradigm means simply rewriting their existing answer in terms of Goals, Questions, and Measures. Check they follow this format (2 points), and it still reflective of their previous answer (2 points). The last point is ½ per each 3 spelling, grammar, punctuation errors as before.

Activity 5 Reflection (10 points, INDIVIDUAL):
Discuss your perspective on software process improvement. For many students this is a very difficult thing to do as you may not have a long track record of work experience and/or worked on projects at scale. Drawing from any personal experience you have (work, school, outside club, any activity you have been involved in for any length of time), discuss what kind of process improvements you would have like to have seen, and whether you would recommend a GQM-oriented or Agile (human-oriented) process improvement approach. Then map this back to your opinion on software process improvement. At least 500 words.

2 points 500 words, 1.5 points < 400, 1 point < 300, 0.5 points if < 200, 0 < 100.
1 point for spelling grammar, punctuation, 3 defects per ½ point.
2 points for sharing a personal process experience
2 points for suggesting process improvements
1 point for GQM or Agile
2 points for mapping back to the originally expressed opinion (perspective of sentence 1).

Lab 4 key provided to students as feedback

SER416 Software Enterprise IV — **Spring 2019**
Risk Management — **due March 12 2019**

Objectives:

- Exercise the steps of the risk management process
- Understand how to use risk analysis techniques to prioritize requirements.
- Gain an awareness of individual personality types and how these play a role in teamwork.

Activity 1: (15 points)
In this activity you will prioritize alternatives to delivering project requirements using the general decision tree approach presented in Boehm's paper. Below you are provided data on two hypothetical requirements, "Req1" and "Req2". It does not matter what these requirements actually are. The project management team has identified a risk associated with each requirement (Table 1), and also identified risk mitigation and contingency planning possibilities with respect to the risks and requirements (Table 2). Using this data, you are to identify the sequence of activities your project should undertake. Specifically, apply Boehm's model to Req1 and determine whether it is worthwhile to build a Prototype or not, and to Req2 to determine whether Training Courses are worthwhile.

	Risk	***Req1***	***Req2***
1	Inadequate understanding of user needs	0.5 / 15	
2	Poor understanding of needed technology		0.3 / 20

Table 1. P(UO) / L(UO) per risk and requirement. As an example, read the entry in row 1, Req1 as "there is a 50% chance that Risk 1 will cause a loss of 15 business value points"

	Risk Mitigation – Risk Reduction/Cost	**Risk1**	**Risk2**
1	Prototype	0.4 / 5	
2	Training courses		0.1 / 5

Table 2. Risk Mitigation actions with risk reduction and cost values.

The 1st value in each cell is the probability by which the risk is reduced, and the 2nd value is the cost of implementing the risk mitigation strategy. For example, prototyping (Mitigation Plan 1) may reduce Risk1 for Req1 from 0.5 to 0.1 (0.5 – 0.4 = 0.1), but will come at a cost of 5 story points.

1. For each requirement, create a tree formulating the requirement's RE using the data in Table 1 and whether it is worthwhile to apply the mitigation plan in Table 2. State your conclusion for each Risk/Req.

I will send a sketch of my 2 trees I did freehand. I also let them draw theirs freehand. There can be some variation in the tree – for example I do not show the 2nd branch on the 2nd level of the tree as it is the case the Loss value will be zero, thereby making the entire Risk Exposure (RE) value 0 for that branch, so I don't think it is worth including. But many students will and that is OK. Also, another variation is some may have written a tree per mitigation strategy (Table 2 above), as in "prototype or do not prototype" for tree 1, and "train or do not train" for the other. This should be fine – the math should work out exactly the same as they impact separate risks from table 1. Rubric:

a) *2 points for any attempt at each of the 2 tables (4 points total) even if incorrect*
b) *1 point per table for getting a correct (or near-correct) table, whether oriented by Req1/2 or Risk Mitigation (RM ½). Again as stated above these should be the same. 2 points total*
c) *½ point for each of the 4 RE values (2 from each tree). 2 points total*
d) *1 point each tree for indicating whether the RM1/2 should be used or not. (Req 1/RM1 yes, Req2 / RM2 no)*

2. Now add a new wrinkle, add a row (risk) to table 1 and a column to table 2 as such:

	Risk	***Req1***	***Req2***
1	Inadequate understanding of user needs	0.5 / 15	
2	Poor understanding of needed technology		0.3 / 20
3	Decaying architecture	0.2 / 30	0.1 / 40

Table 3. P(UO) / L(UO) per risk and requirement with new row 3.

Now we have a risk that applies to both requirements. Here is how our mitigation plans help with this new risk:

	Risk Mitigation – Risk Reduction/Cost	**Risk1**	**Risk2**	**Risk3**

1	Prototype	0.4 / 5		0.1 / 3
2	Training courses		0.2 / 5	0.1 / 2

Table 4. Risk Mitigation actions with risk reduction and cost values, risk 3 added.

Now we have a situation where a single risk impacts more than one requirement, and each mitigation strategy impacts more than one risk. Revisit your RE and decision trees for each of the mitigation plans to see if your answers change.

So there is actually a mistake in the 2nd part of this problem (there should not be 2 separate cost values in Table 4). Also the students struggled very badly with it. You will see them try multi-level tables, or tables with lots of branches as they work through the combinatorics. If they made a reasonable effort to solve it, give them the full 5 points. Only deduct points if the effort was not there.

Activity 2: (15 points)

Activity 2 expects you to use a risk matrix type approach to determine which Mitigation and Contingency plans are worth applying to different Risk Categories in a hypothetical organization. It is expected you will use the matrix on slide 9, and you may also choose to construct your own matrix so you may decorate it with counts and costs of dealing with the Risk Events of each Risk Category.

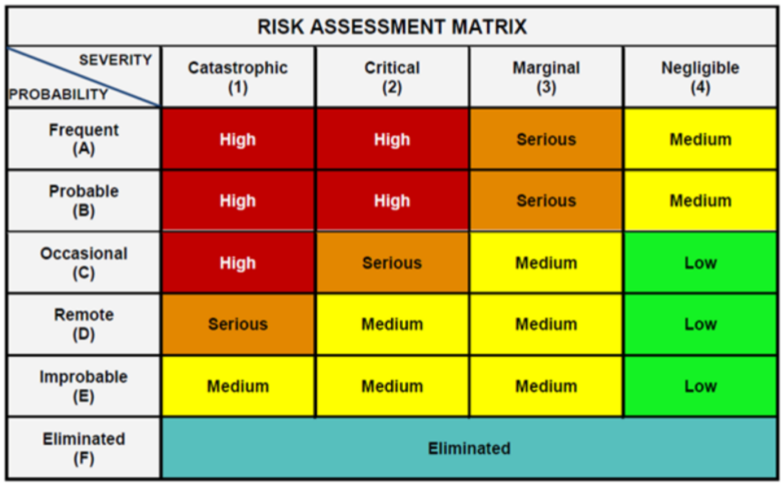
Risk Category /Event	**Severity**	**Probability**	**Mitigation Action**	**Contingency Action**	**Outcome of Actions**
PERSONNEL					
Lack of knowledge in this hw/sw	1	A	1, 2		Probability is reduced 3 levels
Insufficient resources available	3	C	2	7	Probability reduced 1 level
EQUIPMENT					
Delivery date slip	2	D		7, 8	Severity reduced to Negligible (4)
Insufficient configuration	3	B	5, 6	3, 4	Probability reduced 2 levels, Severity reduced 1 level
LOGISTICS					
Multiple customer sites	4	A	9, 11		Probability is reduced 1 level
Physical separation of team & customer	2	E	10, 11		Probability is now zero

Mitigation / Contingency Plan Actions (Preventative Measures and Contingency Measures columns). Relative costs of each action are given as HIGH, MEDIUM, or LOW

1. Provide appropriate training.	HIGH
2. Hire trained specialists.	HIGH
3. Install temporary hardware.	MEDIUM
4. Utilize internal hardware temporarily.	LOW
5. Purchase additional equipment.	HIGH
6. Implement product functionality in a phased manner.	LOW
7. Adjust deadline and get customer buy-off.	MEDIUM
8. Increase estimates for the related tasks	HIGH
9. Hold regular meetings with customer.	LOW
10. Visit remote sites as needed.	MEDIUM
11. Demonstrate incremental results.	MEDIUM

The Severity and Probability columns are coded to match the numbers and letters in the Risk Assessment Matrix show on slide 9 of your notes.

The Risk Matrix on slide 9 is:



RISK ASSESSMENT MATRIX				
SEVERITY / PROBABILITY	Catastrophic (1)	Critical (2)	Marginal (3)	Negligible (4)
Frequent (A)	High	High	Serious	Medium
Probable (B)	High	High	Serious	Medium
Occasional (C)	High	Serious	Medium	Low
Remote (D)	Serious	Medium	Medium	Low
Improbable (E)	Medium	Medium	Medium	Low
Eliminated (F)	Eliminated			

Answer the following 3 questions based on this table:

Q1: Using the leftmost 3 columns only, indicate which Risk Category (Personnel, Equipment, or Logistics) presents the most serious problem for the organization and why? Use the risk matrix on slide 9 to help you explain your answer.
Personnel = High+Medium, Equipment = Medium+Serious, Logistics = Medium+Medium, ***Personnel***
4 points. 1 for identifying Personnel and 3 for the reason (comparing these values above)

Q2: Now add to the consideration of Q1 the rightmost column (Outcome of Actions) in the table. Do not consider the Mitigation or Contingency columns yet. Applying the outcomes of the actions, reconsider your answer to Q1. That is, presuming the application of the Mitigation and/or Contingency plans, the Severity and/or Probability is impacted in different way. Consider how the risk matrix changes with these new values.

For Personnel it changes the 1st risk down 3 P levels (rows) and changes it to Serious. Risk 2 stays Medium
For Equipment it reduces Risk 1 to Low, and 2nd Risk to Low as well
For Logistics it has no impact on Risk 1, and eliminates Risk 2.
Personnel is still the biggest concern, even though all have been reduced to some extent
4 points – 1 for re-calculation of each category and 1 for identifying Personnel remains the biggest risk category. You may award ½ points as needed as there are 2 risks per category.

Q3: Now factor in the actual Mitigation and Contingency plans. Specifically, the cost of implementing these plans is given in the list with each action. Considering how often an action appears in the table, how much it costs, and the Outcome of Actions (Q2), indicate a) which will be the most expensive category to apply actions to, and b) whether it is worthwhile to apply the actions given the cost information.

11 Actions:
1. *(Mitigation) Appears only once and with 2, but reduces probability a lot. Cost is HIGH*
2. *(Mitigation) Appears twice, both under Personnel, and cost is HIGH*
3. *(Contingency) Appears once alongside 3 other actions and costs MEDIUM*
4. *(Contingency) Appears once alongside 3 other actions and costs LOW*
5. *(Mitigation) Appears once alongside 3 other actions and costs HIGH*
6. *(Mitigation) Appears once alongside 3 other actions and costs LOW*
7. *(Contingency) Appears once alongside another contingency action and costs MEDIUM*
8. *(Contingency) Appears once alongside another contingency action and costs HIGH*

9. *(Mitigation) Appears once alongside another mitigation action and costs LOW*
10. *(Mitigation) Appears once alongside another mitigation action and costs MEDIUM*
11. *(Mitigation) Appears once alongside 2 other mitigation action on 2 separate risks and costs MEDIUM*

Personnel has 3 actions: HIGH+HIGH+MEDIUM
Equipment has 6 actions: MEDIUM+LOW+HIGH+LOW+MEDIUM+HIGH
Logistics has 3 actions: LOW+MEDIUM+MEDIUM

a) Equipment is the most expensive category – 3 points, they should sum up as I did (they may not list out all 11 like my list, but they do need to indicate somehow the volume of costs associated with the 6 actions. -2 if they do not state the reason. -1 if they say Personnel instead of Equipment.

b) *Which actions to apply requires an interpretation of the data, acknowledging there is some more data we would like (how much are H,M,L costs? How much impact does an individual action have on the different risks?). The question is intentionally obfuscating these details in order to get the students to reason about the tradeoffs. Look for answers that are BASED ON THE DATA IN THE TABLE (2 points) and COMPARE ALTERNATIVES (2 points). 4 points total. Below is what I would respond, but it really isn't important to match my answer; again they have to state answers based on data and comparing between alternatives!*
   a. *For Personnel I would invest in action 2 as this is a big risk category. I would also invest in 7 as it is medium cost and impacts Equipment as well. I would probably skip 1 due to the high cost, and is somewhat orthogonal to #2*
   b. *For Equipment I would have already invested in 7, and in the absence of information that tells me otherwise, I would invest in 4 and 6 as low cost options for the 2nd risk (since it starts at Serious). #3 and #5 cost more.*
   c. *For Logistics I would invest in #11 as it appears in both and the cost is MEDIUM.*

Activity 3: Individual Task (10 points)

Individually take the DISC personality test. You should then map the DISC type to one of the 3 personality types under Motivation on slide 13 of your PM overview notes. Optionally (only if you are comfortable), I'd like you to work with a partner (even out of class) and have them take the DISC test pretending that person is you, and see what result that person gets for you. Finally, and again only if you are comfortable, I'd like you to work with your capstone team to compose all DISC personality types and see if they are "compatible". Please submit a summary of your personality results and of the optional discussions in a brief (min 300 words) well-written summary that adheres to the general grading rubric.

3 points for evidence showing they took the DISC test and 2 points for mapping to one of the 3 personality types in the notes (Task-oriented, Self-oriented, or Interaction-oriented). The part about working with a partner was optional so I am not including it in the grading rubric. The other 5 points are for the writing rubrics which we used on past labs. -1/2 point if < 250 words, -1 if < 175 words. -1/2 point for every TWO spelling, grammar, and punctuation errors up to a max of 2 points off (8 errors). The remaining 2 points for writing style.

Activity 4: Individual Reflections (10 points)

1. Describe (400 words) how your Capstone (or 316 if you are not in Capstone) team has gone through the "formin', stormin', normin', and performin" stages of the Tuckman model that is popular in Agile team practices (ignore "adjourning"). Provide a timeline and the events that exemplify each stage (and I know it may not be the case that you have gone through all 4 stages yet). Please then share with at least one of your Capstone team members and discuss similarities and differences in your summaries. Jointly write a 200 word comparison (in can go in each of your individual submissions, but identify the team member).
2. Provide a paragraph min (300 words) that describes how your Capstone team deals with risk, even if you do not have a formal risk management process. First, is there any discussion of risk in the project management deliverables you have been asked to produce? Second if yes, do you follow these procedures, and if so, are they helpful? If you do not have them defined or you do not follow them, then describe how you *informally* deal with risk – risk is a reality whether you have a formal plan or not, and it comes in many forms – late on deadlines, team members not participating, lack of stability in your incremental sprint releases. How does your team presently deal with these and other risks you can identify?

5 points for #1 the 400 word statement. -1/2 for < 350 words, -1 for < 250. Up to -2 for S, G, and P as specified in Activity 3. 2 points for connecting to Tuckman's model and including a timeline.
2 points for joint 200 word comparison. 1 point for length and SGP (½ for at least 150 words, ½ for < 4 SGP)
3 points for #2. ½ for <250 words, ½ for < 4 SGP. 2 points for answering the 2 questions asked for.

The Software Enterprise in ASU's BS in Software Engineering

Kevin Gary, Associate Professor
School of Computing, Informatics, and Decisions Systems Engineering, Arizona State University
kgary@asu.edu, (480)727-1373

The Software Enterprise is a multi-semester sequence designed to accelerate student learning of practical, "real world" considerations in applied computer science. The Enterprise has several defining characteristics that separate it from other problem-based courses:

Continuous – The Enterprise is currently designed as a contiguous four-semester sequence. This sequence exposes students, in a specified order, to all phases and roles of the software process lifecycle, and provides a continuity to the experience.

Scalable – Students work through two project lifecycles instead of one while in the Enterprise. The continuous, yearlong project cycle allows for larger teams and larger projects, so students get exposed to a myriad of issues that occur when scaling up.

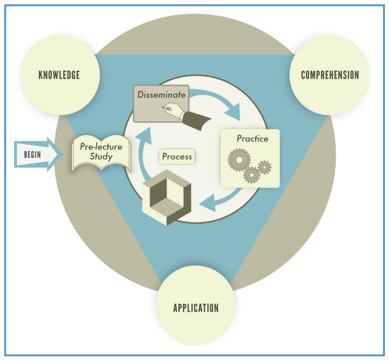
"Real-world" – Students are exposed to the full spectrum of forces affecting software development projects. Teams are asked to cope not only with technical issues but also with social or soft-skill issues with real-world industry sponsors.

Collaborative – Students work in teams across academic year boundaries. Students role-play, with participants responsible for different process-oriented roles. Teams deal with outside roles, such as customers and CXX-level management. Teams work with other Enterprise teams. For example, 4th semester students of the Enterprise mentor and manage students in the 2nd semester.

The five semester sequence leads students through "Personal Software Process", "Tools and Process", "Construction and Transition", "Inception and Elaboration", and "Project and Process Management". By the conclusion of the Enterprise sequence, students have an appreciation for the role of software process, the challenges of software maintenance, the impact of open source, the pros and cons of off-the-shelf software integration, business considerations in building software, and other practical aspects of software development.

Our vision for the Software Enterprise is to educate students on the following principles:

- Provide a situated educational experience for students that expose them to impacts of organizational, social, and cultural influences on the success of software projects.
- Help students realize software decisions *are* business decisions; such decisions are not made based on technical attributes alone.
- Help students understand stakeholder expectations; Project Managers, Customers, End users, SMEs, Business Development, etc.
- Provide extended hands-on enterprises applying the techniques learned in traditional classroom settings so new graduates not only comprehend the subject content but know how and when to apply it.



- Have student learning evolution follow the learning track of new professionals so as to expedite the assimilation of our graduates as new hires in software organizations in the greater Phoenix metro area and beyond.

The Software Enterprise employs a teaching and learning model (shown at right) combining traditional lecture, group engagement, problem-centered learning, reflection, and hands-on practice on scalable applications. In this model, students attend a reduced lecture schedule to create awareness and understanding, put the lecture concepts into practice through group lab sessions each week, integrate these skills into the scalable yearlong project, reflect on this rapid learning experience, and engage in group discussion regarding lessons learned. It is our premise that this model facilitates comprehension through to applied knowledge, making Enterprise graduates better prepared for the workforce upon graduation.

The Enterprise is currently in its 12th year. With support from the National Science Foundation, Kaufmann Foundation, and ABOR, and State Farm, the Enterprise has engaged in multi-university projects with a variety of industry sponsors on over 75 projects.

References:

1. Gary, K., "The Software Enterprise: Practicing Best Practices in Software Engineering Education", *International Engineering Education Special Issue on Trends in Software Engineering Education*, 2008.
2. Gary, K., "The Software Enterprise: Preparing Industry-ready Software Engineers", *chapter accepted to Software Engineering: Effective Teaching and Learning Approaches and Practices*, Ellis, H. (ed.).
3. Gary, K., Koehnemann, H., and Gannod, B., "The Software Enterprise: Facilitating the Industry Preparedness of Software Engineers", Proceedings of the National Conference of the American Society of Engineering Education, 2006 (ASEE '06).
4. Gary, K., Gannod, G., Koehnemann, H., and Blake, M.B., "Educating Future Software Professionals on Outsourced Software Development", Proceedings of the National Conference of the American Society of Engineering Education, 2005 (ASEE '05).
5. Gary, K. Gannod, G., Koehnemann, H., Lindquist, T., and Whitehouse, R., "WIP: The Software Enterprise", Frontiers in Education, 2005 (FIE '05).

Support:

1. Gary, K. "The Software Enterprise: A Reinforcing Pedagogical Teaching Model", National Science Foundation Course Curriculum and Laboratory Improvement Phase I Grant (NSF CCLI I), January 2009-December 2001, $148,938.
2. Gary, K., Koehnemann, H., Sannier, A., Kagan, A., and Acharya, R., "Agile Methods for Entrepreneurship", Enterprise@ASU *Pathways to Entrepreneurship Grant (PEG)* Award ($39,731) through support from the Kaufmann Foundation, November 2007 (implementation 7/1/08-6/30/09). *This award supports a variety of activities, one of which is to introduce a cohort to the Software Enterprise, joining DCS students with B-school entrepreneurs.*
3. Gary, K. and Gannod, B., "The Software Enterprise: Preparing Industry-ready Software Engineers" Arizona Board of Regents Award ($24,041), Learner-Centered Education (LCE) program, April 2005. *This grant supported initial work creating LCE labs and post-graduations surveys of the Enterprise.*
4. Gary K., Ghazarian, A., and Mazzola, D. "The Software Enterprise", State Farm Foundation Good Neighbor Grant, $40K ($18K share). 2015

Student Written Work - Sample 1

Curbing Cyber Attacks Through Improved Cyber Security Standards and Software Development Practices

Tresor Cyubahiro, Wesley Davis, Kevin Hale, James Ortner, Scott Watkins

Team: A Business of Ferrets

SER 416: Software Enterprise: Project & Procedures

April 12, 2019

Introduction

Looking back in history, there have always been those who try to skirt or exploit the rules and take advantage of the systems in place that govern and shape society. These people all have different reasons for their actions whether they be for the greater good of the community or for personal gain. Unfortunately, not all cultures have the same values or ethical alignment and so in order to protect their property, they need to defend in the respective manner. To better illustrate how one group of people defend their property, an examination of the progression of society can help enlighten the gravity of what the world is dealing with today. Native American tribes that would battle each other for territory would do so in hand-to-hand combat. Nations fighting across oceans would use navies. As technology advanced, nations around the world began to use aircraft as the next form of battle. Each time that a new offensive strategy was made, then countermeasures were made to defend.

In today's society, the new offensive tactics take place not in a tangible manner, but a digital one: the internet. The era of today is dubbed "The Information Age", and as such, many of the threats that we face are not from armies, navies, or fleets, but from those who wish to be influencers and change opinions. Oftentimes, these digital attacks are directed at society and society doesn't even notice, especially because almost everyone carries with them a means to be attacked digitally with them in their pockets and handbags. A side effect of living in the Information Age is that society has incredible stores of information at their fingertips within seconds.

While physical attacks used to take months of planning and months of execution, a digital attack can be executed within seconds and affect millions. What can be done to mitigate the damage of current and future attacks? One method of doing so is through improved software development, with cybersecurity at the forefront of the design and implementation. Instead of focusing on rapid development to satisfy the desire for the here and now, focus instead on security standards that will mitigate future damage by evolving digital threats. By asking the following questions, a better direction can be gained on how to better prepare for more cyber attacks. What technical approaches are developers currently using to combat these cyber attacks? What process approaches are managers and team leaders using for the development process? What are the current and future impacts of cyberattacks on society? How do these standards and impacts of current events impact the Software Development Lifecycle?

Technical Summary

Ian Sommerville, author of *Software engineering*, states that a cyber attack is "an exploitation of a system's vulnerability. Generally, this is from outside the system and is a deliberate attempt to cause some damage." (Sommerville 2011, 303). Sommerville also says that "When you consider security issues, you have to consider both the application software (the control system, the information system, etc.) and the infrastructure on which this system is built" (Sommerville 2011, 367). This approach to mitigation of cyber attacks involves the developers, but also the team leads and project managers. Each of these groups of workers have different methods of verifying that the end product is secure, and each method comes with its own positives and negatives.. The approaches of the developers are more technical in nature and are related to the source code directly. These methods are focused on catching defects before they appear in a production environment and include either manual or automatic reviewal of the code. A few of these methods include verification and formal methods, model checking, and statistical testing. On the other hand, the team leads and project managers focus their efforts on ensuring that the process of development fosters security focused design decisions. These approaches include software process improvement, risk management, defect management, and work breakdown structures. The decisions and plans that this second group makes generally affect the first and are highly dependent upon the estimates of work from the first group. In order to mitigate the number of security issues, both teams must do their parts to ensure the whole application stack is secure.

For the technical approaches, the first method to discuss is verification and formal methods which are the systematic reviewal of all the source code, either manually or automatically. One approach to this method is in-person, formal reviews of each individual line that is written. The main benefit achieved from this is that through traceable requirements, each line of code can be traced back to a specific requirement that addresses certain standards that it must reach. For example, a software application deployed in hospitals must conform to the privacy policies as stated by HIPPA. A formal code review can ensure that the proper measures were taken to correctly implement such features and allow the team to trace the specific methods and functions back to the requirements of the code. This approach is very effective to find and remove defects as long as the team is diverse enough to be able to understand where and why defects would appear. This method is also very expensive and time consuming, which is one of the reasons that it is not used as often as it should be.

Another method that is more effective for embedded systems but can still be utilized for general software products is model checking. Model checking is the process of creating a state model of all the possibilities of a product and verifying that for a given situation, the state of the model is as expected. This is useful for testing security issues as many different scenarios can be tested quickly to ensure that all responses are nominal. Just as the last method was expensive, this method has an exponentially increasing cost relative to the size of the model as the number of states increase. This also increases the number of areas that could potentially inject security flaws. Continual testing of this method are computationally expensive and generally not useful for large software products.

The final and most effective method of ensuring product features are consistent, is automated static analysis which can provide the benefits of in-person formal reviews and the results can be viewed and verified much more efficiently. As the developers add features to the product, they also brainstorm as many ways conceivable to bypass or break the feature and use those test cases as a basis for how the feature should be designed. This method also allows versioning of the product and ensures backwards compatibility. These tests are run every time a new feature of the project is implemented to verify that all other parts still work as previously intended. Automatic static analysis is the most cost effective of all the technical approaches to security and could be viewed as the most effective for verifying backwards compatibility.

In terms of the software processes, many of these approaches can be integrated with security in mind. Defect Management can be used to keep track of vulnerabilities, which Pfleeger Charles P. defines in his book *Security in Computing* as "a defect, which enables an attacker to bypass security measures"(Pfleeger 1997). Therefore, a lot of the same processes and methods used to manage defects in a software project can be used to manage software security vulnerabilities. However, where software vulnerabilities differ from normal defects is in the fact that vulnerabilities have lots of data available by multiple sources for analysis. Agencies like the National Institute of Standards and Technology (NIST) have a database of vulnerabilities that constantly get updated, and lots of companies offer Common Vulnerabilities and Exposures (CVE) databases for software developers to utilize. Tools are available to search software projects for these vulnerabilities, and Github will even alert users if it detects projects using libraries with vulnerabilities from open source databases mentioned above. This can make finding vulnerabilities much easier than finding defects in some cases, because not only are the software project team looking for these security defects, but millions of people around the world are able to contribute to discovering these problems and make software more secure. These vulnerability databases can also be used to help with the software process of defect estimation, as many of the databases list the severity of these software defects. When a defect is detected, estimating its size can be useful in determining how immediate a fix needs to be applied to the software. The same can be used for vulnerabilities, as not all security inneficices are equal. For instance, the vulnerability database NIST provides lists a base score of how critical the defect is, a score of its impact, and a score of its exploitability, all on a scale from zero to ten. These numbers helps software developers estimate how long it will take to fix the vulnerability, also help them determine how critical the vulnerability is. So not only does security get introduced into defect management and defect estimation, but often times will directly contribute to these software processes.

Another software process that can be used, while keeping security in mind is the work breakdown structure (WBS). The WBS is used to breakdown the different components that make up deliverables within the software process. The way the WBS is structured will vary a little depending on what is needed for the software process of the specific project. In terms of security itself, the WBS can easily be used in order to give a high level overview of the security tasks that need to be completed in order to limit the number of security defects that make it into the final build of the software. However, to make full use of the WBS, it is important for the team to be properly organized in order to use this process to its full potential. In the article “Work

Breakdown Structure (WBS) in Project Management", by Vasile Zecheru and Bianca Georgiana Olaru, they liken the project manager to an architect, as the project manager is often in charge of creating the WBS (Zecheru and Olaru 2016, 63). Additionally, the project manager is the main beneficiary of the WBS application technique . Aside from the project manager, the rest of the team is responsible to complete the deliverables that are present on the WBS and it is assumed that the team will have high performance overall. Additionally, the components that make up the WBS should be structured in a way that prioritizes the most important tasks first. Security itself is only a small part of this process, but the high level overview that it provides can lead to the team knowing which tasks need to be completed for the software process as a whole.

In order to ensure a high level of security in a software, organizations may also utilize the approach of process improvement and process maturity. This approach is employed in order to gradually improve an organization's software development process to a mature level by introducing good and standardized software engineering practices into the organization's software development process. As Sommerville puts it, "The level of process maturity reflects the extent to which good technical and management practice has been adopted in organizational software development practice" (Sommerville 2011, 706). As a result of taking the approach of software process improvement and maturity, organizations are able to produce good quality software, which correlates, to some degree, to secure software.

The software process improvement and maturity approach incorporates practices such as statistical quality control. As Sommerville describes statistical quality control, it is a practice which is based on measuring the number of defects and deriving a correlation between the defects and the process [Sommerville 2011, p. 707]. The objective of this practice is to "reduce the number of product defects by analyzing and modifying the process so that the chances of introducing defects are reduced and defect detection is improved" [Sommerville 2011, p. 706]. With this practice, the organization can incrementally review every improvement cycle, and whenever a lower defect count is reached in an-ongoing cycle, the current process is then standardized and re-employed in the subsequent cycles while adding changes and enhancements that improve the software development process and software project management.

Case Study: A Software Security Marketplace Case Study (September 2011)

In September 2010, HP acquired the source code analysis tool vendor known as Fortify. This transfer process, however, ended up taking over a decade to complete. In the end, this technology ended up in the hands of a provider with a global reach after starting as just a project funded on a federal research grant (McGraw 2011, 9). The transfer that happened here is exceedingly rare because while it took over a decade to complete, the sheer difficulty of the transfer also required millions upon millions of dollars of investment to research and develop what was needed to complete this transfer.

Traditionally computer security was focused on networks; little focus was given to securing computers outside of this realm. This of course was an attempt to create barriers such as a firewall, to keep attacks created by malicious hackers from hitting machines that were vulnerable. Firewalls today are just about everywhere, but firewalls alone are not enough as serious security issues continue to persist despite their widespread implementation.

Starting in the late 90's, the needs for computer security had evolved, birthing what is known as software security. The idea was to engineer software that is able to continue functioning properly even when it's under a malicious attack (McGraw 2011, 9). Efforts working towards integrating better security practices into their software development life cycles was spearheaded by Microsoft. The company Cigital, which was also responsible for early versions of the technology base used by Fortify, was also highly important in bringing software security through its professional services to a wider market (McGraw 2011, 9). The combination of the efforts from both of these companies, as well as other companies working towards similar goals lead to the creation of the Security Development Lifecycle (referred to as SDL from here on).

This led to the creation of the Building Security in Maturity Model (BSIMM), which describes the activities used by enterprise-level software security initiatives and is used as a gauge to measure these initiatives (McGraw 2011, 9). From this model came three commonly used methodologies: Microsoft’s SDL, Cigital’s Touchpoints, and the Comprehensive, Light-weight Application Security Process (CLASP), These methodologies were created by the Open Web Application Security Project (OWASP) (McGraw 2011, 10). All of these methodologies aim to inject quality security practices into the life cycle of software projects. This includes a range of different tests and reviews during a project’s life cycle, but the most commonly used practices are architectural risk analysis and code reviews.The main reason this is true is because every software project has to have code, which also means it can be reviewed. Additionally, it is now possible to easily automate some of this process using static analysis tools, which were invented and developed by companies such as Cigital.

Citigal was founded in 1992, under the name Reliable Software Technologies, and at the time was funded through federal grants from various agencies. Starting in the late 90s Citigal turned its focus on working on security for Java. This included many software-centric approaches towards discovering issues such as scanning code for potential security risks and fault injection. Eventually Citigal developed their tool ITS4, which was the world’s first code scanner for both C and C++ (McGraw 2011, 10). Overtime the patterns that Citigal was able to automatically parse for issues became more sophisticated and could be used for more use cases.

Eventually a new code scanner called Mjolner was created. This scanner far surpassed ITS4, but was not initially ready for use outside of the testing done by scientists working on it.

In the early 2000s, Mjolner was renamed to SourceScope, and was seen as a research prototype that was dependent on many open source tools to even work (McGraw 2011, 11). Due to the state that the program was in, Cigital only used it for consulting engagements for code review (McGraw 2011, 11). So while SourceScope had the ability to discover more interesting vulnerabilities than its predecessor ITS4, it was also incredibly painful to try and use, and required the user to have knowledge of navigating a program's source code during code reviews.

Later in 2003, Citigal ended up licensing their SourceScope technology to a startup partner associated with Kleiner Perkins Caulfield & Byers firm. This startup is what would eventually become Fortify, and at the time only had a small team of four members. This team then tore SourceScope apart and rewrote it from the ground up. This eventually paid off and they were able to create a commercial-grade product from the previously barely functional product (McGraw 2011, 11). After seven years Fortify had managed to get several release cycles out to several hundred customers in the real world.

After all this time, the tools were finally production ready and Fortify started to get noticed by large technology corporations. Their initial tools were for black-box web application testing tools, but they followed up with tools for white-box testing next (McGraw 2011, 11). These many years of hard work eventually lead to the company becoming integrated into HP and work on creating better tools and improving old ones still continues today.

Standards in Security

Standards define how products such as software should be implemented for the safety of the consumer and to ensure product reliability. Standards also help in assessing the quality of a product and establish the basic requirements a product must meet prior to the planning or development of the product. Basically, standards have helped set guidelines for the creation of many products ranging from computers to cellphones resulting in reliable products such as the Windows PC and Apple iPhone. In this section, we will discuss the brief origins of security standards in software engineering, the Trusted Computer Systems Evaluation Criteria (TCSEC) or “The Orange Book”, and ISO 15408 also known as Common Criteria; as they pertain the importance of creating a reliable and secure software product.

A Brief History

The origins of security standards begin with the United States Department of Defense (US DoD) and the need to protect computers that emanated electromagnetic frequencies. The TEMPEST standard, considered to be the first computer science security standard, was created by the DoD in the late 1950s to determine the acceptable level of emanation used to transmit classified information. This standard was created to ensure that any information that was deemed classified was transmitted in a secure way from sender to receiver without the potential threat of an “electronic eavesdropper”. This standard would then later produce the Industrial TEMPEST Program (ITP), a program that was established around the notion of three goals; to set commercial standards; designate criteria for testing equipment that met the set standards; and to certify vendors equipment (Yost 2007, 599 - 600). The only issue with the TEMPEST standard, was that it was focused and geared toward devices that used electrical emanation, making this standard unacceptable or not needed for devices that would later work on networks.

It was not until the 1970s that another security standard was established, which would address the multiple facets of computer technologies such as computer networking and time-sharing. The *Department of Defense Trusted Computer System Evaluation Criteria* or TCSEC for short, was the culminated efforts of the National Computer Security Center (NCSC) and the MITRE Corporation to standardize government procurement requirements. The TCSEC also provided structure in which manufacturers would use to evaluate and measure the overall security of their systems (Yost 2007, 606). TCSEC was the standard that was most commonly used within the US and sparked European countries to begin research and development of their own variation of the TCSEC called the *Information Technology Security Evaluation Criteria* or ITSEC (Yost 2007 609).

At this point, the TCSEC standard had not yet reached the domain of local area network (LAN) and wide area network (WAN) systems. It was not until 1987, where the NCSC had published the Red Book, which included interpretations of the TCSEC with security features, assurance requirements, and ratings structures of computers that resided on a LAN or WAN or even internetwork systems (Yost 2007, 609). The standards created in the improved TCSEC Red Book were not the end all be all solution to the continuous change of the security standards in computing technologies. With the consideration of improvements from the Redbook Series and research from a variety of countries, including the International Standards Organization (ISO), the Common Criteria standards were completed in 1996. They would later be changed to ISO 15408 after the Common Criteria version 2.0 was passed and established in 1998 (Yost 2007, 610).

TCSEC: The Orange Book

As mentioned earlier, TCSEC was one of the original standards created by the DoD to try and provide a generalized structure in which organizations could ensure their products were secure. TCSEC's overall premise was that it was possible to evaluate a system and place that system into a category that would be understood to hold a certain degree of trust with communities that required the differentiation of classified and unclassified data. These different categories could be broken down into four different levels of security protection (where from left to right is strongest to weakest): A – Verified Protection, B – Mandatory Protection, C – Discretionary Protection, and D – Minimal Security Protection. Each category would then also have numbered sub-categories where the higher the sub-category reference, the more secure the product was evaluated to be (Yost 2007, 607). For example, for a product to be classified as an A2 in the TCSEC security standard, evaluators would have to perform formal verification on the product all the way down to the source code level (Lipner 2015, 24).

The evaluation process for TCSEC proved to be more rigorous and time consuming that what was originally thought. For companies that were ensuring their product would at least reach a C2 or B1 level of standard acceptance they had to ensure that the documentation for their product included the design documentation for evaluators to understand how the system functioned (Lipner 2015, 27). Since design documentation could potentially lead to exposure of sensitive materials such as proprietary system components, vendors created design documentation that would be used to satisfy TCSEC evaluators needs, instead of guiding developers to learn and understand how their implementations worked. This also lead evaluators to create their own interpretation of the TCSEC standards to match their understanding of how the product they were evaluating should work (Lipner 2015, 27). Overall, the TCSEC was an initial attempt to create generalized standards that should be used to evaluate security products; however, due to misinterpretations and lack of proper product standing, TCSEC fell short in its effort to provide a surplus of quality security system.

ISO 15408: Common Criteria

The global community in computing information security developed the Common Criteria standards in the mid-1990s with the help of ISO. More commonly, Common Criteria was changed and referred to after its second version release as ISO 15408 in 1998 (Yost 2007, 610). ISO 15408 establishes the basic concepts of information technology security and specifies the models of evaluation for products that wish to become evaluated under this standard (ISO 2009). This standard supersedes the TCSEC standard, although there are some similarities between the two, especially in how ISO 15408 breaks down its evaluation into several evaluation tiers.

The criteria in which a company would evaluate their product under the ISO 15408 standard can be broken down into seven different security criteria levels called Evaluation Assurance Levels. Each level is labeled as EAL 1 – EAL7, where the higher the value, the more rigorous testing has to be to validate that the product is secure (Yost 2007, 610). For example, if a company was wanting to ensure their products meet EAL 4, they would have to make sure their product was methodically designed, tested and reviewed to ensure that the product could withstand penetration attacks (ISO 2009). ISO 152408 is still in use today, and in fact recently had a revision by ISO in 2014, more information on this standard can be found on the organization's website.

The need and usage of standards in the domain of security is extremely important to the trust and protection of proprietary information worldwide. As seen in history, security standards did not stay stagnant, nor did they simply cover one aspect of a technology stack. As different technologies came about the need to create standards to prevent potential risks evolved and grew overtime. As we continue moving forward in computer technologies, the need to create and further develop our current standards with also continue, all to ensure a quality and trustworthy product.

Case Study: Facebook Security Breach (September 2018)

In September 2018, Facebook's engineering team announced that they had discovered a security breach on their systems. This was a massive security breach which had affected approximately 50 million accounts, according to the official report from the company (Rosen 2018, par. 1). The report mentions that this security breach came about due to a then-newly implemented feature "View As" which allowed users to see what their own profile looks to somebody else checking their profile. This feature however introduced a number of vulnerabilities and bugs to the Facebook application systems. Consequently, attackers were able to take advantage of these bugs and vulnerabilities and stole Facebook access tokens from millions of accounts, which they used to log into the systems as the users whose access tokens had been stolen.

This security breach is relevant to the discussion of software and cyber security because it originated from software defects. According to the report generated by the company, the security breaches came about as a result of three particular defects which were introduced during the implementation of the "View As" feature (Rosen 2018, par. 10). Firstly, the intended functionality of this feature was to make it a "view-only interface" (Rosen 2018, par. 11). This meant that the users would only be able to view the View As page as it was rendered and in no-way should have been able to interact with any of the user interface components on this particular view. However, as the report mentions, one of the user interface components which normally allows users to upload videos "incorrectly provided the opportunity to post a video."

(Rosen 2018, par. 11). Additionally, this video uploader would incorrectly generate an access token that granted the current user the permissions of the Facebook mobile application. Lastly, the access token generated by the video uploader was not for the current user, but rather for the user whom the current user was looking up (Rosen 2018, par. 13). The report adds that the combination of these three bugs made it possible for attackers to gain access to many Facebook users' accounts and log in into the system as the people whose access tokens had been stolen (Rosen 2018, par. 14).

Companies which collect and handle such sensitive user data should have an even stronger and more thorough SDLC processes in order to ensure such cases do not happen. Also, in cases where the security breaches do actually happen, such organizations should put in place efficient risk mitigation strategies in order to minimize the amount of damages such security breaches can bring about. Looking at this issue from the outside, one can argue that these bugs could have been found before release by implementing commonly used processes in software development such as code tests and code reviews. However, it is common that bugs end up in release versions and only surface during the operations and maintenance phase of the SDLC. Plus, new developments and enhancements to software systems introduce changes which when integrated with current features can introduce un-anticipated security vulnerabilities, as was the case for this case study. Therefore, it is important that organizations follow and emphasize activities such as conducting continuous monitoring during the operations and maintenance SDLC phase [Kissel 2018, p. 34] and assessing system security during the implementation and assessment SDLC phase [Kissel 2018, p. 30].

As is a standard practice for companies developing security-vulnerable software systems, such companies' SDLC must include activities and plans which ensure that some of the preventable issues and risks are dealt with or planned for in advance and that preventive measures are built-in in the software systems as they are being developed. Some of the activities implemented in SDLC that Facebook implemented include continuous monitoring. The report from the company mentions that this security breach was discovered by the engineering team. The team was then able to analyze the breach and trace it back to its origin in the newly implemented feature "View As". In addition, Facebook had in place a number of risk mitigation strategies and plans to deal with such security breach. Firstly, the company reported that it fixed the bugs and vulnerabilities, and temporarily disabled the "View As" feature in order to conduct a thorough security review of this feature. Secondly, the company reported that in addition to informing users and law enforcement of this security breach, it also "reset the access tokens of the almost 50 million accounts we know were affected and also took the precautionary step of resetting access tokens for another 40 million accounts that have been subject to a View As look-up" (Rosen 2018, par. 16).

All in all, Facebook's security breach of September 2018 is a relevant case to software security because it is an example which illustrates the inter-connection between software security and SDLC, its processes and activities. It also illustrates and puts in perspective the effect of setting and implementing risk mitigation plans in cases where security breaches do actually occur despite the activities which may be performed during SDLC to ensure that the probability of such breaches occurring is reduced.

Current and Future Impact Discussion

The discussion of software security has seriously impacted software development, not only in how software is designed currently but also in how systems and designs will be engineered in the future. The National Institute of Standards and Technology (NIST), one of the leading agencies of software standards and security, states that “executing a risk management-based approach for systems and projects means integrating security early and throughout the agency’s established system” (Kissel 2008, 4), and that implementing such a policy “plays a significant role in measuring and enforcing security requirements throughout the phases of the life cycle.” (Kissel 2008, 4). This means that a project that desires to implement security into their software need to do so at the earliest stage of their development, and continue to keep security at the forefront of their development throughout the entire software development life cycle. This impacts how software is architected, designed, and developed.

Instead of the topic of security being just another feature to add as an afterthought, security needs to be at the core of a software project and inside every part of the software development life cycle. This change in mindset also affects legacy systems. NIST outlines its security considerations for new projects, but also states that those considerations “are still relevant to … legacy systems, and should be applied and documented to ensure security controls are in place and functioning effectively to provide adequate protections for the information and the information system” (Kissel 2008, 8). Depending on their state, these legacy systems and their maintenance could be seriously impacted by the need to keep these adequate protections.

Software security also affects how the future of software development will progress. As software is trending more and more towards cloud deployment and virtualization, the discussion on software security is changing how these new technologies are being designed, tested, marketed, and are overall driving the future of software as a whole. The advancing technology of virtualization, or the ability to run multiple virtual computers with their own independent OS on one physical computer, brings with it new security questions and topics.

The technology of virtualization brings many benefits, but NIST points out that it also "requires additional security planning for unique security risks inherent in virtualization…such as data interception…, keystroke logging …, denial of service to the host's resources", and the exacerbation of security issues not unique to virtualization "such as malware, data leaks, patch management, and weak access controls." (Kissel 2008, 44) These unique problems caused by virtualization has led to large companies like Amazon and Google developing cloud solutions with security tools to help manage these security risks. For example, Amazon Web Services' home page at https://aws.amazon.com/ lists seventeen different services and tools they provide specifically for security, identity, and compliance. All of this taken together, it starts to become apparent how far reaching and impactful software security is on the past, present, and future of software development. From legacy systems needing maintenance or rehauling, to modern projects needing to design with security in mind from the beginning, to the future of software systems having more and vastly different types of security concerns that will shape how development will proceed, software security affects every aspect of software development.

Impact on the SDLC

During all stages of the software development life cycle, you can find reasons to look at security. However, two of the stages most affected by security are Requirements and Testing. During the Requirements stage, the core features and use cases of the software are determined. When security is a concern, the requirements phase is the best time to begin anticipating security problems and architecting solutions. NIST asserts that "early planning and awareness will result in cost and time saving through proper risk management planning. Security discussions should be performed as part of (not separately from) the development project to ensure solid understandings among project personnel of business decisions and their risk implications to the overall development project" (Kissel 2008, 13). When security discussions happen as part of the Requirements phase, the implementation is not only smoother during the other stages of the software development lifecycle, but can also make the rest of the requirements and design of project better.

This has a large impact on how the entirety of the project is designed, and requires everyone associated with the project to come to a mutual understanding of the security issues and their corresponding proposed solutions. In their paper, *Software Security Rules: SDLC Perspective*, C. Banerjee and S. K. Pandey recommend that "all the personnel from requirement engineers to maintenance engineers and other stakeholders should make themselves aware about the latest software security issues, especially the critical ones. For SDLC team, this awareness should be more technical, and, for other stakeholders, the awareness should be more general, but necessary." (Banerjee 2009, 127) This first, and important stage in the software design lifecycle is greatly affected when security is taken into consideration; not only from a length perspective, but from a people perspective. It requires everyone involved in the project to be aware of security in order for the rest of the software process to be most successful.

Another software lifecycle stage greatly affected by security is Testing. Banerjee and Pandey explains that "security rules will broaden the role of test engineers and they will be able to choose the appropriate tools and techniques for testing the software from a security point of view, with focus on destructive testing." These tools and techniques for destructive testing will be vastly different from the usual validation testing model, such as unit tests and functional tests.

NIST also explains that by adding security, activities to “plan and conduct system certification activities in synchronization with testing of security controls” becomes necessary, as well as “complete system accreditation activities” (Kissel 2008, 28). All of these validation activities take time, and often time accreditation is a very involved process as well.

These new activities taken together with the normal activities that happen at Testing make the stage quite a bit longer than before, and puts a greater importance on making sure everything is validated correctly. While there are certainly other stages in the SDLC impacted in similar fashions, Requirements and Testing are the ones that will always end up having a large change compared to a project that doesn’t focus on security. As the world of technology advances further and an increasing amount of sensitive data is being exposed to the internet, it is more important than ever to insure that the software handling this sensitive data is designed and architected in such a way to keep it secure, and is validated to work as expected. These things would not be possible without a large impact on the Requirements and Testing phases in the SDLC.

Conclusion

Cybersecurity is the new defensive frontier and software developers are the soldiers on the front lines. As society continues to advance in this information age, the number of attacks will continue to grow and so should the efforts of the developers to ensure that private information stays private. This can be accomplished as a team effort between the developers and project managers to verify that the code is secure. An examination of the different risks and strategies to mitigate those risks needs to be developed and analyzed to make sure that the product being built is actually what is needed and that extraneous, potentially insecure features are not added. There have been many attempts to create tools to help in this effort but it has only been in recent years that the focus has shifted to a security first development approach. However, even though these tools are becoming better, as seen in the Facebook case study, defects can still make their way into the final products. This can be seen as a major impact on the software development lifecycle as it increases time and money spent to validate the software and tools for each application. Also, if a bug does make it into the final code, finding and fixing the issues that arise can be mitigated with proper procedures and policies in place to protect the further security breach.

Cyber attacks are changing the way that society uses the internet and the way that software is being developed. As more companies adhere to security standards, then the public can trust software that they use for personal matters as it has been rigorously tested and vested. The most impacted technology today being affected by security first development is cloud computing. Cloud computing is being used by all forms of businesses and should be the first line of defense. In order to combat the unseen enemies, development should be hyper-focused on what will protect us. Just as the world adapted to different types of attacks, so should society today adapt and create ironclad defenses for digital content to protect the unseen boundaries all around.

Bibliography

Banerjee, C., and S. K. Pandey. “Software Security Rules, SDLC Perspective.” *International Journal of Computer Science and Information Security* 6, no. 1 (n.d.): 123–28. https://arxiv.org/abs/0911.0494.

“ISO/IEC 15408-1:2009.” ISO. International Organization for Standardization, January 1, 2014. https://www.iso.org/standard/50341.html.

“ISO/IEC 15408-3:2008.” ISO. International Organization for Standardization, May 1, 2011. https://www.iso.org/standard/46413.html.

Kissel, Richard, Kevin Stine, Mathew Scholl, Hart Rossman, Jim Fahlsing, and Jessica Gulick. Security Considerations in the System Development Life Cycle , Security Considerations in the System Development Life Cycle § (2018). https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-64r2.pdf.

Lipner, Steven B. “The Birth and Death of the Orange Book.” *IEEE Annals of the History of Computing* 37, no. 2 (2015): 19–31. https://doi.org/10.1109/mahc.2015.27.

Mcgraw, G. “Software Security.” *IEEE Security & Privacy Magazine* 2, no. 2 (2004): 80–83. https://doi.org/10.1109/msecp.2004.1281254.

Mcgraw, Gary. “Technology Transfer: A Software Security Marketplace Case Study.” *IEEE Software* 28, no. 5 (2011): 9–11. https://doi.org/10.1109/ms.2011.110.

Pfleeger, Charles P. “Security in Computing.” Essay. In *Security in Computing*. Upper Saddle River (New Jersey): Prentice Hall, 2000.

Rittinghouse, John W., and William M. Hancock. “Cybersecurity Operations Handbook.” Essay. In *Cybersecurity Operations Handbook*, 4–44. Amsterdam: Elsevier/Digital Press, 2003.

Rosen, Guy. “Security Update.” Facebook Newsroom, September 28, 2018. https://newsroom.fb.com/news/2018/09/security-update/.

Sommerville, Ian. “Dependability and Security.” Essay. In *Software Engineering*, 289–302. Boston, MA: Pearson, 2011.

———. “Dependability Engineering.” Essay. In *Software Engineering*, 341–55. Boston, MA: Pearson, 2011.

———. “Security Engineering.” Essay. In *Software Engineering*, 366–86. Boston, MA: Pearson, 2011.

———. “Process Improvement.” Essay. In *Software Engineering*, 706–07. Boston, MA: Pearson, 2011.

Yost, Jeffrey R. “A History Of Computer Security Standards.” Essay. In *The History of Information Security a Comprehensive Handbook*, 599–610. Amsterdam: Elsevier, 2007.

Zecheru, Vasile, and Bianca Georgiana Olaru. “Work Breakdown Structure (WBS) in Project Management.” *Revista De Management Comparat International* 17, no. 1 (March 2016): 61–69. https://search.proquest.com/docview/1814287304.

Student Written Work - Sample 2

Scaled Agility in Software Development

Maurice Ajluni, Vatricia Edgar,

Joshua Goralczyk, Natallia Karaliova, Jeremy Powell

Project Management 416

Professor Baron

April 13, 2018

TABLE OF CONTENTS

INTRODUCTION

Brief **H**istory **o**f **D**evelopment **M**ethodologies

Software engineering is more than just "programming." The development of high quality software products requires planning, design, and testing to ensure that requirements are met and customers are happy. The earliest framework for software development was "waterfall," a method which simply lays out the stages of development in a practical order: requirements, design, implementation, verification, and maintenance. It was first formally described in 1956 by Herbert D. Benington (Henseler 2018). Software development frameworks continued to evolve, with the Spiral method introducing iterative methodologies in 1986 and the methodology of "Scrum" being first mentioned that same year (Henseler 2018). The paper *Scrum methodology* was published in 1995 by Ken Schwaber and Jeff Sutherland, and in 2001, the *Manifesto for Agile Software Development* was published by the Agile Alliance (Henseler 2018). The *Manifesto for Agile Software Development* declared agile to be radically different from the earliest framework, waterfall, with one simple sentence: "Responding to change over following a plan" (Beck, et al. 2001). Every framework which falls under the agile umbrella shares the common goal to reduce the overhead of process and documentation, to increase collaboration and interaction, and to always welcome change.

Agile **F**rameworks

Agile processes arose as to address the software industry's need for constant change. Unlike other types of products, software is relatively easy to change, even after development has begun, due to software itself having no physical parts. Different agile frameworks, such as Kanban and Scrum, implement the agile principles to handle such changes. Kanban offers continuous development, embraces change at any time, and promotes the idea of not having roles to govern the team. This process is generally used in small development groups (Rehkopf 2019). Scrum adds an extra layer of complexity with fixed length development intervals, called sprints, during which developers work on user centric features called user stories from the sprint's backlog. A burndown chart tracks the team's progress towards the sprint goal and is used to determine the team's velocity, or the amount of work the team can do in a sprint. Scrum also has a set of roles, such as Scrum Master, Project Manager, the development team, and uses several types of meetings, daily stand-ups, end of sprint reviews and retrospectives and beginning of sprint planning, to maintain order and address issues. The suggested size of a Scrum team is between four to eight people. For large projects who want to take advantage of agile, the small scale frameworks that began the agile movement are not sufficient.

Scaling **A**gile

Most agile frameworks were initially developed for use by a single, small team. While useful for small businesses and small projects within large businesses, existing agile frameworks did not address the issues large projects face such as managing more developers and merging workflows. This lead to the development of new agile frameworks, intended to be used at a large scale. Some of the most popular of these "scaled agile frameworks" include Large Scale Scrum (LeSS), the Scaled Agile Framework (SAFe) and Disciplined Agile Delivery (DAD). Each framework is grounded in agile principles, but they take different approaches in achieving their goal of helping large projects and businesses reap the benefits of lean, agile development. As with any framework, none of these may be declared "the best." Which framework should be used depends on the needs of the project and the company as a whole. In this paper, we will explore the process of each of these large scale agile frameworks, their benefits and drawbacks, and the future of scaled agile.

TECHNICAL SUMMARY

Technical **S**ummary: **D**AD

Originally designed by IBM, the purpose of Disciplined Agile Delivery is to take methods from multiple other proven development techniques and use them to fill in the gaps left behind by other agile methods (Ambler 2013, 3). DAD helps development teams find the most efficient practices, and to describe these methods for future use. On top of delivering a shippable product, DAD also focuses on creating supporting documentation. The idea is to create "potentially shippable solutions," not just "potentially shippable software" (Ambler 2013, 3). Customers desire more than just a piece of software, they expect a solution to their problem that is understandable and easy to use. While a stakeholder is a common concept in many agile methods, DAD thinks in terms of stakeholders, not just users; it is important to consider not only what the end user needs, but the business expectations and needs of the stakeholders as well. This is because it is usually not possible to discuss directly with an end user what they desire in the software. However, the group ordering the software is available for discussion and feedback as the process continues (Ambler 2013, 3).

Disciplined Agile Delivery takes the successful aspects of multiple other agile methods, such as Scrum, Extreme Programming, Lean, and many others, and merges them in a manner that creates a goal driven environment for agile development. According to Ramakrishnan, one of the greatest advantages of agile processes is the array of practices available (Ramakrishnan, Gopinath, and Ambler 2019, 1). However, this may also be one of its greatest disadvantages; it becomes very challenging to decide which practices are the best for an entire team and to organize them together. Disciplined Agile guides these decisions and shows how the different methods may work together to create effective production.

Technical **S**ummary: **S**AFe

Developed by Dean Leffingwell in 2011, The Scaled Agile Framework (SAFe) focuses on leveraging the benefits of the classic agile methodology on enterprise level by accommodating enterprise level teams that range in size from 50 to 125 individuals. The Scaled Agile Framework integrates individual activities into four organizational levels: *Team, Program, Value Stream, and Portfolio* (Alqudah et al. 2016, 5). SAFe Agile *Teams* typically combine Agile methods (Scrum and Kanban) over short iterations with smaller user stories, demo the working system at the end of each iteration, and conduct retrospectives on potential improvements and work-in-process. Under the doctrine of the Scaled Agile Framework, collections of agile teams are referred to as Agile Release Trains (ART). The release trains are assembled from individuals skillful in competencies needed to define, build, test, and deploy a working solution. Each ART establishes a Continuous Delivery Pipeline comprised of Continuous Exploration, Continuous Integration, Continuous Deployment, and Release on Demand. Each component of the pipeline provides a feedback loop from customers back to the development team helping the ART to redefine the vision, roadmap, and set of features for a solution. The Scaled Agile Framework defines the *Value Stream* as “long-lived series of system definition, development and deployment process steps which is utilised to develop and deploy systems that supply a constant flow of value to the business, customer or end user” (SAFe White Paper 2018). While SAFe identifies two types of value streams (operational value streams and development value streams), its primary focus is on the development value stream which is defined as “the people and steps used to develop new products, systems, solutions, and services sold by the enterprise, or that support internal operational value streams.” Development Value Streams (and the ART Agile Teams) aim to achieve the strategic goals of the *Portfolio.* SAFe’s

White paper emphasizes that SAFe *Portfolio* management strategy forfeits traditional fixed annual planning and budgeting in favor of strategy and investment funding designed to put the people who are doing the actual work (portfolio stakeholders) in charge of funding value streams. At *Program* level, SAFe introduces additional teams needed to coordinate the alignment of multiple ARTs: a Business Owner team of three to five stakeholders with "the ultimate fiduciary, governance, efficacy and ROI responsibility for the value delivered by a specific release train"; a Release Management Team (RMT) overseeing releases; a DevOps team providing "tighter integration of development and operations as well as maintaining deployment readiness for the program", a System Team whose responsibilities constitute providing assistance in building and using the development environment infrastructure (CI/CD, build and test environments, Test Automation Frameworks) (SAFe White Paper 2018).

Since its original release in 2011, SAFe has undergone five major updates fostering the evolution and expansion of the framework in response to volatile market conditions driven by changing customer needs, and emerging technologies. The latest version of The Scaled Agile Framework, SAFe 4.6, was released on October 3, 2018 and introduced the Five Core Competencies of the Lean Enterprise: Lean-Agile Leadership, Team and Technical Agility, DevOps and Release on Demand, Business Solutions and Lean Systems Engineering, and Lean Portfolio Management (SAFe White Paper 2018). Scaled Agile, Inc (the central hub for SAFe framework started by Dean Leffingwell, the creator of SAFe and Chief Methodologist at Scaled Agile, Inc) provides enterprise level training and certification for SAFe with 70% of Fortune 100 companies and over 300,000 practitioners trained to date. As demonstrated by numerous case studies conducted by Scaled Agile, Inc in partnership with small and large enterprises (including Air France, Capital One, Intel, Swisscom, Cisco, HP, Discount Tire, Valpak, John Deere, etc) implementation of SAFe methodology has contributed a 20 to 50% increase in productivity, a 25 to 75% improvements in quality, a 30 to 75% faster time-to-market, and a 10 to 50% increase in employee engagement and job satisfaction (Leffingwell 2018).

Technical **S**ummary: **L**eSS

The scaled agile framework Large-Scale Scrum was developed by Bas Vodde and Craig Larman to accommodate for large companies that have a large number of development teams, and multisite product development. LeSS follows the belief that less management, roles, and overall organizational structure removes unnecessary complexity, allowing the product owner to work directly and equally with individuals (Kalenda 2018, 4). Vodde and Larman explain that when scaling Scrum, their approach is to adopt standard one-team Scrum and build on it (Vodde, Bas, and Craig Larman 2013, 1). This is because LeSS examines the elements of a one-team Scrum and endeavors to reach the same goal within the constraints of standard Scrum rules. Vodde and Larman elaborate that the dissolution of single-function, specialized subgroups, such as a testing group or architecture group, is also a necessary step to scale Scrum (Vodde, Bas, and Craig Larman 2013, 1).

With LeSS, traditional job titles and management roles are dissolved; this leaves the focus on the individual and the product owner. Additionally, LeSS eliminates the traditional team lead or project manager roles, and embraces that there is "no organization wide standard process that everyone must follow" (Vodde, Bas, and Craig Larman 2013, 2). Instead, the team is treated as one team, with no specialized roles, and has a dedicated Scrum master following under the Product Owner. Despite having multiple teams, LeSS still follows a One-Team Scrum. Therefore, there is only "one product backlog, one definition of done, one potentially shippable product increment, one (overall) product owner, one sprint, and no specialists on any team"

(Vodde, Bas, and Craig Larman 2013, 2). Essentially, all of the teams work in a 2-4 week sprint towards a common goal to deliver a potentially shippable increment while maintaining a daily Scrum under their Scrum master.

The LeSS process begins with a two-part Sprint Planning. The first part is under the absolute order of the Product Owner, with at most two representatives from each team attending. The attending team members can slice their cut of the product backlog while the Product Owner breaks any conflicts. Next is the second stage where the team independently works within their group to assign tasks. However, if the need occurs to coordinate, it is accepted that the needed teams communicate between each other. This idea transcends to the daily Scrum where if information sharing is needed, the teams may attend each daily scrum. To further refine this process, an optional "Inter-team coordination meeting" where representatives from each team share information. Halfway through the sprint, each team undergoes a "product backlog refinement" where planning for future sprints begin. It is recommended to locate in a large room, which can accommodate every team member, and have white boards or other tools to discuss thoughts to "apply rotation writing and other large-group workshop techniques" (Vodde, Bas, and Craig Larman 2013, 3). Next is the Sprint Review which consists of the Product Owner, other stakeholders, and two members from each team. Instead of a grand reveal of their sprint's progress, a "'science fair'-style" presentation takes place where each team shows their respective features (Vodde, Bas, and Craig Larman 2013, 3). The stakeholders or Product Owner can then oversee each feature and pay more attention to their interests while the representatives explain. A recommended, optional approach before the Sprint Review is to informally seek the Product Owner or other stakeholders to "reduce the inspection and discussion that would otherwise be required spent at the Sprint review." Finally, is the Team Retrospective, which is the same as a

Scrum Meeting where the team talks about their difficulties and future plans. As an additional step, there is an optional Joint Retrospective. This step is held early, usually within the first week, in the next sprint, where Scrum Masters and a representative from each team discuss improvements to the for the overall product or organization (Vodde, Bas, and Craig Larman 2013, 3).

LeSS also has a plan for working across multiple locations. LeSS multisite depends on continuous integration for every step, Free Open Source Software (FOSS) and strong communication and trust between team members. This is because multisite scaling tends to run into problems involving licensing, testing, or even bottlenecking due to a member's knowledge, or lack thereof (Vodde, Bas, and Craig Larman 2013, 3). For successful Scrum, communication is invaluable. LeSS mitigates the multisite lack of communication with "free, ubiquitous tools" such as "Google Hangouts or a projector". Furthermore, the Sprint Planning and Product Backlog depend on the aforementioned online meeting tools and online web tools such as Google Spreadsheets (Vodde, Bas, and Craig Larman 2013, 3). As for the sprint itself, it is a modification of a one-team sprint, which is later discussed.

Large-Scale Scrum has two variants of which both focus less on the process and more on the individuals (Kalenda 2018, 4). The first variant of this process comprises of eight or fewer agile teams that work within an organization. This variation combines the methodology of a "one-team scrum" with the methodology of a larger number of teams. The second alternative is called "LeSS Huge" which involves nine or more agile teams working within an enterprise. LeSS Huge manages its requirements by using scaled practices which include "requirement areas, area product owners, and area product backlogs" (Kalenda 2018, 5). Because of hundreds of people and many teams, the Product Owner has too much overhead from the entire process.

To mitigate this, Less Huge introduces "Area Product Owners" who serve to focus on one area backlog (Kalenda 2018, 17). The Product Owner then communicates with the "Area Product Owners". Vodde and Larman argue that while a divide-and-conquer approach is unavoidable with large enough teams on a product, it is still inflexible, slow, and wasteful when dividing into specific groups of interest, such as architecture (Vodde, Bas, and Craig Larman 2013, 5); divide-and-conquer via architecture contradicts agile's philosophy of flexibility, and the overall overhead between handing it off and dependence on work-in-progresses makes the process ineffective. Instead of dividing by specific groups, LeSS divides around the customer's major area of requirements (Vodde, Bas, and Craig Larman 2013, 5). As a result, a "requirement area" column is added to the Product Backlog where the teams under their respective Area Product Owner communicate and follow LeSS process.

RELEVANT STANDARDS

ISO/**I**EC/**I**EEE **S**tandards

IEEE Standard 1074, approved, in 1991, is an early IEEE standard for software development processes, specifically for organizations developing their own development process. Although published before the widespread popularity of agile, the standard was already loosely amicable to agile methodologies. Under section 2.1, it states that a software life cycle model must be chosen for the development of a software product, and that while no specific implementation is required, all of the activities outlined in the standard should be present in the life cycle model. The standard may seem strict with a set of required activities, yet it clearly states that the activities are "non-time-ordered." They include every process from project initiation to retirement, which the iterative methods of agile may complete in any order. The standard even mentions Rapid Prototyping, an iterative approach, as a sample software life cycle model (IEEE 1991, 19). Even before agile became popular, at first amongst small teams, the standard already paved the way for larger organizations to adopt agile practices by remaining loose enough that large organizations could adopt agile practices while continuing to conform to standards set forth by the IEEE. Future standards continue this today, with sections and sometimes whole standards dedicated to adapting the standards to agile methodologies.

ISO/IEC/IEEE Standard 12207 describes the standards for software lifecycle processes. The standard's introduction defines software engineering as a "means to enable the realization of successful software system." The introduction asserts that a "structured development process" is formed in the process of software engineering, beginning with conception of a software product and moving through to maintenance. While a "structured development process" may seem counter to agile principles, the standard is actually quite compatible with agile. Inclusion of multiple disciplines and dedication to stakeholder needs are key principles highlighted in the standard's introduction which are also key to the scaled agile frameworks we have seen (ISO/IEC/IEEE 2017, vii). DAD has a heavier focus on all stakeholders, not just end users, which is in line with ISO 12207's approach to meet the needs of all applicable stakeholders, business technical or otherwise. In fact, the interactive, customer centric approach that all of these frameworks is in line with the standard. Annex H: Application of Agile describes the many ways that businesses using agile can claim conformance to the standard. The section points out that as long as every step in the software process is completed, order does not matter. That is to say that the iterative approach of agile, where the "stages" of software development might be done in any order or even in parallel, is not in conflict with the standard (ISO/IEC/IEEE 2017, 127). The two-part Sprint Planning meeting used in LeSS is an example of a planning and design phase which happens over and over, each sprint, in an iterative fashion. It corresponds well with the Project Planning processes in the standard. Reviews and retrospectives, present in all of the frameworks presented here, correspond well with the Project Assessment and Control processes. While scaled agile conforms to the goals of the standard, it does not necessarily conform strictly to the activities demanded by the standard, instead opting to combine or otherwise change them. Annex H of the standard then suggests that agile organizations claim conformance to outcomes, as described in 4.2.1 of the standard (ISO/IEC/IEEE 2017, 127).

ISO/IEC/IEEE Standard 26515 is about documentation in agile projects. This standard is especially applicable to DAD, since DAD takes a holistic approach to developing, ensuring that the user receives a workable solution complete with software and documentation. The standard acknowledges agile's drive for less documentation, but also acknowledges the user's need for clear and helpful documentation. It cites that a reduction in training and help desk support, and an increase in good reputation are both benefits of good user documentation (ISO/IEC/IEEE 2018, vi). LeSS and SAFe both describe a focus on lowering process documentation, but no emphasis is placed on end documentation for the user. In this way, DAD is much more amicable to this standard, as user documentation is baked into the process. It was a deficiency which the creators of DAD sought to resolve (Ambler 2013, 3).

Standards **o**f **D**AD

Each of the frameworks has its own set of principles, which can be seen as the standards for using the framework. An example of this is Disciplined Agile's principles, described by "The Principles Behind Disciplined Agile." The goal of Disciplined Agile is for teams to be disciplined. For a team to work well together and to know what is good for them, a team leader must also be disciplined, as without a strong leader teams may work inefficiently and possibly in a toxic social environment. It takes practice and unbiased observation to see when problems arise and to determine their best solution (The Principles Behind Disciplined Agile 2019, 1). This record describes the seven principles that act as a standard for what Disciplined Agile should be and how it should be run.

First to be discussed is delighting customers. The best way to keep a customer happy is to not only meet their needs but to exceed them. This is often not possible in software projects due to time and budget constraints. DAD aims to have the team build these expectations into their plan from the start. By increasing efficiency and quality the development team can produce products that go beyond what was asked for while still staying within time and budget constraints.

The next standard of Disciplined Agile Delivery is motivation; a team should be motivated. Choosing individuals who work well together and enjoy working together is an important factor in that teams output. A team leader and the development environment should support teamwork and give resources that will help to lead the developers to success.

After motivation, "The Principles Behind Disciplined Agile" considers pragmatism. Closely related to delighting the customer, the product should be as good as it can be. If more features can be added that do not conflict with the stakeholders desires then they should be. If existing features can be improved in some way, they should be. Being pragmatic in software development may mean doing more than just what your process says. This is where agile comes in; it is built to change. Disciplined Agile is where the additional effort and pragmatism comes in. DAD encourages the resulting product to be the best it can be.

Context and choice are the following standards. Every choice made for a team from the team members to the strategies to use depends upon the context of the project. Depending on the context, what the project and team needs, they can make a choice as to what type of methods from other agile practices to use. Such a choice allows a team to find what techniques work best for them and for their current project. Disciplined Agile focuses on giving the teams the option to use varying other methods they know when best appropriate.

Finally, the last practices and standards are optimization and awareness. Having enterprise awareness means that an individual is working towards the goals of their entire organization not just towards those of their specific work assignment (The Principles Behind Disciplined Agile 2019, 1). This ties into optimization, having an organization and workplace designed so that developers can maximize their efficiency and quality not only affects optimization, but makes everyone thats a part of the team aware of their teammates and goals.

IMPACT ON Software Development Life Cycle

SDLC: **D**AD

Due to the idea of combining multiple other agile methods in the most appropriate way for a specific team and project, the lifecycle of DAD development can vary. While it is mainly based upon Scrum and Kanban it encourages the development team and leadership to adopt other practices where useful (Ramakrishnan, Gopinath, and Ambler 2019, 1). DAD follows the general agile model and uses the general phases such as Inception, Construction, and Deployment. But it uses various alternate ideas within these phases; a team could use Sprints and retrospectives, while also removing formal roles like in LeSS or using a continuous integration format like Spiral.

Because DAD encourages the use of other known agile methods when they best fit, it does not change the Software Development Lifecycle in a significant manner. The only changes come from when the inherited agile methods change it themselves. Changes from DAD are made in how each phase of the cycle is completed, not their order or inclusion in the cycle. Thus, DAD has little impact on the Software Development Lifecycle in general. In a more specific context however, it can foster new versions of the cycle that have not been seen or used before. By combining the other agile techniques unique processes can be created, and unique development life cycles along with them.

SDLC: **S**AFe

SAFe's approach to software development life cycle combines the elements of agile (both Scrum and Kanban) and Lean Software Development. SAFe most prominently affects the planning and development stages during its software development lifecycle, through the use of Agile Release Trains, Program Increments and the planning tasks that go with Program increments. The Agile Release Trains (ART) which comprise of conventional size agile teams operate in eight to twelve week iterations referred to as Program Increments (PI). PI is a timeframe during which an Agile Release Train (ART) "delivers incremental value in the form of working, tested software and systems". A PI typically starts with a PI Planning session followed by execution iterations, concluding with one Innovation and Planning iteration. During PI planning session, Agile teams under the umbrella of ART discuss the deliverables for the integration and demo at the end of the Program Increment. For each execution iteration within the PI, the Release Train Engineer (RTE) facilitates a weekly or bi-weekly Scrum of Scrums meeting in order to coordinate the interdependencies between the ARTs, identify any roadblocks, and discuss the progress of the PI in respect to the overall vision. Each execution iteration is concluded with an iteration review. Iteration reviews are critical in maintaining transparency and a flow of information in the feedback loop between the team and the Product Owner. While arbitrary, four execution iterations are considered to be most effective in a PI, however, the framework allows flexibility to better accommodate specific organizational needs. PI concludes with the System Demo event where the Agile Release Team presents the integrated overview of new features to Business and Product Owners, customers, and other ART stakeholders. Iterative and incremental in its nature, SAFe's approach to software development life cycle captures the essence of agile and promotes the best product development rhythm custom-tailored to bridge market requirements and teams' capabilities (SAFe: Program Increment 2018, 1).

SDLC: **L**eSS

Like the other frameworks, LeSS affects all aspects of the software development life cycle because it is a framework for the software development process. However, the focus on removing complexity and roles leads LeSS to remove specialized testing and design groups, leading to a large effect on the design and testing phases of the software development life cycle. In traditional scrum, the product owner negotiates a internal contract with the Research and Development (R&D) managers who are responsible for the release. However, LeSS believes the main power and vision is under direct control of the Product Owner. As a result, there are no separate R&D roles and no internal contract (Vodde, Bas, and Craig Larman 2013, 1); the role is simply eliminated and places more emphasis on the Product Owner and individual. This means that the designing phase is altered due to the lack of specialized R&D roles, but emphasizes more power to the Product Owner. Additionally, Scrum tends to have sprints that range from two to four week sprints with a potential shippable deliverable at the end of each sprint. As previously mentioned, LeSS believes to dissolve the idea of a big release, managing systems, practices, and its roles. This implies that a "sequential lifecycle" is eliminated, no "prior analysis phase, architecture phase, and no following integration/testing phase [and] the groups that were attached to each phase." However, LeSS states that instead of a potential messy deliverable, sequential lifecycle development is eliminated, and the groups that comes with it (Vodde, Bas, and Craig

Larman 2013, 1). Instead, LeSS believes in continuous integration multiple times a day verified by automated testing. If a build breaks, a mitigation strategy of stop and fix emerges (Vodde, Bas, and Craig Larman 2013, 3). Overall, LeSS aims to follow a one-team Scrum approach, which emphasizes on each individual and the product owner, ultimately altering the design and testing phases.

CASE STUDIES

Case **S**tudy: **D**AD

There have been several implementations of DAD in industry; one notable case is from the company OpenLink Financial in 2016. OpenLink decided to make a change to their development techniques when they decided they needed to modernize their methods. They determined that too much time was spent on maintaining product quality and not on implementing new functionalities (Scott Ambler Associates, Inc. 2016, 1). Scott Ambler and Associates (SA+A) guided the company to adopt a combination of agile and lean methods, in the form of DAD practices.

The transition to a disciplined agile approach first began with training. A workshop for executives and product management was held to educate OpenLink leadership of agile and disciplined agile practices and advantages. There were two teams assigned with projects under the new DAD process as pilot tests. Scott Ambler and Associates acted as coaches for the teams to ensure they used the new models to its full potential and received the most benefit from it. One of the changes that SA+A made to the workplace at OpenLink was "Agile Pods." They went about redesigning the office space into, "three configurations of Agile Pods with sizes of 9 to 20 team members," (Scott Ambler Associates, Inc. 2016, 2). Team collaboration was shown to dramatically increase and resulted in better team functioning, better quality software, and shorter development times. After these first two pilot projects were successful SA+A and OpenLink decided to go further. They spent the next two years launching more teams and projects with this model in mind. Once OpenLink management and leadership had gotten the hang of operating the DAD model by themselves, SA+A.

Scott Ambler and Associates still returns to OpenLink periodically to help those who have mastered the basics of the methods they learned with digging deeper into new techniques and plans. Since integrating the Disciplined Agile Delivery ideas and methods OpenLink has recorded, "a dramatic increase in productivity, lower defect counts, fewer customer escalated issues," and more improvements (Scott Ambler Associates, Inc. 2016, 1). Tom Marrapodi, Vice President of Development at OpenLink claimed that the adoption of the DAD methodology helped, "to dramatically improve… the morale and effectiveness of our teams and their focus on producing high quality software..." Along with several other testimonies from another OpenLink Financial executive and the company as a whole, it is clear that the adoption of disciplined agile ideals in their workplace has guided them to a much more productive environment for the company and developers.

Case **S**tudy: **S**AFe

According to the data collected from a series of case studies conducted by Scale Agile, Inc (the organization started Dean Leffingwell, the founder of the Scaled Agile framework), one of the largest reported SAFe deployments took place at Intel's Manufacturing Development Organization (SAFe Case Study: Intel 2019). In a competitive information technology industry, Intel must continuously improve through innovation, cost control, and quality. As of 2019, Intel employs more than 100,000 people globally with a net revenue of $59.4 billion. Intel's Manufacturing Development Organization (MDO), examined in this case study, is responsible for testing and validation of Intel solutions. Today, MDO produces over two million lines of code every two weeks. Operating at such volume presents significant challenges in terms of people and process management. To mitigate that, the Manufacturing Development Organization first adopted Lean-Agile practices in 2005, later integrating Scrum in 2012, and then started developing homegrown solution for scaling Scrum (SAFe Case Study: Intel 2019).

As the size of the division kept increasing, MDO began to face challenges scaling Scrum to meet its every-growing volume when in 2013, MDO discovered the Scaled Agile Framework. Allen Ringel, Lean & Agile Transformation Leader at Intel who was in charge of the transition to SAFe, points out the reasoning behind the decision to implement SAFe: “In an organization as large as MDO we needed to standardize the planning and execution process we use to work together to deliver value. When we encountered SAFe it provided a proven, public framework, with well-defined roles and artifacts for applying Lean and Agile at the enterprise level.” (SAFe Case Study: Intel 2019).

During the first Program Increment planning event, MDO trained more than 1,500 people, assembled and launched eight Agile Release Trains (ARTs) comprised of 170 Scrum teams, and hired 15 Intel Lean-Agile coaches at the 14 MDO’s sites. In 2017, MDO was merged with another group to form the group Manufacturing Value Engineering (MVE). MVE, now twice the size of MDO prior to the merge, rapidly trained nearly 2000 employees in an effort to integrate the new-comers into existing framework resulting in over 440 Scrum teams of software and hardware engineers organized into 35 ARTs (SAFe Case Study: Intel 2019).

In the case study findings, Ringel reports that the implementation of SAFe in the division has impacted the company as a whole and helped it maintain the industry’s rapid rate of growth by contributing to the 65% increase in product variants with the same capacity delivered by

MVE, the improvement of commit-to-accept ratios from 74% to over 90%, while maintaining the reduction of scope to less than 5%. SAFe deployment fostered increased transparency and strong performance-to-schedule discipline. “Lean & Agile help us deliver more products without adding more people, so we may stay competitive and keep up with Moore’s Law,” Ringel says (SAFe Case Study: Intel 2019).

CURRENT & FUTURE IMPACT

The challenge of scaling a Scrum process for companies and or teams of developers is not as much of a monstrous undertaking as it one was. As Martin Kalenda emphasises, “there are several frameworks [nowadays] that can be used to guide the scaling process in organizations” (Kalenda 2017, 3). Therefore, the current impact of these scaled agile frameworks is reflected in the large number of companies who have both adopted agile and successfully scaled the process into a larger framework to utilize its techniques. The impact of a framework is currently derived by that framework's adoption level. Collectively, 82 percent of companies use a scaled form of agile with the top framework being SAFe with 28 percent, LeSS with three percent, and DAD with a 1 percent adoption rate (Kalenda 2017). This is a significantly high number when accounting for all companies who are presently applying Scrum within their development teams. To get a sense in quantifying how large of a group works with some companies who use the LeSS framework, Bas Vodde states their method was “applied in groups [made up of] 1,500 people, involving seven developments sites spanning the globe” (Vodde 2013, 1).

The future impact on the industry by the scaled agile frameworks mentioned above will become evermore significant as they are gradually adopted and modified. When utilized correctly, an organization will have the upper edge increasing the possibility to be disruptive within the industry. As companies become evermore involved with software, “increasing [this] digitalization will cause most (if not all) companies to become software companies [and then into] ...software organizations" (Kettunen 2017).

CONCLUSION

With the ongoing rapid changes of software, a reflexive and responsive system that embraced change, while still following a plan, was needed. The "waterfall" methodology was too linear and could not respond quick enough for immediate actions. This gave to the rise of agile methodologies where the goal is to continually respond to change while following a plan. Frameworks, such as Scrum and Kanban soon sprouted from agile's goal, promising the reflexive and responsiveness needs. However, Scrum and Kanban tend to only fit for a team size of four to eight people. The difficulties were evident when projects required hundreds of team members or teams that are not located in the same area were trying to piece together their software and efficiently work together.

Disciplined Agile Delivery remedies this with the approach of creating shippable solutions, not just shippable software, while appeasing their main audience of stakeholders. DAD emphasizes on discipline, motivating team morale, and optimization, all while under a strong team leader. Additionally, DAD understands that to deliver a shippable solution, a combined goal driven methodology such as Scrum, Extreme Programming, and Lean is a common solution to the organization's needs (Ambler 2013, 3). The Scaled Agile Framework enhances the Agile methodology by integrating four new organizational levels: Team, Program, Value Stream, and Portfolio. By adding these four new levels, teams that range from 50 to 125 individuals are possible and earn the name Agile Release Trains, which establishes a continuous pipeline with a vision, roadmap, and set of features for a solution. To ease the communication between teams, SAFe introduces the Business Owner Team, Release Management Team, and DevOps team to oversee the other teams (SAFe White Paper 2018). Large-Scale Scrum takes a different approach. Instead of adding more roles and divide-and-conquer solutions, it states in order to remove unnecessary complexity to the team, the organization must dissolve management, roles, and organizational structure (Kalenda, 2018, 5). This puts more power into the product owner and follows a one-team scrum approach, the ultimate goal of delivering a potentially shippable increment while maintaining a daily Scrum under their Scrum master. LeSS has been shown to work across multiple locations using “free, ubiquitous tools” and has two versions to accommodate smaller and larger teams (Vodde, Bas, and Craig Larman 2013, 2, 5).

Agile is necessary in projects which need responsive and reflexive changes which can continuously deliver and show results. Scrum takes the agile principles and expands on it. Big businesses need agile practice but they need them to scale, and new software methodologies have emerged to fulfill this need. Evidence shows that DAD, SAFe, and LeSS can significantly improve the software development process while working across hundreds of team members. As companies grow, scaling agile will be seen as a norm due to the promises of agile and growth of businesses.

BIBLIOGRAPHY

Alqudah, Mashal & Razali, Rozilawati. (2016). A Review of Scaling Agile Methods in Large Software Development. International Journal on Advanced Science, Engineering and Information Technology. 6. 828. 10.18517/ijaseit.6.6.1374. Accessed April 08, 2019. https://www.researchgate.net/publication/311916796_A_Review_of_Scaling_Agile_Methods_in_Large_Software_Development.

Ambler, Scott W. "Going Beyond Scrum Disciplined Agile Delivery." Disciplined Agile Consortium. October 2013. Accessed April 8, 2019. https://disciplinedagileconsortium.org/Resources/Documents/BeyondScrum.pdf.

Beck, Kent, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, Robert C. Martin, Steve Mellor, Dave Thomas, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Ken Schwaber, and Jeff Sutherland. *Manifesto for Agile Software Development*. PDF. Agile Alliance, 2001.

Henseler, Tomás. "Timeline of Software Development Methodologies." Hexacta. February 05, 2018. Accessed April 09, 2019. https://www.hexacta.com/2018/02/05/timeline-of-software-development-methodologies.

IEEE Standard for Developing Software Life Cycle Processes. New York, USA: IEEE, 2017.

ISO/IEC/IEEE International Standard - Systems and Software Engineering — Developing Information for Users in an Agile Environment. New York, USA: IEEE, 2018.

ISO/IEC/IEEE International Standard - Systems and Software Engineering -- Software Life Cycle Processes. New York, USA: IEEE, 2017.

Kalenda, Martin, Petr Hyna, and Bruno Rossi. Scaling Agile in Large Organizations: Practices, Challenges, and Success Factors. May 16, 2018. https://onlinelibrary-wiley-com.ezproxy1.lib.asu.edu/doi/full/10.1002/smr.1954.

Kettunen, Petri, and Maarit Laanti. "Future Software Organizations – Agile Goals and Roles." *European Journal of Futures Research* 5, no. 1 (December 16, 2017). doi:10.1007/s40309-017-0123-7.

Leffingwell, Dean. "About Scaled Agile Framework." Scaled Agile Framework. October 03, 2018. Accessed April 08, 2019. https://www.scaledagileframework.com/about/.

Ramakrishnan, Gopinath, and Scott Ambler. "A Hybrid Toolkit." Disciplined Agile (DA). January 17, 2019. Accessed April 9, 2019. https://disciplinedagiledelivery.com/hybridframework/.

Rehkopf, Max. "Kanban vs Scrum." Atlassian. 2019. Accessed April 08, 2019. https://www.atlassian.com/agile/kanban/kanban-vs-scrum.

"SAFe Case Study: Intel Staying Ahead of Moore's Law Intel MVE Delivers 65% More Product Variants." Scaled Agile Framework. Accessed April 08, 2019. https://www.scaledagileframework.com/case-study-intel.

"SAFe White Paper." Scaled Agile. November 2018. Accessed April 07, 2019. https://www.scaledagile.com/resources/safe-whitepaper/.

"SAFe: Program Increment." Scaled Agile. 18 October 2018. Accessed April 07, 2019. https://www.scaledagileframework.com/program-increment/.

Scott Ambler Associates, Inc. "ENTERPRISE AGILE TRANSFORMATION: A PRODUCT COMPANY'S SUCCESSFUL ADOPTION OF DISCIPLINED AGILE." Disciplined Agile Consortium. 2016. Accessed April 9, 2019.

https://disciplinedagileconsortium.org/resources/Pictures/Case studies/Approved Open Link Case Study.pdf.

"The Principles Behind Disciplined Agile." Disciplined Agile (DA). February 21, 2019. Accessed April 11, 2019. https://disciplinedagiledelivery.com/principles/.

Vodde, Bas, and Craig Larman. Scaling Agile Development. CrossTalk. June 2013. Accessed April 10, 2019. http://static1.1.sqspcdn.com/static/f/702523/22609354/1367558447003/201305-Larman.pdf?token=q/dHT7DJOn7C1byQkY2lTjVkvso=.

Student Written Work - Sample 3

Healthcare Systems

A look into the Software Engineering in the medical field.

Team 404

Elizabeth Layman	elayman1
Christopher Cody	cdcody
Jack Northcutt	jgnorthc
Nubian Tesfai	ntesfai
Joshua Bresette	jbresett

Table of Contents

INTRODUCTION

In our increasingly technology-focused world software permeates practically every activity and industry known to man. Software and computer systems guide planes through the sky, control entire factories worth of equipment, and create more data yearly than has ever been compiled by humanity to date. One important industry that has come to rely increasingly on software systems is that of healthcare. The business of saving and prolonging life is a vital component of modern society, and the numbers speak to that effect. According to the Center for Medicare and Medicaid Services, the United States alone accounted for annual healthcare expenditures of 3.5 trillion dollars in 2017, which made up a massive 17.9% of the country's GDP {CMS}. This industry is not likely to decline in the near future either, as current projections predict that healthcare will grow faster than GDP until at least the year 2027.

To many, healthcare is an industry and a business that holds great importance in the global economy. While the financial impact of this sector is clearly of great interest to society as a whole, it is also important to account for the individual person and their perceptions.. Healthcare has a deep personal importance to many people who have experienced issues with illness or their mental well-being, whether directly or through those they know. This importance stems not just from the visible life saving outcomes of many healthcare procedures, but also from a general capacity to increase peoples' quality of life.

Given the enormous importance of the healthcare industry on both people and economies, it follows that software engineering within this context inherits a great degree of this importance. There is great potential for growth in medical software, but the immense stakes involved result in an element of risk that is rarely matched among other domains. The risks involved in medical software stem from a variety of causes and can result in a wide range of negative effects on patients, healthcare clients such as hospitals, and the parties who develop such software.

One of the primary risks in the healthcare sector is the continued well-being of the people who depend on it. This is particularly the case in embedded device applications, as there are many people whose lives rely upon such devices every day. Medical systems are often tasked with monitoring insulin levels, dispensing medication, and other functions that require near 100 percent reliability to be considered effective. Software is often only a single component of these systems, but it is nonetheless critical in the proper function of these devices.

Unfortunately, the issues with medical systems are not always limited to merely failing homeostasis, but also have the potential to cause exceptional degrees of active harm. The dangers involved are of such magnitude that one of the most infamous disasters in all of information technology is an unfortunate example of medical software: the Therac-25 radiation therapy machines [1]. This incident resulted in the deaths of at least six people, which is a testament to the level of risk involved to the lives and quality of life for patients using medical software.

Although loss of life is arguably the most pressing consequence of the Therac incident and others, there are other interested parties across the entire market that are also affected heavily by such tragedies. The companies involved in medical software failures can be drawn into extremely costly lawsuits, which can harm businesses and careers, although such consequences could be considered just [1]. The Software Engineering Code of Ethics and Professional Practice defines 8 principles of ethical practice within the profession [2]. Four of these principles stand out in the context of such failures. These responsibilities are to act consistently with the public interest, act in the best interests of clients and employers, ensure that delivered products meet the highest professional standards, and to further the integrity of the software engineering profession within the public eye. These ethical and professional duties are a major part of what it means to participate in the healthcare industry, especially those with the decision making power to affect lives.

TECHNICAL SUMMARY

The high level of risk and serious ethical implications that pervade medical software demand a level of quality that cannot be met through individual measures or ad-hoc processes. Rather, practitioners within this domain, with assistance from federal agencies and private companies, have developed a rich ecosystem of technical and organizational techniques meant to assure positive outcomes throughout the lifecycle of each product. These techniques include national and international standards, improvements to the verification and validation processes, improved testing models, and high traceability requirements.

One of the key forces in the technical development of this sector within the United States is the Food and Drug Administration (FDA). While regulatory agencies rarely contribute new concepts, their ability to canonize certain practices and standards sends ripples throughout the whole software process. This is especially the case within the areas of software requirements engineering and system validation. Any software component in a medical device developed after June 1, 1997 is subject to the FDA's regulations on software validation [3]. While the term "software component" initially seems limited only to the product in question, it is much more broadly encompassing than might be expected. FDA guidance and regulations apply directly for the following categories of software: software used as a component, part, or accessory of a medical device, software that is itself a medical device (e.g., blood establishment software), software used in the production of a device (e.g., programmable logic controllers in manufacturing equipment), software used in implementation of the device manufacturer's quality system (e.g., software that records and maintains the device history record), and software used to manage electronic records and signatures [3].

One validation technique that applies to these categories of software is the design review. In a design review, the system’s current design artifacts and decisions are examined by stakeholders in and outside of the development team. The review team asks questions such as whether development tasks have been fulfilled and the proper outputs produced. These results are then documented and referenced against any other reviews or internal standards that may apply. Such design reviews are common in all fields of software in both formal and informal styles, but medical software is uniquely obligated to include this practice. According to section 3.5 of the General Principles of Software Validation, FDA compliant software systems must include at least one formal design review [3].

Another nearly universal validation technique used in medical software is software testing. All production software systems have an interest in assuring that code works correctly, and medical systems are no exception. Testing is often performed using standard metrics and techniques such as path coverage and boundary value analysis [3]. Path coverage tests every possible route from beginning to end: if a device had multiple switches and sensors, every combination of switch and sensor would be tested. Boundary value tests input values and incorrect values to see how the program would respond: if the software is expecting someone to enter a number between 0-100, what happens at 0, 50, 100, -1, 101, and ‘q’ are entered? Despite all these methods, the difficulty still remains that no non-trivial software system can ever be completely tested. The safety-critical nature of the healthcare industry has thus encouraged software engineers to adopt innovative and intensive new testing methodologies.

One such testing strategy is statistical software testing. In a general sense, statistical testing is often thought of as testing with a collection of random test cases. This simplified definition is problematic, however, as it misses out on many factors that make statistical analysis so powerful. In the words of statistical testing leader Jesse Poore, “The term should be understood, however, as the comprehensive application of statistical science, including operations research methods, to solving the problems posed by industrial software testing [4].”

One of the foundational concepts in statistical testing is the usage model. A usage model is a “formal statistical representation of all possible uses of a system” often represented as a familiar state graph [4]. One method that can be used to determine the nodes and edges within this diagram is a high level set of use cases. This has the advantage of interfacing well with UML use case diagrams and user requirements, but is not always formal or traceable enough to be sufficient for medical contexts. Another process that may be preferable is the formal sequence specification defined by Poore and Prowell [5]. This process has the advantage of adding determinism and traceability to the sequence, which is particularly desirable in the medical field.

The newly created usage model is then evaluated using simulations and walkthroughs of system usage. This process creates a large amount of data that can be used to determine the most failure prone sequences and processes, which then feeds into creating test cases that have a greater chance of discovering defects than simply generating numbers. While this method of software testing is fairly intensive, the scale and risk of medical software systems makes statistical testing a promising prospect.

Another way that software in the healthcare sector distinguishes itself is in a heavy emphasis on highly traceable requirements. Broadly speaking, traceability is about being able to ask “why” with regards to any given subsection of a system, no matter the level of granularity.

For instance, a line of code might be traced to a write buffer component, which might be traced to a caching routine, which may have been specified in order to fulfill a high level performance requirement.

One benefit of traceability is that speculative design and unused code artifacts are reduced. Another benefit that is increasingly relevant in today's distributed society is in providing a single source of truth across organizations [6]. When a project involves multiple team members across what might be dozens of locations, knowing an exact "why" is exceptionally valuable. By providing a detailed, project-wide set of requirements that links every line of code to a reason for existing, traceability cuts down on unnecessary complexity in the codebase. Complexity is a major cause of issues in almost any field, so risky sectors such as health are thoroughly incentivized to invest in such measures.

Traceability standards are valuable enough to be used in a variety of fields. Formalized specifications exist in the automotive, aerospace, nuclear, and rail control industries in addition to medicine [6]. Specific standards organizations in health software include ANSI/AAMI/IEC led by the Association for the Advancement of Medical Instrumentation, the European Medical Device Directive, the US FDA, and the ISO [6].

The specific standard that will be discussed here is the FDA's guidance on software validation [3]. One reason is that the FDA has arguably the greatest regulatory power of any of these organizations, so these standards are near universally applicable within the United States. Another is that its stipulations are fairly generic, which gives a greater understanding of traceability issues as a whole rather than one particular instantiation of these practices.

Traceability in the design phase is largely an issue of assuring that software requirements are sufficiently relevant to the system requirements and the results of risk analysis activities. The recommended solution for this task is a formal design review. Such a review will involve conversation and confirmation between business and technical staff that are meant to answer the key questions of traceability analysis. Once design artifacts are produced using these requirements, another check is made to verify that all design choices and software requirements are interconnected. This pattern continues for implementation and testing, with each component being traced up the requirements hierarchy. So long as this bidirectional flow holds for every component in the system, the FDA's traceability requirements are met. The specifics of how these checks are made vary by organization, but the general pattern is the same.

In conclusion, healthcare is a high-stakes industry that demands a great deal of technical competence from its practitioners. As such, the software engineers who work on such systems have invested immense amounts of effort into their methodologies and techniques. Detailed verification and validation processes give stakeholders confidence that medical devices will continue to improve the quality of life for patients. Among these processes are innovative testing methodologies that have the potential to revolutionize interactive software systems as a whole. Finally, an emphasis on traceable requirements has helped this field to become a leader in best practices for safety critical systems.

RELEVANT STANDARDS

Like many advanced Software Engineering domains there are set standards that Healthcare Systems must maintain and demonstrate. When it comes to healthcare systems, these standards apply greatly to clinical life-critical systems and are expected to ensure the utmost safety and reliability to the user. In order to ensure this level of safety and reliability, healthcare systems follow guidelines set by HIPAA. HIPAA provides two key rules that play a role in the type of software they allow to run their system. These rules are the HIPAA Security Rule, HIPAA Privacy Rule and National Institute of Standards and Technology Cybersecurity Framework. HIPAA Enforcement Rule is the factor that keeps the healthcare industry adhering to those rules.

One in regard to how Software Engineering influenced healthcare, would be information security. “A covered entity or business associate must comply with the applicable standards, implementation specifications, and requirements of this subpart with respect to electronic protected health information of a covered entity” [7]. Maintaining information and protecting patient confidentiality is crucial to healthcare profession. Healthcare professionals can pull a patients files in a fast, efficient and manageable way. The field was able to take the paper copy document and translate it to a digital copy that could be protected from improper data changes to stored files, provide authorized users access to documents quicker for patients with medical emergencies and effectively provided patients with a form of secure confidentiality between doctor and patient. “Today, providers are using clinical applications such as computerized physician order entry (CPOE) systems, electronic health records (EHR), and radiology, pharmacy, and laboratory systems. Health plans are providing access to claims and care management, as well as member self-service applications” [8]. Software engineering also provide technical safeguards for healthcare companies and employees. These include access control(provides authorized personnel with access to e-PHI), audit control(procedurally control the activity on e-PHI), integrity controls(measures to ensure e-PHI can not be improperly altered or destroyed), and transmission security(technical security to prevent unauthorized access to e-PHI packets that are sent over a network) [8].

Another one in regard to how Software Engineering influenced healthcare, would be information privacy. “A major goal of the Privacy Rule is to assure that individuals’ health information is properly protected while allowing the flow of health information needed to provide and promote high quality health care and to protect the public's health and well being” [9]. Privacy is extremely important in healthcare. Information protected by the Privacy Rule includes the patient's physical or mental health or condition, provision of healthcare to the individual and payment in the future toward their healthcare. One of the ways software can help provide an avenue for healthcare is securing documents for healthcare employees who are authorized to access them and enforce privacy for the patient. Software provided healthcare a way to move from paper to digital documents. This in turn restricts and lowers the amount of people handling confidential patient files and the less people who handle said confidential documents, the less of a chance of a violation towards HIPAA. While lowering handling it also can keep track of who is interacting with any information stored in their system, which we will go into later on. Other ways software can benefit healthcares privacy standard is by the data encryption(to eliminate even unauthorized people that do get their hands on data from knowing the contents), audit trails and password protection. The more boundaries a system has the better, due to more securities directly relating to less chance of privacy breach.

Along with those rules, the healthcare profession also follows a set of standards created by the National Institute of Standards and Technology (NIST) Cybersecurity Framework. “The Cybersecurity Framework provides a voluntary, risk-based approach—based on existing standards, guidelines, and practices—to help organizations in any industry to understand, communicate, and manage cybersecurity risks” [10]. This is a standard on a more technical level that goes hand in hand with the HIPAA Security Rule and provided slight cross over to help companies find potential risks more effectively. “A HIPAA covered entity or business associate should be able to assess and implement new and evolving technologies and best practices that it determines would be reasonable and appropriate to ensure the confidentiality, integrity and availability of the ePHI it creates, receives, maintains, or transmits “ [10]. Some categories the NIST covers would be access management, business environment, governance, risk assessment and risk management strategy. There are many more as well covering all aspects of cybersecurity. One last note. “Users who have aligned their security program to the NIST Cybersecurity Framework should not assume that by so doing they are in full compliance with the Security Rule. Conversely, the HIPAA Security Rule does not require covered entities to integrate the Cybersecurity Framework into their security management programs ” [10]. Healthcare still needs to be careful because compliance with one standard does not mean compliance with all other standards especially if there are common rules in both.

HIPAA also has an enforcement policy if any privacy or security policy is violated.”Enforcement Rules to implement statutory amendments under the Health Information Technology for Economic and Clinical Health Act (‘‘the HITECH Act’’ or ‘‘the Act’’) to strengthen the privacy and security protection for individuals’ health information” [11]. HIPAA uses the enforcement to lower the amount of violations that occur, since there are consequences in place to handle violators. One example is how software can track who has access to information at a certain time. This means it can be tracked and if it is noted that someone violated the privacy or security rules while doing so. They can use the enforcement rule to instill consequences on the person who did. The enforcement rule is constantly updating and affects all other rules to better interaction between employees and patients. The privacy and security rules keep the patients interest, while the enforcement monitors the employees in the healthcare field.

Making sure everyone is benefited that has some form of interaction with the healthcare system. Without these set guidelines it would be very hard to maintain a well functioning healthcare system with their employees and patients in mind. These are just a few reasons why the healthcare field relies so heavy on well built software systems and due to most impacts being beneficial healthcare will continue to dive deep when it comes to improving their software systems.

CURRENT AND FUTURE IMPACT

Software engineering has been expanding rather quickly in the healthcare field. The healthcare field went from a time of having paper records and stethoscopes, to electronic records and to a wearable pacemaker, stabilizing a weak heart's beat. In today's world, as software quality improves and society continues to push technology forward, there are less and less devices and systems in the healthcare field that are not controlled by some type of software [12]. With all these advances in software in the healthcare field, where does software engineering stand in the healthcare field today? Where is software engineering headed in the future of healthcare systems? Software engineers need to understand the present and future impacts in order to produce more reliable software.

Today in health care, medical software is in a state of convergence, where software engineers, are in the process of combining medical software with non-medical software [13]. This convergence is the integration of medical devices and IoT, which has a few problematic issues. "Medical software falls under federally required quality and design controls that apply to firmware, standalone software applications, software for installation in general-purpose computers, accessories to medical devices containing software, and commercial off-the-shelf software used in medical devices," [13]. Basically, every device or system involved in the healthcare field that utilizes some sort of software, is regulated under specific quality measures. The issue lies in the connection to the external devices because external devices that are outside the health care system may not necessarily abide by the same standards as the devices in the healthcare system [13]. If a patient had a phone application that monitored the status of their attached pacemaker, the patient wouldn't want an application malfunction that would result in their death. It is a software engineers responsibility to ensure quality measures are applied across the board.

In 2016, a study was done that found there to be approximately 251,454 deaths that were related to medical errors in the United States and "nonfatal medical errors is between $17 billion and $19 billion each year" [14]. This is an issue that may be considered the driving force for the convergence and integration of medical software in the healthcare field. "It has given rise to a comprehensive socio-technical model for managing healthcare through software solutions" [14]. Though with the rise in software, in the healthcare field, the question arises, can we trust the software? "Unregulated medical devices rarely find their way to patients, the same cannot be said about the largely unregulated market for health applications and software" [14]. This is a real issue in healthcare field that may often be forgotten about. The healthcare devices may be regulated and have regulated embedded software to help ensure the safety of the patients, but what about the healthcare software that actually manages the healthcare system, such as processes and procedures? These too need to have more strict regulations to improve patient safety and also promote a higher level of efficiency of health care staff, which may in turn reduce the overall cost to the patients[14].

Software engineering may have made many advances over the years in that allowed for expansion of the healthcare system, but the United States is still behind on the adoption rate of in health information technology [15]. This is primarily due to financial issues in the healthcare field in general and the type of government [15]. Since the United States does not have socialized medicine in the United States, there is not a single, unified health care system. Now the United States does not need to necessarily have a single health care system, but it is currently in a state where the system is not unified. Why would a for-profit hospital want to spend their money on providing a global system of information that can be shared between hospitals? "A doctor in an emergency department still faces the problem of having no information about the past history of his patient other than what the patient was able to tell," [13]. This quote summarizes the current state of software engineering in software. There are many hospitals throughout the United States, but if a person travels and gets into an accident, the hospital treating the patient may not be able to determine the proper treatment for the patient in an immediate situation due to not having all patient records on file.

So where is software engineering headed and what is the future impact of software engineering in the healthcare field? As discussed previously, software engineering is currently in a state of convergence and at the point where software engineers are beginning to merge the healthcare software with the software of the outside world [13]. Software engineering should be pushing for a time where healthcare software is integrated to be able to communicate across the United States and possible all around the world. The United States could possibly be at a point where it doesn't matter which hospital a patient is at , and the doctors will be able to know the patient's medical history, which could potentially save many lives [15].

In the future development of healthcare software, software engineer’s will continue to build upon and improve trustworthy software models and frameworks because “There is a need to formulate a healthcare software model that can accurately propagate trustworthiness throughout the process” [14]. This can be done by enhancing: security, efficiency, safety, functionality, reliability, regulation, validity and accuracy, which are the main attributes of a trustworthy software system [14]. A key factor in improving the software development process in healthcare depends on the continued use of SPI (Software Process improvement). “(SPI) framework for the medical device industry acts as a key enabler of best practice for the healthcare sector” [14]. With a solid software improvement model, the medical software industry can only continue to improve the software development process by continuing to resolve issues as they come, which is done by assessing the process through continuous iteration of each phase of the software development cycle [14].

Software engineer’s still have a lot of work to do in the healthcare field to refine the process in order to make this dream happen. Software engineers in the healthcare field need to strive to learn more about the healthcare industry to better understand the risks that are involved. Software engineers can always do various testing to reduce the number of bugs in a software system, but can never limit them completely [13]. “Medical software demands the establishment of its own best practices and management strategies within hospitals,” [13]. As long as software engineers continue to develop and refine the software development practices and develop a stronger knowledge base of the type of healthcare system the engineers are developing, software engineering will continue to launch the healthcare industry into the future.

SOFTWARE DEVELOPMENT LIFECYCLE (SDLC) IMPACT

In engineering we rely on minds to produce solutions to problems and most of those solutions are the result of a logical process. Any successful software product or solution must satisfy the requirements of the end users and stakeholders while at the same time being free of any defects. This is a very large task to undertake because of the inherit degree of complexity involved not just in the writing of software, but in the efficient use of available resources to optimize both budget and time constraints that exist. In healthcare there is a growing need for new software products that support better patient outcomes and reduced waiting time and costs. But there are also external requirements that demand software be built with existing standards and regulations in mind.

"Businesses now operate in a global, rapidly changing environment…Software is part of almost all business operations…"[16]. Because hospitals need to respond to competitive pressures like any other business there is a need for processes that tracks progress and plans project activities in advance. Because of this software development in the healthcare domain is created under what is called a Software Development Life Cycle (SDLC) framework. Generic process frameworks that are typically used in software development include the waterfall model, incremental development, integration and configuration and rapid software development otherwise known as agile.

Johannessen, Liv; Ellingsen, Gunnar [17] write "When traditional waterfall methods are seen as bureaucratic and slow, characterized by a pre-planned process of requirements capturing, analyzing, designing, coding, and testing, agile methods are seen as the opposite.". They go on to write that during the development of a new laboratory service ordering system from Well

Diagnostics early requirements from the users were difficult to extract. “When Well Diagnostics started to build a system for electronic requisitions from scratch, they faced many unknown factors. Initially they had some ideas for the technological concept, but the developers did not know the work practices of the users well enough to be able to develop the system on their own. They had also experienced earlier that users were not usually able to formulate their needs and requirements well enough. Well Diagnostics therefore saw an agile approach as a way to make up for these shortcomings” [17].

It is important to mention that no software process will perfectly match every project as requirements and outcome expectations are different for each project; the SDLC or any framework can be adapted to meet those requirements and outcome expectations. Because of the difficulty in extracting requirements can vary significantly, as in the separate case of the acquisition and purchase of software to assist in the Home Health care portion of a hospital. McMurtrey writes “To that end, we met with the various stakeholders (i.e., the Director of the Home Care facility and potential end-users) to map out the requirements needed from the new system. Copious notes were taken at these meetings, and a conscientious effort to synthesize our recollections was done. Afterwards, the requirements were collated into a spreadsheet for ease of inspection ” [18]. The final software package would be purchased from a set of three vendors.

In contrast to the electronic lab service software, the home health software acquisition project was able to document and define the requirements gathered by management and other stakeholders. The home health project as previously stated opted for a more agile approach, despite having access to the user group of the software to be developed, the designer of the Well Diagnostics project stated that it was “*impossible for them to communicate their needs in technical terms as it is for us to transform the technology into their needs. So that is why this [the* *agile approach] is an efficient way of cooperating. If you deliver software frequently you get corrections all the way to get you onto the right track".*

Another instance of the SDLC being adapted to achieve a set of functional requirements is when medical devices are designed with security at the beginning of a new project. Fernandes, Avelet; Pai, Anusha; Colaco, Mesquita recognize the need for security that conforms to existing standards in healthcare; *"*Health data is highly sensitive and any system handling such data needs to be protected adequately. Specific regulations such as HIPAA (Health Insurance Portability and Accountability Act of 1996) in the United States have been created to protect such health data.*"*[19]. The alternative approach was to integrate the Open Web Application Security Project (OWASP) standards integrated throughout the development life cycle resulting in a "Secure-SDLC for IoT"[19]. Because this was an application for healthcare compliance requirements were identified by both HIPAA and the Payment Card Industry (PIC) of the United States of America.

Designing a software application with security embedded throughout the SDLC cycle was done using security threat modeling techniques. "All the threat modelling approaches revolve around the two main methodologies STRIDE and DREAD. STRIDE method is used for threat identification while DREAD method is used for risk computation." [19]. During development, code reviews intended to "identify common coding errors which can pose security risk."[19]. These errors are recognized by: "hardcoded connection strings in the web.config file…", "comments containing passwords" & incomplete logging which assist the user or system in recognizing the source of a defect.

Another example of a framework adapted to measure the progress and processes for a medical imaging software product was to merge the best practices from the Lean principles defined by the Toyota Motor Company, and Agile frameworks. Manjunath, Jagadeesh & Yogeesh describe Lean as such: “Lean is a production practice and the philosophies say to eliminate the waste and increase the value to the customers. It defines 7 principles in total, each one identifies the opportunity to improve the product quality, tries to reduce the risk through early feedback, removal of wastages and increasing the cost factors” [20]. They further highlight the differences between waterfall model and agile being that code developed under the waterfall model is “released with minimal Integration test - code contains a lot of bugs”[20]. To overcome this the developers adopted continuous integration during the testing of the software; once a requirement is completed it is handed over to quality assurance (QA) for testing and feedback is given to the developers.

A strong argument is made for adopting the previously mentioned framework & Lean principles in the development of software for the healthcare industry based on figures comparing the number of defects found in the same software project developed using the waterfall model. They write: “… when compared to waterfall where the requirement is completely implemented and then it is given for black box testing…Total number of defects found in all phases of testing is 2788 in traditional waterfall model and with iterative development. And the number is reduced to 1271 in Iterative + Agile.”[20]. Although the previous projects examined skew towards favoring the agile software process (with some modifications added); this is not to say the waterfall is obsolete.

As previously mentioned in McMurtrey’s review of the SDLC process used in the purchase and acquisition of software for a healthcare facility; requirements were determined defined in the same phase. The design & development phases were skipped in this case study because both were already carried out by the vendors of the software. Implementation of the

Home Health care unit "…utilized the Parallel Installation method for approximately 60 days before the "go live" date."[20]. Hospital staff would enter patient data into both the old system and the new system to minimize "disruption" according to the Director of the Home Health Care unit. Training occurred in two phases with approximately half of the staff trained one week and the rest the following week. This method of delivery of the end product could be beneficial in cases where stakeholder's needs are clearly defined. But in other cases such as the electronic laboratory services ordering system[17], the medical imaging application[20], an iterative approach to delivery may be preferred by both the developers and the customer.

Typically, in agile software development emphasis is placed on delivering working software to the customer as frequently. Four to six-week iterations were used in the development of the medical imaging application where a "mini-release" was done at the end of the iteration period. This approach allowed for consistent and on time delivery of working software throughout the SDLC. Well Diagnostics followed a similar approach but released the first solution to their customer after four months into the project. "The developers at Well Diagnostics worked in three-week iterations. The first step in an iteration was to collect user stories and estimate the work involved in fulfilling them."[17].

CASE STUDIES

The Therac-25 Failure

One of the more famous cases in the history of medical technology failure involves the Therac-25, a radiotherapy machine made by Atomic Energy of Canada Limited (AECL) in 1982. This machine was designed to treat cancer using ionization therapy. However, Several incidents had occurred with this model. "Between June 1985 and January 1987, six known accidents involving massive overdoses by the Therac-25 - with resultant deaths and serious injuries [1 pp.18].” There were a series of causes for this issue, including lack of mechanical backups, a poorly constructed designed development process, poor programming, and how the issues were handled once they were found.

The leading cause of the accidents was the Therac-25 relying more on software than its predecessors. The prior model, Therac-20, had “independent productive circuits for monitoring electron-beam scanning, plus mechanical interlocks for policing the machine and ensuring safe operations [1 pp.20]”. These safety features were removed for development of the Therac-25, with it relying on cheaper software checks instead. These checks were found to be less reliable: many of the Therac-25 failures would not have been possible on the Therac-20. This leads to one of the key engineering aspects for medical and other life-critical systems: knowing when not to rely on hardware, and to apply mechanical backups whenever possible.

The second cause was a poorly constructed development and design process. Leveson and Turner described several of the failures in detail [1 pp.20-21], including: having a single developer, limited documentation, minimal testing that lacked a test plan. The lack of a second developer prevented peer review and pair-programming, allowing for errors that would have otherwise been caught. The documentation issue made finding the cause and resolving the errors more difficult in the long run. Additionally, the lack of good testing allowed several bugs to go through which would have otherwise been caught. These issues highlight why the entire development process is critical to good programming. The code had passed the tests (verification), however the tests were not made to cover all the needs of the software (validation). Proper test plans, documentation, and development scope ensures the code is fully validated: that it fulfills all it’s stakeholder needs and runs in a real-world environment.

The third cause was poor software design. Research into the software design found that it "allows concurrent access to shared memory, there is no real synchronization aside from data stored in shared variables[1 pp.21]". The poor design allowed for race conditions, where setting a value from 0 to 1 and reading it immediatly after could still get a 0 back, or some other value entirely. This meant some of the safety checks would always show as "OK", even when an issue occurred. The problem also relates to the lack of proper testing: the situations where the error occurred were either not tested for, or not run enough times to catch it.

The last cause was how the issues were handled once they were found. The first known case, detailed by Leveson and Turner [1 pp.21-23], showed several points where multiple people either delayed or disregarded the issue. After a patient claimed they were burned, the technician denied the possibility. A later request to the AECL had them also respond that the Therac-25 could not cause that problem. It was only a year later, when the AECL received a lawsuit, that further investigation to the machine's operations began in earnest. This shows the criticality of the support and maintenance phase, especially in medical systems. This goes beyond the scope of the developer, on how issues are handled by the medical staff, their hospital, and the company or companies that designed and built the machines. While this would not have prevented the first issue, had it been taken more seriously the issues would have been found earlier and less patients would have been overdosed.

This case exemplifies why the software engineering process needs to be well developed for the medical field. The Therac-25 failure was not a single step, but a cascade of issues that occurred throughout the entire process line. Requirements were not properly set prior to starting. Design removed critical backup components. The coding was not well designed, and testing far too limited in both scope and amount. Support and maintenance were minimal, and lacked proper handling of the accidents. Because of the lack of a proper development process, several people were killed or seriously injured.

Medtronic Pacemaker Recall

Cardiovascular devices, including defibrillators and pacemakers, are an important part of life-saving medical technology. They also include the highest recall rates for hardware and software issues. Alemzadeh et al.'s report [21 pp.22] shows the statistics from FDA reports over the course of 6 years, from January 2006 to December 2011: 586,079 defibrillators, 40164 pacemakers, and 38,394 arrhythmia detectors were recalled. Due to system failures, there were 376 dead and 17,662 injuries from these devices. These devices are the largest recall issues to-date, with "73.8 percent (31/42) of class I [high risk of severe injury or death] recalls were for cardiovascular and general hospital devices, such as defibrillators, patient monitors, and infusion pumps [21 pp.21]." This makes the process of developing, building, and monitoring issues for Cardiovascular devices as one of the most life-critical in the medical field. One specific case, a recent pacemaker recall by Medtronic in January of 2019, highlights why building a thorough testing process is important, the cost when a flaw is found in the field, and the life-saving benefits when the rest design and maintenance phases are done right.

Building a thorough testing process is a critical part of the software engineering in the medical field. Similar to the Therac-25 situation, the Medtronic pacemakers required a rare series of events for a circuit error to occur. In the pacemaker's case: "When programmed to a dual chamber mode with atrial-sensing … a unique combination of events must take place while the device is processing an ateril-sensor event. If this error occurs, the device will be unable to provide pacing until a ventricular-sensor event (VS) is detected [22]". To wit: the device had to be in a certain mode, with a certain sensor running a process, and a rare sequence of events occurring at that exact time, and it still wouldn't show an issue unless there was also problems with a ventricular-sensor or the patient also had ventricular pacing issues. This scenario shows both the importance of, and complexity to, building a thorough testing process. While testing for every potential hardware failure is impossible, devices should have as many checks and safety features as possible. Hardware, software, and all the possible ways they interact must be tested as fully as possible to ensure the patient's safety and health.

The cost of this issue not being found before release was staggering. While the scope of this issue only affected a small portion of the devices so far, which "occurred in three (3) devices from a total of 156,957 devices sold worldwide [22]," a similar failure could still occur with the rest of these devices. This meant a high risk for further injury and a potential for loss of life. These devices were "distributed worldwide between 10 March 2017 and 7 January 2019 [22]." This meant Medtronic had to go through a global suspension and recall process for the unused device, and develop further processes for the safest way to adjust or replace the devices already in use. To redistributing these recalled devices, they would need to develop a software update to fix the firmware issue, test it, get regulatory approval, and finally update the devices before. While an exact dollar amount was not mentioned in the report, the details included would guarantee a hefty reduction in profits or overall loss for Medtronic.

Medtronic's actions also show why the design and maintenance phases are as important, if not more so, then the software development and testing phases. With both the Therac-25 radiotherapy machine (described in the prior case study) and Medtronic pacemakers had failures, the quality control and handling of reported issues for Medtronic was far superior. Less than 2 thousandths of a percent of the pacemakers failed within the 2 years since their initial release. Within those 2 years, despite only having 3 devices fail, Medtronic was able accurately find the issue, work out an ideal solution, and develop a recommended process for handling already installed pacemakers until the ideal solution could be fully developed and implemented. This was in stark contrast to the Therac-25, who’s first response was equivalent to “that can’t happen”, and lawsuits going into effect before any problem solving began. This shows that Medtronc’s original design process was good enough to narrow down the issue once found, and the maintenance phase good enough to both spot the problem early on and then fix it.

These events help to show the importance of the complete software engineering process and requiring high standards for development in the medical field. Despite the critical nature of these life-saving devices, “The FDA approved almost all those devices under a medium level of regulatory controls (510(k) clearance).[21 pp.21]”. For Medtronic, a single point of failure in development and testing caused a massive recall. However, their efforts in design, development, and testing meant there were no larger scale issues and no loss of life.

REFERENCES

[1] N. G. Leveson and C. S. Turner, “An investigation of the Therac-25 accidents,” Computer, vol. 26, no. 7, pp. 18–41, Jul. 1993

[2] [5]D. Gotterbarn, K. Miller and S. Rogerson, "Software Engineering Code of Ethics", 1997. [Online]. Available: https://dl.acm.org/citation.cfm?id=265699. [Accessed: 13- Apr- 2019].

[3] "General Principles of Software Validation; Final Guidance for Industry and FDA Staff", *Fda.gov*, 2002. [Online]. Available: https://www.fda.gov/downloads/medicaldevices/.../ucm085371.pdf. [Accessed: 11- Apr- 2019].

[4] J. Poore and C. Trammell, "Application of Statistical Science to Testing and Evaluating Software Intensive Systems," in *Statistics, testing, and defense acquisition*, Washington, D.C.: National Academy Press, 1999, pp. 124-170.

[5] S. Prowell and J. Poore, "Sequence-based software specification of deterministic systems", Software: Practice and Experience, vol. 28, no. 3, pp. 329-345, 1998. Available: https://www.researchgate.net/publication/297407025_Sequence-based_software_specification_of_deterministic_systems.

[6] F. McCaffery, V. Casey, G. Coleman and M. Sivakumar, "Medical Device Software Traceability", in Software and Systems Traceability, J. Huang, A. Zisman and O. Gotel, Ed. London: Springer, 2012.

[7]https://www.hhs.gov/sites/default/files/ocr/privacy/hipaa/administrative/combined/hipaa-simplification-201303.pdf. [Accessed: 11-Apr-2019]

[8] HHS Office of the Secretary,Office for Civil Rights and Ocr, “Summary of the HIPAA Security Rule,” HHS.gov, 26-Jul-2013. [Online]. Available: https://www.hhs.gov/hipaa/for-professionals/security/laws-regulations/index.html. [Accessed: 11-Apr-2019].

[9] HHS Office of the Secretary,Office for Civil Rights and Ocr, “Summary of the HIPAA Privacy Rule,” HHS.gov, 26-Jul-2013. [Online]. Available: https://www.hhs.gov/hipaa/for-professionals/privacy/laws-regulations/index.html. [Accessed: 11-Apr-2019].

[10]https://www.hhs.gov/sites/default/files/nist-csf-to-hipaa-security-rule-crosswalk-02-22-2016-final.pdf [Accessed: 11-Apr-2019]

[11]https://www.govinfo.gov/content/pkg/FR-2013-01-25/pdf/2013-01073.pdf. [Accessed: 11-Apr-2019].

[12] Majchrowski, B. (2010). Medical software's increasing impact on healthcare and technology management. Biomedical Instrumentation & Technology, 44(1), 70-74. [Online]. Available:

http://login.ezproxy1.lib.asu.edu/login?url=https://search-proquest-com.ezproxy1.lib.asu.edu/docview/743281452?accountid=4485. [Accessed: 07-Apr-2019].

[13]Majchrowski, B. (2010). Medical software's increasing impact on healthcare and technology management. Biomedical Instrumentation & Technology, 44(1), 70-74. [Online]. Available: http://login.ezproxy1.lib.asu.edu/login?url=https://search-proquest-com.ezproxy1.lib.asu.edu/docview/743281452?accountid=4485. [Accessed: 07-Apr-2019].

[14] R. M. Abbas, N. Carroll, I. Richardson, and S. Beecham, "The Need for Trustworthiness Models in Healthcare Software Solutions," *Proceedings of the 10th International Joint Conference on Biomedical Engineering Systems and Technologies*, pp. 451–455, 2017. [Online]. Available: doi: 10.5220/0006249904510456**.** [Accessed: 10-Apr-2019].

[15] W. Ke and Z. Liu, "Software Engineering in Public Health: Opportunities and Challenges," 2012 International Conference on Computer Distributed Control and Intelligent Environmental Monitoring, Hunan, 2012, pp. 630-637. [Online]. Available: doi: 10.1109/CDCIEM.2012.155

[16] I. Sommerville, Software Engineering, 5th ed. Old Tappan: Pearson Education UK, 2006, p. 73.

[17] L. K. Johannessen and G. Ellingsen, "Integration and Generification—Agile Software Development in the Healthcare Market," Computer Supported Cooperative Work (CSCW), vol.

18, no. 5-6, pp. 607–634, 2009. [Online]. Available: https://link-springer-com.ezproxy1.lib.asu.edu/article/10.1007/s10606-009-9097-8. [Accessed Mar. 30, 2019].

[18] M. Mcmurtrey, “A Case Study of the Application of the Systems Development Life Cycle (SDLC) in 21st Century Health Care: Something Old, Something New?,” Journal of the Southern Association for Information Systems, vol. 1, no. 1, 2013. [Online]. Available: https://quod.lib.umich.edu/j/jsais/11880084.0001.103?view=text;rgn=main. [Accessed Mar. 30, 2019].

[19] A. M. Fernandes, A. Pai, and L. M. M. Colaco, “Secure SDLC for IoT Based Health Monitor,” 2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA), 2018.

[20] K. N. Manjunath, J. Jagadeesh, and M. Yogeesh, “Achieving quality product in a long-term software product development in healthcare application using Lean and Agile principles: Software engineering and software development,” 2013 International Mutli-Conference on Automation, Computing, Communication, Control and Compressed Sensing (iMac4s), 2013. [Online]. Available: https://arizona-asu.userservices.exlibrisgroup.com/view/action/uresolver.do?operation=resolveService&package_service_id=16904600430003841&institutionId=3841&customerId=3840. [Accessed Mar. 30, 2019].

[21] H. Alemzadeh, R. K. Iyer, Z. Kalbarczyk, and J. Raman, “Analysis of Safety-Critical Computer Failures in Medical Devices,” *IEEE Security & Privacy*, vol. 11, no. 4, pp. 14–26, 2013.

[22] Medtronic and Chris Harrold, “Dual Chamber Pacemakers: Urgent Medical Device Recall,” *URGENT MEDICAL DEVICE RECALL*, Jan-2019. [Online]. Available: https://www.medtronic.com/us-en/healthcare-professionals/products/product-performance/dual-chamber-pacemaker-recall.html. [Accessed: 07-Apr-2019].