

## Form

### Application for Reappointment, Tenure, and/or Promotion

[Approved by the Faculty Senate 4/98. Revised 3/04; 08/11; 03/15; 6/17; 5/19]

Procedures for the RTP process will be governed by pertinent sections in Chapter IV in the Faculty Handbook.

Name of RTP candidate: Lucas Layman

Department: Computer Science

Personnel action applied for: Reappointment at the rank of Assistant Professor

Effective date: Aug 2017–May 2021

End of current contract period: May 2021

End of current academic year: 2020–2021

#### I. Academic Status at UNCW

Present rank: Assistant Professor

Effective date: August 7, 2017

Previous rank(s) and date(s) at UNCW: None

Current employment status: Nontenured, fourth year of initial five-year appointment–reappointment application deferred one year for Family Medical Leave.

#### II. Education

Institution	Concentration	Dates	Degree
North Carolina State University	Computer Science	2004-2009	Ph.D.
North Carolina State University	Computer Science	2002-2004	M.S.
Loyola College (now Loyola University)	Computer Science	1998-2002	B.S.

#### III. Professional History (other than UNCW)

Position/Rank	Institution	Dates
Senior Research Scientist	Fraunhofer USA, Inc.	2009-2017
Research Associate	National Research Council Canada	2009

#### IV. Contributions to Teaching

##### A. Required subcategories:

1. Courses taught (a non-chronological list of course numbers and titles)

Course number	Title	Term(s)
CSC 231	Introduction to Data Structures	SP21, FA20, SP20, SP19, SP18
CSC 242	Computer Organization	FA18, SP18, FA17
CSC 315	Mobile Applications Development	SP21, FA20, SP20, SP19, FA18
CSC 350	Secure Programming	SP21
CSC 475/592	Engineering Secure Software	SP19

2. Sample course materials. Identify the materials you have uploaded on the secure RTP electronic website (e.g., course syllabus, tests, modules).

[Text in blue are links to Supporting Materials](#)

##### **CSC 231 – Introduction to Data Structures**

- [Syllabus](#)
- Module – ArrayList data structure: [ArrayList worksheet](#), [slides](#)

##### **CSC 242 – Computer Organization**

- [Syllabus](#)

3. Summary of student evaluations since last personnel action at UNCW:
4. Peer evaluations of teaching since last personnel action at UNCW. Include the name of the evaluator, rank/position, course observed, and date. (Upload a copy of all peer evaluations and any appended responses by the candidate to the designated RTP electronic website).
  - Assoc. Prof. Devon Simmonds, CSC 242 – Computer Organization, 16 October 2017 - [Link to Supporting Materials](#)
5. Reflective analysis of teaching strengths and actions taken to implement improvements. (Limit to a maximum of three examples and no more than one page in total.)

**My biggest strength is that I listen to the students.** When they are confused or disengaged with the material, that is because I have not communicated well, so I ask questions and go over the topic again. When the same quiz question is answered incorrectly many times, the problem is most likely with the question or my delivery. If students are not coming to me with questions, I have not motivated them enough. If students offer feedback that an assignment was kind of boring, then I need to revise my assignment. I care deeply about the students' success, and the best way to help them is to listen to their issues and adjust to their needs. Students are good at detecting and, usually, reciprocating the instructor's effort. I do my best to be open and inviting to the students, but many remain reluctant to ask for help. I have implemented a few practices to open other channels of feedback and communication:

Term	Course (Section)	Adjusted Averages			Raw Scores		
		Summary Evaluation	Progress on Relevant Objectives	Ratings of Summative Questions	Summary Evaluation	Progress on Relevant Objectives	Ratings of Summative Questions
Fall 2020	CSC 231 (801)/ CSC 231 (803)	4.6	4.5	4.6	4.5	4.3	4.6
	CSC 315 (801)	4.5	4.3	4.7	4.5	4.3	4.7
Summer 2020	-	-	-	-	-	-	-
Fall 2019	-	-	-	-	-	-	-
Summer 2019	-	-	-	-	-	-	-
Spring 2019	CSC 231 (3)	4.9	4.9	4.9	4.9	4.9	4.9
	CSC 315 (1)	4.6	4.5	4.7	4.6	4.5	4.7
	CSC 475 (1)	4.8	4.8	4.7	4.3	4.3	4.3
Summer 2018	-	-	-	-	-	-	-
Spring 2018	CSC 231 (1)	4.5	4.3	4.7	4.5	4.3	4.7
	CSC 242 (1)	4.6	4.4	4.7	4.6	4.4	4.7
	CSC 242 (2)	4.4	4.9	4.8	4.3	3.9	4.7
Fall 2017	CSC 242 (1)	4.6	4.5	4.6	4.5	4.4	4.6
	CSC 242 (2)	4.8	4.7	4.9	4.8	4.7	4.9

Term	Course (Section)	Adjusted Averages			Raw Scores		
		Combined Averages of Summative Ratings	Overall, I rate this teacher an excellent teacher	Overall, I rate this course as excellent	Combined Averages of Summative Ratings	Overall, I rate this teacher an excellent teacher	Overall, I rate this course as excellent
Spring 2020	CSC 231 (3)	4.9	4.9	4.9	4.9	4.9	4.9
	CSC 231 (5)	4.8	4.9	4.8	4.8	4.9	4.7
	CSC 315 (1)	5.0	5.0	5.0	4.8	4.8	4.7
Fall 2018	CSC 242 (1)	4.9	5.0	4.7	4.9	5.0	4.7
	CSC 242 (2)	4.7	4.9	4.5	4.7	4.9	4.5
	CSC 315 (1)	4.9	4.9	4.9	4.9	4.9	4.9

1. In CSC 231 – Introduction to Data Structures, most programming assignments and quizzes are **automatically graded by testing programs** that I give to the students. The test programs help the students debug their code, and also produce scores so they know where they stand. The test programs help get “smaller” questions out of the way, and thus the questions I get on these assignments are more conceptual or logically challenging.
2. I pass out index cards and ask students to write down *anonymously* the **“muddiest point”** – something they are unclear on from the course in general. I review and sort the cards and start off the next class discussing the most common topics. The students see that other people are confused on a topic, and I get to address a learning gap. Students react very positively to this exercise, and probably no other activity builds rapport with them as strongly in the classroom.

3. To facilitate the move to online learning, I invite all my students to join a **Slack workspace** (<https://slack.com>) with channels for individual classes. The synchronous chatting, instead of email or Zoom, has become the de facto way of communicating in my online classes. We are able to chat, share files, screenshots, and generally collaborate much more easily than via email and without perceived social pressures of a Zoom meeting. Students ping me via Slack when working on assignments at odd hours, and I often respond, which is not something either of us would do via Zoom. In general, the tool has helped me keep more aware of students' progress, while also offering the students a personal synchronous communication avenue to me. In my Fall 2020 course evaluations, many students mentioned Slack as a positive means to "stay connected" to the instructor.
6. Academic advising within the department (Include the semester and # of undergraduate and graduate advisees.

Semester and Year	Undergraduate Advisees	Graduate Advisees
Spring 2021	33	1
Fall 2020	33	1
Spring 2020	28	2
Fall 2019	FMLA	FMLA
Spring 2019	23	2
Fall 2018	23	2

B. Optional subcategories

1. Bulleted list of courses developed/revised/new to the individual or to the university
  - CSC 231 – Introduction to Data Structures (individual)
  - CSC 242 – Computer Organization (individual)
  - CSC 315 – Mobile Applications Development (individual)
  - CSC 350 – Secure Programming (university)
  - CSC 475/592 – Engineering Secure Software (university)
2. Bulleted list of theses, dissertations, and DIS supervised. Include the title of thesis/dissertation/DIS, student name, date completed, your role on committees (e.g., chair, member)
  - **CSC 594 Master's Capstone Chair**, Imprnt: A Cross-Platform Mobile Application for Personality-Based Pet Adoption, Kinsley Sigmund, in progress.
  - **GGY 499 Honors Thesis Committee Member**, Using Geospatial Technologies to

3. List and describe your three (3) best examples of special initiatives/incentives that enhance student learning. Limit each description to no more than 3 - 4 sentences.

**"Brain Games"** in CSC 231 (Introduction to Data Structures) are in-class, 20-minute, multi-question quizzes that students complete in pairs. Pairs are assigned randomly, and the questions are challenging but performance does not negatively impact their grade. Individuals accumulate points throughout the semester on a leaderboard, and students are awarded extra credit based on their overall score at the end of the semester. The activity yields very lively discussion, peer education, builds camaraderie, and provides me with valuable, no-stakes insight into students' strengths and weaknesses.

**"The Sorting Sleuth"** in CSC 231 (Introduction to Data Structures) is a program that students use to run experiments to determine the hidden identities of five different sorting algorithms. The students write a report that presents the experimental evidence and justifies their determination of which sorting algorithm is which. The assignment is a break from programming assignments but requires a deep understanding of the subject matter. The assignment can be completed in pairs and requires the students to exercise both their writing and critical reasoning skills in a different way from what is often required in early CSC courses.

I use **externally sponsored term projects** in CSC 315 (Mobile Application Development) wherein I solicit mobile app ideas from the UNCW community (and the students themselves) that students create for the bulk of their course grade. The project sponsor(s) make a presentation to the class and answer questions, and students are allowed to choose from among the sponsored project(s) or their own idea. Outside project ideas implemented by the students include an app to navigate and learn about ecosystems in Carolina Beach State Park from Drs. Taylor and Kubasko from Watson College of Education, an app that matches students with study abroad locations from the Office of International Programs, and a shooting range shot timer using microphone input for Dr. Tagliarini in Computer Science. I find that students are more motivated to work on a project for a third party, or for themselves, than on a concocted-for-the-course project created by the instructor.

4. Bulleted list of efforts to improve teaching effectiveness, evidence of self-learning, and evidence of commitment to fostering the intellectual development of students through:
  - attendance at professional meetings or sessions primarily devoted to teaching (In a bulleted list, highlight up to five (5) of the most recent or most important meetings/sessions you have attended)
    - ACMSE 2020: The Annual ACM Southeast Conference, Virtual due to COVID, April 3-4, 2020.
  - completion of continuing education, workshops, symposia, or other specialized training primarily devoted to teaching
    - Webinar - Best Practices in Online Assessment UNCW Distance Education and eLearning March 17, 2020
    - Webinar - Let's get real: authentic online assessments UNCW Distance Education and eLearning March 17, 2020
    - Webinar - Spring Semester 2.0: Revising Your Syllabus and Policies for New Circumstances UNCW CTE and DEeL March 16, 2020

- Workshop - 2018 Applied Learning Summer Institute ETEAL July 25, 2018 - July 26, 2018

5. Bulleted list of grants and/or fellowships related to teaching at all levels including K-12 (Include title of grant/fellowship, granting agency, dates, amount, list of investigators (designate PI or co-PI))

- **Submitted to funder**, Cyber-Seahawk (C-Hawk) CyberCorps(R) Scholarship for Service Program, National Science Foundation, \$1,747,839, U. Clarke (PI), G. Stoker (Co-PI), R. Vetter (Co-PI), S. Mittal (Co-PI), M. Modares-nezhad (Co-PI), L. Patterson (Co-PI), **L. Layman (Co-PI)**, and J. Cummings (Co-PI).
- **Funded**, Telemetry for Learning Analytics in Programming-Intensive Courses, UNCW SURCA, June-August 2020, \$5,000, **Lucas Layman (PI)** and Aaron Csetter (Student PI).
- **Funded**, A Safe Environment for Teaching Computer Security in an Adversarial Setting, UNCW CTE, June-July 2019, \$3,000, **Lucas Layman (PI)**.
- **Funded**, Leveraging Applied Learning in CSC 131: A Department-wide Initiative Impacting 400 Students Per Year, UNCW Applied Learning Strategic Initiatives, Song, Y. (Principal), Ferner, C. S. (Co-Principal), Pence, T. B. (Co-Principal), Ebrahimi, E. (Co-Principal), **Layman, L. M. (Co-Principal)**, Morago, B. A. (Co-Principal), Narayan, S. (Co-Principal), Tompkins, J. A. (Co-Principal), Stoker, G. M. (Co-Principal), Internal Grants/Funding, "Leveraging Applied Learning in CSC 131: A Department-wide Initiative Impacting 400 Students Per Year", Applied Learning Strategic Initiatives, University of North Carolina Wilmington, \$30,000.00. (sub: March 15, 2019).
- **Funded**, Implementing CSC 315 – Application Development for Mobile Devices, UNCW CAS Curriculum Initiative, June-August 2018, \$3,500, **Lucas Layman (PI)**

6. Bulleted list of honors, listings, or awards related to teaching

- Graduating student impact recognition – every semester since Spring 2018-Present.

## V. Scholarship/Research/Artistic Activities

### A. Required Subcategories:

1. List bibliographic information for refereed publications (including juried or peer-reviewed performances , exhibits, artistic works, productions or writings).
    - a. Published
      1. L. Layman *et al.*, "Toward Predicting Success and Failure in CS2 : A Mixed-Method Analysis," in *Proceedings of the 2020 ACM Southeast Conference (ACMSE 2020)*, Tampa, FL, USA: ACM, 2020, p. 8. doi: [10.1145/3374135.3385277](https://doi.org/10.1145/3374135.3385277). [Online]. Available: <https://arxiv.org/abs/2002.11813> - [link to supporting materials](#)
      2. W. Roden and L. Layman, "Cry Wolf : Toward an Experimentation Platform and Dataset for Human Factors in Cyber Security Analysis," in *Proceedings of the 2020 ACM Southeast Conference (ACMSE 2020)*, Tampa, FL, USA: ACM, 2020, pp. 264–267. doi: [10.1145/3374135.3385301](https://doi.org/10.1145/3374135.3385301). [Online]. Available: <https://arxiv.org/abs/2002.10530> - [link to supporting materials](#)

----- Items below are prior to joining UNCW -----

    - 3. L. Layman *et al.*, "Toward Reducing Fault Fix Time: Understanding Developer Behavior for the Design of Automated Fault Detection Tools," in *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, Madrid, Spain: IEEE, Sep. 2007, pp. 176–185. doi: [10.1109/ESEM.2007.11](https://doi.org/10.1109/ESEM.2007.11) - [link to supporting materials](#) (**Best Paper Award**)
  - b. Accepted for publication
    1. None.
  - c. Under consideration
    1. None.
2. Publications (or performances, exhibits, artistic works, productions or writings) not listed in the refereed category (e.g., abstracts, book reviews, technical reports, white papers, magazine or newspaper articles/columns.)
  - a. Published
    1. W. Roden and L. Layman, *The Cry Wolf IDS Simulator - An environment for conducting controlled experiments of cyber security analysis tasks*, 2019. [Online]. Available: <https://uncw-hfcs.github.io/ids-simulator/> (visited on 09/16/2020)
    2. F. Shull *et al.*, "Technical Debt: Showing the Way for Better Transfer of Empirical Results," in *Perspectives on the Future of Software Engineering: Essays in Honor of Dieter Rombach*, J. Münch and K. Schmid, Eds., vol. 9783642373, Elsevier, 2013, pp. 179–190. doi: [10.1007/978-3-642-37395-4\\_12](https://doi.org/10.1007/978-3-642-37395-4_12) - [link to supporting materials](#)

- b. Accepted for publication
    1. None.
  - c. Under consideration
    1. None.
  3. Research grants or research fellowships
    - a. Awarded (include authors, title, organization, amounts, duration)
      1. Dodson, C. H. (Principal), **Layman, L. M. (Co-Principal)**, Van Riper, M. (Supporting), "Refinement and Usability Testing of a Pharmacogenomics App for Dosing Guidelines for Oncology", Oncology Nursing Foundation, Private, \$25,000.00. (start: January 15, 2020, end: January 15, 2022).
      2. **Layman, L. M. (Principal)**, Internal Grants/Funding, "Anchoring Bias in the Identification of Cyber Attacks", College of Arts & Sciences Pilot Grant Award Fall 2019, University of North Carolina Wilmington, \$3,500.00. (start: May 2020, end: June 2020).

----- Items below are prior to joining UNCW -----

    - 3. **Layman, L. (Principal)**, Diep, M. (Co-Principal), External Grants/Sponsored Research, "Data Protection Policy Effectiveness Measures", Cisco University Research Program Fund, \$100,000.00, Private. (start: December 2017, end: July 2017)
  - b. Applied for (include dates and status: pending or not funded)
    1. **Currently Under Review**, Dodson, C. H. (Principal), **Layman, L. M. (Co-Investigator)**, Carroll, R. M. (Co-Investigator), External Grants/Sponsored Research, "PHS 2019-02 Omnibus Solicitation of the NIH for Small Business Technology Transfer Grant Applications (Parent STTR [R41/R42] Clinical Trial Not Allowed: Refinement and Usability Testing of a Pharmacogenomics App for Dosing Guidelines for Pain Management Treatment in the Oncology Field", National Institute of Nursing Research, Federal, \$148,926.00. (sub: April 5, 2020).
4. Grants or research fellowships for off-campus study or professional development
  - a. Awarded (include authors, title, organization, amounts, duration)
    - None.
  - b. Applied for (include dates and status: pending or not funded)
    - None.
5. Presentations (including readings, lectures, posters) at professional meetings. (Please indicate if refereed)
  1. **Refereed**, "Cry Wolf: Toward an Experimentation Platform and Dataset for Human Factors in Cyber Security Analysis", ACMSE 2020: The Annual ACM



Southeast Conference, April 4, 2020.

6. In-progress scholarship/research/ artistic activities (list up to 3)
  - Primacy Effect and Availability Bias in the Identification of Cyber Attacks (Ongoing)
  - Impact of IDS Confidence Estimates on Analyst Behavior in a Divded Attention Task (Planned)
- B. Optional subcategories (Include only a bulleted list)
  1. Attendance at professional meetings (List up to five meetings. Do not list individual sessions or workshops.)
    - International Conference on Software Engineering, Montreal, QC, May 27-30, 2019.
    - Attendance at conferences prior to joining UNCW is omitted.
  2. Other initiatives in professional development
    - Training - Safe Zone training UNCW LGBTQIA Office March 4, 2019
    - Training - "Green Zone" training UNCW Military Affairs December 5, 2018
  3. Professional consultancies related to research
    - a. Paid
      - Fraunhofer USA, Inc., Creativity Workshop Organizer, October 2020-present.

## VI. Service

- A. University (e.g., committee memberships, leadership positions, or administrative duties)
  - Community Engagement Grants, Grant Reviewer, December 3, 2018–December 4, 2018
  - Convocation Small Group Discussion Leader, August 20, 2018
- B. College or school (e.g., committee memberships, leadership positions, or administrative duties, advising clubs or campus groups, student counseling or advising other than routine work with department advisees, etc.)
  - Computer Science Department Chair Search Committee, Committee Member, November 2020–Present
- C. Department and/or Program (e.g., committee memberships, leadership positions, or administrative duties, advising clubs or campus groups, student counseling or advising other than routine work with department advisees, etc.)
  - 2021 Lecturer Search Committee, Committee Member, October 2020–Present.
  - CSC 131/231/331 Program Coordination Committee, Committee Chair/Member, August 2018–Present.
  - Faculty Search Committee - Intelligent Systems Engineering, Committee Member, September 2017–May 2018
- D. Professional (e.g., manuscript editor or editorial board member, artistic juror, grant or accreditation reviewer, advisor/leader/director in workshops or consultations, leadership in professional or scholarly societies, leadership in seminars or short courses taught to professionals in the candidate's discipline (for each activity indicate whether paid or pro bono))
  - **Grant Reviewer**, Air Force Office of Scientific Research, Trust and Influence program, March 2020, pro bono.
  - **Program Committee Member**, 2020 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - Industry Track, July 2020–August 2020, pro bono.
  - **Reviewer**, ACM Transactions on Computer Education, Journal Article June 8, 2020, pro bono.
- E. Community Consulting activities for discipline-related activities (e.g., awards, honors, boards, offices, presentations/workshops/programs, continuing education, newspaper or magazine articles for the lay public)
  - **Guest Speaker/Lecture**, Cape Fear Museum, "What's Brewing in Science" Town Hall on Cybersecurity, May 9, 2018
  - **Radio panel**, WHQR, "CoastLine: Blockchain, Beyond Bitcoin and Unpacked", June 14, 2018.

## Sample Course Materials

### CSC 231 Syllabus

## CSC-231-801-803 merged (Fall 2020)

[Jump to Today](#)

### Introduction to Data Structures, Fall 2020

**Where:** Online

**When:** Weekly modules. Materials will be posted prior to Monday 8am. Assignments are due Sunday night.

**Instructor:** [Dr. Lucas Layman \(Links to an external site.\)](#)

- **Email:** [laymanl@uncw.edu](mailto:laymanl@uncw.edu)
- **Slack channel:** [https://join.slack.com/t/drlaymansclas-7jh3938/shared\\_invite/zt-i0z8a035-EppR8j6fAfxMb92Z4JXu3A](https://join.slack.com/t/drlaymansclas-7jh3938/shared_invite/zt-i0z8a035-EppR8j6fAfxMb92Z4JXu3A) ([Links to an external site.](#))
- **Office hours:** Use Slack any time. 1-on-1 Zoom teleconferencing appoints are also available. See the Communications Policy for details.

### Tools for Online Coursework

- [Slack \(Links to an external site.\)](#) - The best way to get in touch with me, get help, and chat with your peers.
- [Zoom \(Links to an external site.\)](#) - video conferencing software for screen sharing and live meetings.
- [Canvas guides \(Links to an external site.\)](#) - for help with anything Canvas related

---

## Course description

Study of basic data structures and their applications. Lists and trees; searching and sorting algorithms; hashing; analysis and design of efficient algorithms. Recommended for CSC majors only. A grade of 'C' (2.00) or better is required for taking courses for which CSC 231 is a prerequisite.

Storing, searching, and sorting data are basic functions performed in every software system. Strategies and algorithms for organizing data are foundational to computer science, and are applicable to computing domains from systems engineering, to software engineering, to digital arts. Students will gain an understanding of the strengths and weaknesses of well-known data structures and algorithms by implementing them in Python.

**What to expect:** The class will be a mix of virtual lectures and programming activities. Each homework assignment and quiz will compound on all previous work so that you will have ample opportunity to practice and become expert in the material. This course is *programming intensive*. You must demonstrate programming competence and growth throughout the semester in order to pass this course.

- Class will *not* meet in person.

- Class will *not* be meeting synchronously online for classes. There may be rare exceptions, e.g., for quizzes, but you will be notified in advance.
- Course content will be organized in the Modules section of Canvas. Usually, you will have one module per week, and the week's materials will be posted prior to Monday. I will post an Announcement to Canvas when material is posted.
- I will upload videos on course topics to YouTube and link them from the Canvas Module for the week.
- Most weeks, you will have homework due on Sunday. *Do not* underestimate the time required to complete the homework. For everyone, there is a point in the homework where you get stuck and you need help. You want that time to be Wednesday or Thursday (or before). Complete the lectures and accompanying exercises Monday-Tuesday, and begin the homework assignment as soon as possible. Waiting until Friday or later is a recipe for failure.

**Prerequisite:** [CSC 131 \(Links to an external site.\)](#) with a grade of 'P', 'C', or better

**Corequisite:** [CSC 133 \(Links to an external site.\)](#)

**Student learning outcomes:**

1. Students develop knowledge of basic data structures for storage and retrieval of ordered or unordered data. Data structures include: arrays, linked lists, binary trees, heaps, and hash tables.
2. Students develop knowledge of applications of data structures including the ability to implement algorithms for the creation, insertion, deletion, searching, and sorting of each data structure.
3. Students learn to analyze and compare algorithms for efficiency using Big-O notation.
4. Students implement projects requiring the implementation of the above data structures.

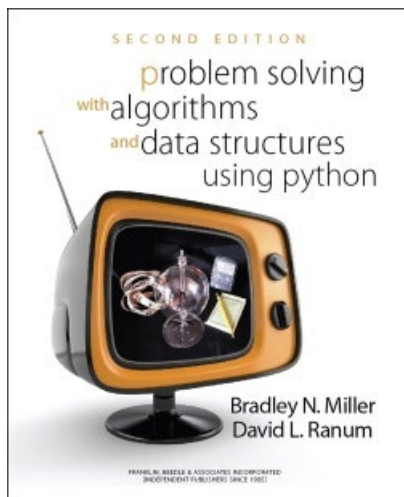
## Communication Policy

There are no dumb questions. Please ask me if you are unsure about a topic, an assignment, or a course policy, or just need help. My goal is for you to learn the material and thus to be successful in this course.

I will be available virtually to answer your questions. Here is what that will look like:

- **Join the [Slack Workspace](https://join.slack.com/t/drlaymansclas-7jh3938/shared_invite/zt-i0z8a035-EppR8j6fAfxMb92Z4JXu3A) ([Links to an external site.](https://join.slack.com/t/drlaymansclas-7jh3938/shared_invite/zt-i0z8a035-EppR8j6fAfxMb92Z4JXu3A))** ([https://join.slack.com/t/drlaymansclas-7jh3938/shared\\_invite/zt-i0z8a035-EppR8j6fAfxMb92Z4JXu3A](https://join.slack.com/t/drlaymansclas-7jh3938/shared_invite/zt-i0z8a035-EppR8j6fAfxMb92Z4JXu3A) ([Links to an external site.](https://join.slack.com/t/drlaymansclas-7jh3938/shared_invite/zt-i0z8a035-EppR8j6fAfxMb92Z4JXu3A))) to get help from me and your peers via text chat, screen sharing, and more.
- You are responsible for monitoring your UNCW email account and Canvas for course updates.
- Make sure that you are receiving notifications from Canvas when an Announcement is posted. Go to Account -> Notifications and check your settings.
- **I have a question about my grade, a personal issue, or something else private:** Contact me via Direct Message on Slack, or send me email. Be aware that the UNCW email system blocks emails containing .py attachments.
- **When will you get back to me?** I will respond within 24 hours during the work week. Friday night thru Sunday I will respond within 48 hours. I will often respond quicker.
- **How quickly will you post grades and give feedback on assignments?** Within 1 week of the assignment due date.
- **How will office hours work?** Slack is the best way to get in touch with me. Please email me or send a Slack DM if you would like to set up a 1-on-1 video chat.

## Textbook



### Problem Solving with Algorithms and Data Structures using Python

Bradley N. Miller and David L. Ranum

ISBN-13: 978-1590282571

[Free online edition](#) ([Links to an external site.](#))

[Amazon](#) ([Links to an external site.](#))

[Barnes & Noble](#) ([Links to an external site.](#))

This textbook is *required* to complete the course. The free, online edition is sufficient to complete the course. A printed text version is available from retailers if you desire.

# Course Policies

## Seahawk Respect Compact

We adhere to the [Seahawk Respect Compact \(Links to an external site.\)](#). All individuals in this class will be treated with inclusiveness, mutual respect, acceptance, and open-mindedness by all persons.

## Student Academic Honor Code

All members of UNCW's community are expected to follow the academic Honor Code. Please read the UNCW [Student Academic Honor Code \(Links to an external site.\)](#) carefully. Academic dishonesty in any form will not be tolerated in this class.

Be familiar with UNCW's position on plagiarism as outlined in the UNCW [Student Academic Honor Code \(Links to an external site.\)](#). Plagiarism is a form of academic dishonesty in which you take someone else's ideas and represent them as your own. Here are some examples of plagiarism:

1. You write about someone else's work in an assignment and do not give them credit for it by referencing them.
2. You give a presentation and use someone else's ideas and do not state that the ideas are the other person's.
3. You get facts from your textbook or some other reference material and do not reference that material.

## Collaboration, cheating, and personal proficiency

**Collaboration:** You may discuss course content with your peers. You may seek out additional resources (i.e., the Internet) to help you *understand* the course content but you may not copy code or other solutions.

*You may not share code with your peers* unless explicitly approved by the instructor including for assignments that you have completed. This means:

- Don't show anyone your code
- Don't look at anyone else's code
- Both of these statements apply to assignments you have already completed until given permission by the instructor. The other person may not have completed the assignment yet, and may copy from you.

All coursework is to be completed *individually* except when explicitly indicated by the instructor. If collaboration is permitted, the collaborative coursework must bear the names of all

collaborators on the team and all collaborators must contribute equally. Grading on collaborative assignments may be weighted by individual contribution and peer evaluation.

**Cheating:** Obtaining answers to assignments, quizzes, exams, or projects from any source other than your own brain is cheating. Any person completing work on your behalf is cheating. All coursework is to be completed individually unless explicitly stated otherwise (i.e., "collaborative"). Sharing code is cheating. Incidents of cheating will be addressed according to the policies in the [Student Academic Honor Code \(Links to an external site.\)](#). The minimum penalty for cheating is a grade of F on the assignment and an Academic Honor Code violation filed with the Dean of Students. Repeated or severe infractions will result in a grade of F for the course and an Honor Code violation.

**Personal proficiency:** You are expected to become proficient in the course content. A substantial portion of your course grade is based on timed assessments (i.e., traditional and programming quizzes). Over-reliance on peer discussion or Internet sources will lead to poor grades on these individual assessments.

### Attendance

Class does not meet in person. Class does not meet synchronously online except in rare circumstances (if ever). I will notify you if there will be a mandatory, *synchronous* online session at least one week in advance, for example for a quiz. An unexcused absence for a mandatory online session for a quiz will result in a grade of F on that assessment. Failure to take the Final Exam will result in a grade of F for the course.

### Grading

Your grades will be posted in the Grades section of Canvas. I will also make an Announcement when grades are posted.

You are graded cumulatively on each of the items below. These cumulative grades are then weighted.

- 35% – Programming assignments and other homework. Usually one per week due on Sunday.
- 45% – Quizzes. Timed assessments approximately one per week.
- 5% – Reading Quizzes. Low-pressure checks to make sure you are keeping up and to help you study.
- 15% – Final Exam

For example, suppose you receive a 10/15 on Assignment 1 and 18/20 on Assignment 2. Your cumulative assignment score would be 28/35. This score would be multiplied by 40% and added to the weighted scores for the assignments and quizzes. Your course letter grade will be determined by the sum of these weighted scores according to the scale below.

Change (11-05-2020): The assignment and quiz that detract the most from their respective cumulative grades will be dropped.

- A [94.0,∞]
- A- [90.0,94.0)
- B+ [87.0,90.0)
- B [84.0,87.0)
- B- [80.0,84.0)
- C+ [77.0,80.0)
- C [74.0,77.0)
- C- [70.0,74.0)
- D+ [67.0,70.0)
- D [64.0,67.0)
- D- [60.0,64.0)
- F [0.0,60.0)

**Homework assignments and quizzes** are due at the time and date indicated. Late submissions will receive a grade of zero. You are encouraged to submit your assignments early. There are no make-ups for assignments or quizzes that are late or incomplete for unexcused reasons (see below). Make-ups for *excused* absences will be addressed on a case-by-case basis.

**Final Exam** will be delivered on Canvas during finals week. Failure to take the Final Exam will result in a grade of F for the course.

- Exam opens: Wednesday, Dec 2 at 5pm
- Exam closes: Thursday, Dec 3 at 8pm.
- The exam will have a 3 hour time limit once you begin. The exam will terminate at the specified close time regardless of when you begin. So the latest you should start is Thursday, Dec 3 at 5pm.

**Regrading policy:** Requests to regrade must be made no later than three calendar days after the graded item is returned. For example, if a homework is returned on Monday, the request to regrade must be made no later than Thursday.

**Unexcused reasons:** The following is a partial list of *unexcused* reasons for failing to turn in graded material on time:

- I have a court date.
- My boyfriend/roommate/girlfriend and I are having problems.
- I have an appointment.
- I am going on vacation.
- I have to work.
- I have an admissions interview for another college.
- I got locked out of my apartment.



- I overslept.
- I couldn't find my car keys.
- My dog/cat/bird etc. got out.
- I couldn't get a parking spot.
- I was hungover/I was out late the night before.
- My alarm/roommate/friend did not wake me up.
- Traffic was bad.
- I was having one of those days so I went back to bed.

## University Learning Center

You are encouraged to visit the University Learning Center (ULC) if you need extra support in mathematics, writing, or managing your schedule. The ULC's mission is to help students become successful, independent learners. Tutoring at the ULC is NOT remediation: the ULC offers a different type of learning opportunity for those students who want to increase the quality of their education. ULC services are free to all UNCW students and include the following:

- [Learning Services \(Links to an external site.\)](#)
- [Math Services \(Links to an external site.\)](#)
- [Supplemental Instruction \(Links to an external site.\)](#)
- [Writing Services \(Links to an external site.\)](#)

## Students with disabilities

Students with diagnosed disabilities should contact the Office of Disability Services (910-962-7555). Please submit to me a copy of the letter you receive from Office of Disability Services detailing class accommodations you need. If you require accommodation for test-taking, make sure I have the referral letter no fewer than one week before the test.

## Violence and harassment

UNCW practices a zero tolerance policy for any kind of violent or harassing behavior. If you are experiencing an emergency of this type contact the police at 911 or UNCW CARE at 962-2273. Resources for individuals concerned with a violent or harassing situation can be located at [the UNCW Title IX website \(Links to an external site.\)](#).

## Notional Schedule

The schedule below is notional and subject to change.

ID	Date	Topic	Chapter
Week 0	19-Aug	Course Overview, PyCharm setup	
Week 1	24-Aug	131 review	1.1-1.12,
Week 2	31-Aug	Python under the Hood	
Week 3	8-Sep	Classes and Objects	1.13.1 (skip 1.13.2), 1.14-1.16

<b>ID</b>	<b>Date</b>	<b>Topic</b>	<b>Chapter</b>
Week 4	14-Sep	Analysis of Algorithms	3.1-3.10
Week 5	21-Sep	ArrayList data structure, Stacks, Queues, and Deques	4.1-4.7, 4.10-4.18
Week 6	28-Sep	Linked Lists	4.19-4.21
Week 7	5-Oct	Search Algorithms	6.1-6.4
Week 8	12-Oct	Hashing and hash tables	6.5.1-6.5.2
Week 9	19-Oct	The Map ADT	6.5.3.-6.5.4
Week 10	26-Oct	Recursion	5.1-5.5, 5.10, 5.13
Week 11	2-Nov	Sorting Algorithms	6.6-6.9, 6.11-6.13
Week 12	9-Nov	Trees, Binary Search Trees	7.1-7.3, 7.5, 7.11-7.14
Week 13	16-Nov	Binary Search Trees	
Week 14	23-Nov	???	
Week 15	30-Nov	Final exam	

## CSC 231 ArrayList Worksheet

## CSC 231- The ArrayList Data Structure

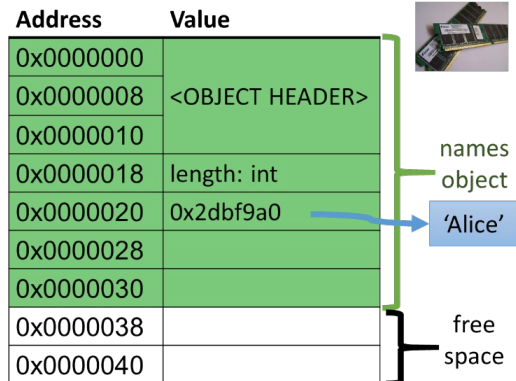
The Python list[] type implements an ArrayList data structure.

An array is a contiguous block of computer memory.

- HEADER is 24 bytes
- length is 8 bytes - returned when you call `len(list)`.
- Each object reference (values in the list) is 8 bytes
- The sizes depend on your computer, but will be constant

The main benefit of an array-based list is that we can access any element by index in  $O(1)$  time by using arithmetic computations.

Python pre-allocates a larger block of memory for the list in anticipation you will add data to it



read by index

`x = list[2]`

length = 6	
index	value
0	'Alice'
1	'Bob'
2	'Fran'
3	'John'
4	12345
5	10.0

write to index

`list[2] = 'Eugene'`

length = 6	
index	value
0	'Alice'
1	'Bob'
2	'Fran'
3	'John'
4	12345
5	10.0

`list.append(value)`

`list.append('Bart')`

length = 6	
index	value
0	'Alice'
1	'Bob'
2	'Fran'
3	'John'
4	12345
5	10.0

`list.pop()`

`list.pop()`

length = 6	
index	value
0	'Alice'
1	'Bob'
2	'Fran'
3	'John'
4	12345
5	10.0

```
list.insert(index, value)
list.insert(0, 'Paris')
```

length = 6	
index	value
0	'Alice'
1	'Bob'
2	'Fran'
3	'John'
4	12345
5	10.0

```
list.pop(index)
list.pop(0)
```

length = 6	
index	value
0	'Alice'
1	'Bob'
2	'Fran'
3	'John'
4	12345
5	10.0

```
list.remove(value)
list.remove('Alice')
```

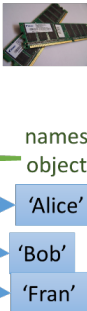
length = 6	
index	value
0	'Alice'
1	'Bob'
2	'Fran'
3	'John'
4	12345
5	10.0

```
list.remove('Horatio')
```

length = 6	
index	value
0	'Alice'
1	'Bob'
2	'Fran'
3	'John'
4	12345
5	10.0

```
list.append() and list.insert(index, value) – revisited
list.append('Bart')
```

Address	Value
0x0000000	
0x0000008	<OBJECT HEADER>
0x0000010	
0x0000018	length: int
0x0000020	0x2dbf9a0
0x0000028	0x42e7599
0x0000030	0x3f81634
0x0000038	<unrelated data>
0x0000040	<unrelated data>



# Python list[] and the ArrayList Data Structure

CSC 231 – Introduction to Data Structures

Dr. Lucas Layman



## Data types vs. data structures

**Data type**: data and operations you can perform on that data, e.g., the Python `list` class

**Data structure**: an implementation of how the data items are stored "under the hood", i.e., how the Python `list` class organizes its data internally.



## The ArrayList data structure

Python's `list` class organizes its data in what we call an ArrayList structure.

An array is a contiguous block of memory.

The main benefit of an array-based list is that we can access any element of the list in  $O(1)$  time.



Address	Value
0x0000000	
0x0000008	
0x0000010	
0x0000018	
0x0000020	
0x0000028	
0x0000030	

## What happens when a `list` is instantiated in memory

```
names = []
```

HEADER is 24 bytes

length is 8 bytes

- the exact number of bytes will depend on your computer, but it will be constant for your computer and that is the important thing.

the `length` is returned when you call `len(names)`.

Python *pre-allocates* a larger block of memory for the list in anticipation you will add data to it



Address	Value	
0x00000000	<OBJECT HEADER>	names object
0x00000008		
0x00000010		
0x00000018	length: int	
0x00000020		
0x00000028		
0x00000030		free memory
0x00000038		
0x00000040		





```
list.append()
```

```
names.append('Alice')
```

Appending an item to the list puts a *reference* to the memory address of the item's value at the end of the list's memory space

Each object reference is 8 bytes

- the exact number of bytes will depend on your computer, but it will be *constant*



Address	Value
0x00000000	<OBJECT HEADER>
0x00000008	
0x00000010	
0x00000018	length: int
0x00000020	0x2dbf9a0
0x00000028	
0x00000030	
0x00000038	
0x00000040	



names  
object

'Alice'

free  
memory

## This is the *data structure* of a Python list

```
names.append('Bob')
```

A data structure is how a data type organizes its data internally.

The Python `list`'s data structure is to use a contiguous array of memory to store references to items

Address	Value
0x00000000	<OBJECT HEADER>
0x00000008	
0x00000010	
0x00000018	length: int
0x00000020	0x2dbf9a0
0x00000028	0x42e7599
0x00000030	
0x00000038	
0x00000040	



names  
object

'Alice'

'Bob'



## A simplified model of the **list** data structure

```
names = []  
names.append('Alice')  
names.append('Bob')  
names.append('Fran')  
names.append('John')  
names.append(12345)  
names.append(10.0)
```

names	
len	6
0	'Alice'
1	'Bob'
2	'Fran'
3	'John'
4	12345
5	10.0



**REMEMBER:** Lists store object references, not the data value!



Python `list`'s data structure dictates the  $O(f(n))$  of its operations

Big-O	Code	Operation
$O(1)$	<code>list[index]</code>	get item by index
$O(1)$	<code>list[index] = x</code>	set object at index to be x
$O(1)$	<code>list.append(x)</code>	append x to the end of list
$O(1)$	<code>list.pop()</code>	remove from end of list and return it
$O(n)$	<code>list.pop(i)</code>	remove the item at index i and return it
$O(n)$	<code>list.remove(x)</code>	remove the first item whose value is x
$O(n)$	<code>list.insert(i, x)</code>	insert x at index i
$O(n)$	iteration (for x in list)	iterate over each item in list

You need to know these.



## A closer look at Python list[] operations

Refer to your worksheet

You are responsible for knowing how the operations from the previous slide work on the ArrayList data structure underlying Python's list[] data type

You are responsible for knowing the Big-O of each operation



Demonstrating the implications of Big-O in code

We will test the runtimes of list[] operations to see firsthand  $O(1)$  vs.  $O(n)$  time complexity



# Abstract Data Types (ADTs), Data Types, and Data Structures (oh my...)

CSC 231 – Introduction to Data Structures

Dr. Lucas Layman



## Python concept: return None

```
def get_index(item, a_list):
    index = 0
    for x in a_list:
        if item == x:
            return index
        index += 1

nums = [1, 981273, 132, 120984, 6, 5, 7456, 345, 23, 4]
print(get_index(120984, nums))
print(get_index(99999, nums))
```





## Python concept: return None (2)

### What is None?

- A built-in constant, like True or False
- None is used to represent the absence of a value

Python functions and methods return None by default if you do not specify a return value



Python `list` is implemented with an ArrayList data structure

Big-O	Code
$O(1)$	<code>list[index]</code>
$O(1)$	<code>list[index] = x</code>
$O(1)$	<code>list.append(x)</code>
$O(1)$	<code>list.pop()</code>
$O(n)$	<code>list.pop(i)</code>
$O(n)$	<code>list.remove(x)</code>
$O(n)$	<code>list.insert(i, x)</code>
$O(n)$	iteration (for x in list)

You need to know these.

Address	Value
0x0000000	<OBJECT HEADER>
0x0000008	
0x0000010	
0x0000018	length: int
0x0000020	0x2dbf9a0
0x0000028	
0x0000030	
0x0000038	
0x0000040	



names  
object  
'Alice'

free  
space



## ADTs, data types and data structures

*Abstract Data Type (ADT)*: a conceptual thing across computing. Usually oriented around solving a problem.

- A collection of data items (its state), and
- The basic relationships among them and operations that must be performed on them (its behavior)

*Data type* or *type* or *class*: The implementation of an ADT using a language

*Data structures*: How the data type organizes its data items



## Your first Abstract Data Type

The *UnorderedList* ADT: a collection of items where each item holds a relative position with respect to the others

It is *unordered* because the position of an item is *not* determined by its value



## UnorderedList, list[], and ArrayList

UnorderedList ADT operations:

- adding and remove items by value
- appending items to the end
- inserting and removing an item at a specific location (index)
- determining the size of the list, or if it is empty
- searching for an item in the list

The Python `list[]` data type is a concrete implementation of the UnorderedList ADT.

- We will create another data type next week that also implements the UnorderedList ADT

The Python `list[]` data type uses an ArrayList data structure to organize the elements

Address	Value
0x0000000	
0x0000008	<OBJECT HEADER>
0x0000010	
0x0000018	length: int
0x0000020	0x2dbf9a0
0x0000028	
0x0000030	
0x0000038	
0x0000040	



## More ADTS – the Stack, the Queue, and the Deque

What distinguishes one ADT from another is the *location* in which these additions and removes occur

A stack adds and removes from the top only

A queue adds to the rear and removes from the front

A deque adds and removes from either end

'Alice'
'Bob'
'Daisy'
'John'
12345
Patient



# Stacks

CSC 231 – Introduction to Data Structures

Dr. Lucas Layman



## Stacks

A *stack* or "push-down stack" is an ADT where:

Addition and removal of items always happens at the *same end*, which is called the top of the stack

A stack has last-in first-out (LIFO) behavior – the most recently added item will be removed first.





## Stack properties and applications

The LIFO nature of stacks means that items are removed in the reverse order in which they were added

Some applications of stacks:

- The 'Back' button in your browser
- Undo
- Parsing programming language syntax



## Stack data items and behaviors

Stacks have the following methods:

- `Stack()` – class constructor
- `push(item)` – adds item to the top of the stack.
- `pop()` – removes and returns the top item on the stack
- `peek()` – returns the top item on the stack, but does not remove it.
- `is_empty()` – returns True if the stack is empty, False otherwise
- `size()`, a.k.a., `__len__()` – returns the numbers of item on the stack

Stack data is typically stored in some sort of list



Visualizing a stack

Refer to your worksheet



## Coding a Stack



## Python concept: modules

.py scripts are technically called Python modules.

Importing a module allows you to use its code (main statements, functions, classes) in another module

- e.g., you may have seen `import sys` or `import random` at the top of files.



## Python concept: modules (2)

Two approaches to using module contents. Either is fine:

```
import stack                # there is a file named stack.py
x = stack.Stack()           # the Stack class in is stack.py
x.push("Alice")
```

```
from stack import Stack
x = Stack()
x.push("Alice")
```



## Python concept: modules (3)

When you import a module, ALL of the code in the module runs.

You may wish to specify that some parts of your module only run when that module is the "main" program.

The code in `if __name__ == "__main__":` will run only when the module is run as the "main" program but will not run when imported.



## Big-O for stack operations

	"Top" of the stack is the <b>end</b> of the list	"Top" of the stack is <b>index 0</b> of the list
Stack()	O(1)	O(1)
push(item)	O(1)	O(n)
pop()	O(1)	O(n)
peek()	O(1)	O(1)
is_empty()	O(1)	O(1)
size()	O(1)	O(1)





# Queues

CSC 231 – Introduction to Data Structures

Dr. Lucas Layman



## Queues

A *queue* is an ADT where:

Addition happens at one end: the rear

Removal happens at the other end: the front

A queue has first-in first-out (FIFO) behavior – the first item added will be removed first.



## Queue properties and applications

The FIFO nature of queues means that the last item added must wait for all the others to be removed

Some applications of queues:

- A printer queue
- Processing keystrokes



## Queue data items and behaviors

Queue data items are typically stored in a *list*

Queues have the following methods:

- `Queue()` – class constructor
- `enqueue(item)` – adds item to the rear of the queue
- `dequeue()` – removes and returns the item at the front of the queue
- `is_empty()` – returns True if the queue is empty, False otherwise
- `size()`, a.k.a., `__len__()` – returns the numbers of items in the queue



Visualizing a queue

Refer to your worksheet



## Big-O for queue operations

	"Front" of the queue is the <b>end</b> of the list	"Front" of the queue is <b>index 0</b> of the list
Queue()	O(1)	O(1)
enqueue(item)	O(n)	O(1)
dequeue()	O(1)	O(n)
is_empty()	O(1)	O(1)
size()	O(1)	O(1)

We **cannot** get all of these to be O(1) if we use a single Python list!

# Deque – double-ended queue

CSC 231 – Introduction to Data Structures

Dr. Lucas Layman



## Deque ("deck") – double-ended queue

A *deque* is an ADT where addition and removal happen at both ends.

A deque essentially combines the behaviors of a stack and a queue





## Deque properties and applications

The deque provides maximum flexibility in where items are added to the list, but is also least efficient

Some applications of deques:

- CPU process scheduling
- Browser history or Undo lists that are limited in size, i.e., your browser can remember only the last 20 sites you visited



## Deque data items and behaviors

Deque data items are typically stored in a *list*

Deques have the following methods:

- `Deque()` – class constructor
- `add_front(item)` – adds item to the front of the deque
- `add_rear(item)` – adds item to the rear of the deque
- `remove_front()` – removes and returns the item at the from of the front of the deque
- `remove_rear()` – removes and returns the item at the from of the rear of the deque
- `is_empty()` – returns True if the queue is empty, False otherwise
- `size()`, a.k.a., `__len__()` – returns the numbers of items in the deque



Visualizing a deque

Refer to your worksheet



## Let's talk Big-O for dequeues

	"Front" of the deque is the <b>end</b> of the list	"Front" of the deque is <b>index 0</b> of the list
Deque()	O(1)	O(1)
add_front(item)	O(1)	O(n)
add_rear()	O(n)	O(1)
remove_front()	O(1)	O(n)
remove_rear()	O(n)	O(1)
is_empty()	O(1)	O(1)
size()	O(1)	O(1)

We **cannot** get all of these to be O(1) if we use a single Python list!

## Improving Queue and Deque operations

The primary behaviors of queues and dequeues that we care about are adding and removing items...

But in both of these ADTs, at least one of these operations (adding or removing) is  $O(n)$  because we use a Python list internally

We can make the adding and removing operations  $O(1)$ , but we need to create our own notion of a list... next time



## CSC 242 Syllabus

## Course Syllabus

### [Jump to Today](#)

**Instructor:** [Dr. Lucas Layman](http://people.uncw.edu/laymanl) [\\_ \(http://people.uncw.edu/laymanl\)](http://people.uncw.edu/laymanl)

- **Email:** [laymanl@uncw.edu](mailto:laymanl@uncw.edu) [\\_ \(mailto:laymanl@uncw.edu\)](mailto:laymanl@uncw.edu)
- **Office hours:** MW, 9:30–11:30am, CIS 2045

**Where:** [CIS 2055](https://www.google.com/maps/search/?api=1&query=cis+building+uncw+28409) [\\_ \(https://www.google.com/maps/search/?api=1&query=cis+building+uncw+28409\)](https://www.google.com/maps/search/?api=1&query=cis+building+uncw+28409)

**When:**

- Section 001: TR, 9:30–10:50am
- Section 002: TR, 3:30– 4:50pm

## Syllabus revision following Hurricane Florence

As you know, all UNCW classes were canceled on 9/10/18 starting at 12:00 pm and resumed on 10/8/18. This resulted in a loss of 600 instructional minutes for this course. UNC System Policy 400.1.6 requires 750 instructional minutes (or the equivalent) for each credit-hour of a course. For example, a 3-credit hour course requires 2250 minutes of instructional time. To make up for such a large and unprecedented amount of lost time, UNCW Academic Affairs developed a diverse and flexible plan. This plan contains adjustments to the academic calendar, changes to the daily schedule, and the opportunity to make-up time through outside of class and/or online assignments. In some cases faculty and students can also decide to hold make-up courses if necessary. This revised syllabus reflects how this plan impacts our course.

A summary of changes are listed here. Changes to the text within the syllabus are highlighted in blue.

- Class now ends 5 minutes later. The start times remains the same. Please review the [revised start and end times for all your classes](https://uncw.edu/aa/ClassSchedulePlanInstructionalMinutes.pdf) [\\_ \(https://uncw.edu/aa/ClassSchedulePlanInstructionalMinutes.pdf\)](https://uncw.edu/aa/ClassSchedulePlanInstructionalMinutes.pdf).
- The final exams are moved to December 12th. Refer to the university's updated [Final Exam Schedule](https://uncw.edu/reg/exams-fall18.html) [\\_ \(https://uncw.edu/reg/exams-fall18.html\)](https://uncw.edu/reg/exams-fall18.html).
- We will meet on October 11th, which was previously Fall Break.
- We will meet on December 6th, which was previously Reading Day.
- You will be given two out-of-class lectures in the form of videos and readings to make up for lost instructional time. You will have a 1-2 week window in which to complete these readings and take an associated take-home quiz. The content of these out-of-class lectures will

CSC 242 Syllabus

appear on subsequent exams.

- The schedule of Assignments and Exams has been posted to help you plan.
- Grade allocation among assignments and exams is changed. There will now be two midterm exams.
- Contact me if the hurricane will impact your attendance or ability to complete assignments on time. Obtaining concessions under false pretenses will result in a grade of F for the course and an Honor Code Violation report.

## Course description

This course is an introduction to *how* computers think. We will cover:

<b>Basic computation</b>	What it means to be a "computer"
<b>Binary code</b>	How computers represent information
<b>Logic circuits</b>	How computers make decisions based on information
<b><u>CPU</u></b>	The complex circuits that make up the brain of the computer
<b>Assembly language</b>	The lowest level language by which programmers and programming languages communicate with the CPU

The first part of the course is dedicated to understanding binary code and logic circuits. In the second part of the course, students will use a simulation tool to create a CPU using complex logic circuits. In the third (final) part of the course, students will implement a program in assembly language that runs on the simulated computer.

**What to expect:** Each class will be a mix of lecture and in-class activities. Each homework assignment, quiz, and exam will compound on all previous work so that you will have ample opportunity to practice and become expert in the material.

Lecture notes will be posted after class with key information missing that was covered during lecture. You must come to class prepared with note-taking implements.

The material in this course will likely be different from other Computer Science courses you have taken. If you were studying to be a mechanic, this is the course where you learn about internal combustion engines and the physics behind them. If you were studying to be a doctor, this is the course where you learn about the heart and the chemical reactions that control it. We will be talking about math, circuits, logic, and what goes on "under the hood" in a CPU. The fundamentals of binary code and boolean logic will come back to you time-and-again in subsequent courses. The basics of how a CPU is organized (caches, registers, etc.) are concepts that are echoed in high-level programming languages albeit often under different names.

**Prerequisites:** [CSC 131](http://uncw.edu/csc/undergrad/courses/csc131.html) [.\(http://uncw.edu/csc/undergrad/courses/csc131.html\)](http://uncw.edu/csc/undergrad/courses/csc131.html)., [CSC 133](http://uncw.edu/csc/undergrad/courses/csc133.html) [.\(http://uncw.edu/csc/undergrad/courses/csc133.html\)](http://uncw.edu/csc/undergrad/courses/csc133.html)

**Student learning outcomes:**

1. Students develop knowledge and understanding between hardware/middleware and frameworks for high level programming languages.
2. Students develop knowledge of combinational and sequential logic circuits.
3. Students learn how modern computers are constructed from basic logic gates and sequential elements.
4. Students learn the major components of a modern processor, ALU, Control Unit and Memory.
5. Students learn how to create and use processor specific assembly language.

## Communication

There are no dumb questions. Please ask me if you are unsure about a topic, an assignment, or a course policy, or just need help. My goal is for you to learn the material and thus to be successful in this course.

I prefer to talk with you in person. I encourage you to attend my office hours. Office hour information is listed at the beginning of the syllabus. Individual appointments outside office hours can be arranged.

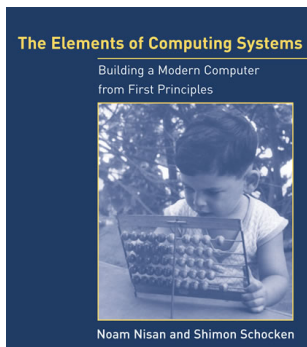
You may also [email me \(mailto:laymanl@uncw.edu\)](mailto:laymanl@uncw.edu). Please note that I do not respond to electronic communication after 6pm, on weekends, or during holidays.

## Textbooks

*The Elements of Computing Systems* is required. The [free, online edition of the first few chapters](http://nand2tetris.org/course.php) [.\(http://nand2tetris.org/course.php\)](http://nand2tetris.org/course.php) is sufficient to complete the course. A printed text version is available from retailers if you desire.

*Code: The Hidden Language of Computer Hardware and Software* is recommended reading. This book describes many course concepts at a high level intended for a general audience, whereas in class we will go much deeper into each subject. This book offers a different perspective that may help you if you are struggling with the course material. It is available at the Campus Bookstore and from online retailers.





(<http://www.nand2tetris.org/>).

*Required*

**The Elements of Computing Systems: Building a Modern Computer from First Principles**

Noam Nisan and Shimon Schocken

ISBN-13: 978-0262640688

**Note:** Chapters will be posted to Canvas. But if you want to buy a hard copy, here are links:

**Amazon** (<http://a.co/gDVhbTt>)

**Barnes & Noble** (<https://www.barnesandnoble.com/w/elements-of-computing-systems-noam-nisan/1100660146?ean=9780262640688>)



---

*Recommended*

**Code: The Hidden Language of Computer Hardware and Software**

Charles Petzold

ISBN: 0-7356-1131-9

[Amazon](http://a.co/gnZQ2dz) [.\(http://a.co/gnZQ2dz\)](http://a.co/gnZQ2dz)

[Barnes & Noble](https://www.barnesandnoble.com/w/code-charles-petzold/1100324884?ean=9780735611313) [.\(https://www.barnesandnoble.com/w/code-charles-petzold/1100324884?ean=9780735611313\)](https://www.barnesandnoble.com/w/code-charles-petzold/1100324884?ean=9780735611313)

## Course Policies

### Seahawk Respect Compact

We adhere to the [Seahawk Respect Compact](http://uncw.edu/diversity/src.html) [.\(http://uncw.edu/diversity/src.html\)](http://uncw.edu/diversity/src.html). All individuals in this class will be treated with inclusiveness, mutual respect, acceptance, and open-mindedness by all persons.

### Student Academic Honor Code

All members of UNCW's community are expected to follow the academic Honor Code. Please read the UNCW [Student Academic Honor Code](http://uncw.edu/odos/honorcode/) [.\(http://uncw.edu/odos/honorcode/\)](http://uncw.edu/odos/honorcode/) carefully. Academic dishonesty in any form will not be tolerated in this class.

Be familiar with UNCW's position on plagiarism as outlined in the UNCW [Student Academic Honor Code](http://uncw.edu/odos/honorcode/) [.\(http://uncw.edu/odos/honorcode/\)](http://uncw.edu/odos/honorcode/). Plagiarism is a form of academic dishonesty in which you take someone else's ideas and represent them as your own. Here are some examples of plagiarism:

1. You write about someone else's work in an assignment and do not give them credit for it by referencing them.
2. You give a presentation and use someone else's ideas and do not state that the ideas are the other person's.
3. You get facts from your textbook or some other reference material and do not reference that material.

### Collaboration, cheating, and personal proficiency

**Collaboration:** You may discuss course content with your peers. You may seek out additional resources (i.e., the Internet) to help you *understand* the course content.

All coursework is to be completed *individually* except when explicitly indicated by the instructor. If collaboration is permitted, the collaborative coursework must bear the names of all collaborators on the team and all collaborators must contribute equally. Grading on collaborative assignments may be weighted by individual contribution and peer evaluation.

**Cheating:** Obtaining answers to assignments, quizzes, exams, or projects from any source other than your own brain is cheating. Any person completing work on your behalf is cheating. All coursework is to be completed individually unless explicitly stated otherwise (i.e., "collaborative"). Incidents of cheating will be addressed according to the policies in the [Student Academic Honor Code](http://uncw.edu/odos/honorcode/) (<http://uncw.edu/odos/honorcode/>).

**Personal proficiency:** You are expected to become proficient in the course content. A substantial portion of your course grade is based on individual, in-class assessment (i.e., quizzes and exams). Over-reliance on peer discussion or Internet sources will lead to poor grades on these individual assessments.

## Attendance

Contact me personally, preferably via email, if the impacts of Hurricane Florence change your ability to attend class on time and remain for the duration. Allowances will be made provided adequate justification and evidence.

From the UNCW [Faculty Handbook](http://www.uncw.edu/facsen/documents/Faculty_Handbook.pdf) ([http://www.uncw.edu/facsen/documents/Faculty\\_Handbook.pdf](http://www.uncw.edu/facsen/documents/Faculty_Handbook.pdf)) (V.A.1.b, pp 120):

Students are expected to be present at all regular class meetings and examinations for the courses in which they are registered. It is the responsibility of the students to learn and comply with the policies set for each class in which they are registered.

You will need to attend class to succeed in the course. You are expected to attend every class, be present at the start time, and stay for the duration.

I will be much less likely to make allowances (e.g., make up quizzes, accepting late homework, scheduling extra office hours) for students with more than two unexcused absences. The following is a partial list of *unexcused* reasons for absence, tardiness, or early departure:

- I have a court date.
  - My boyfriend/roommate/girlfriend and I are having problems.
- CSC 242 Syllabus

- I have an appointment.
- I am going on vacation.
- I have to work.
- I have an admissions interview for another college.
- I got locked out of my apartment.
- I overslept.
- I couldn't find my car keys.
- My dog/cat/bird etc. got out.
- I couldn't get a parking spot.
- I was hungover/I was out late the night before.
- My alarm/roommate/friend did not wake me up.
- Traffic was bad.
- I was having one of those days so I went back to bed.

## Personal electronics and computer use

Phones, tablets, or other personal electronic devices may not be used during class except for programming exercises as directed by the instructor. You will turn off your cell phone and place it either face down on the desk in front of you or stow it in your bag for the duration of class. These devices are a distraction to other students and yourself. It is physiologically impossible to concentrate on classwork and a personal device at the same time.

Laptop and desktop computers may only be used for reading lecture material, taking lecture notes, or performing assigned programming tasks.

Failure to abide by this policy will result in dismissal from the classroom and letter grade deductions.

## Grading

You are graded cumulatively on each of the items below. These cumulative grades are then weighted.

- 35% – Homework assignments/project

CSC 242 Syllabus

- 20% – Midterm 1
- 20% – Midterm 2
- 25% – Final exam

For example, suppose you receive a 10/15 on Assignment 1 and 18/20 on Assignment 2. Your cumulative assignment score would be 28/35. This score would be multiplied by 40% and added to the weighted scores for the quizzes, midterm, and final. Your course letter grade will be determined by the sum of these weighted scores according to the scale below.

<b>A</b>	[94.0,∞]
<b>A-</b>	[90.0,94.0)
<b>B+</b>	[87.0,90.0)
<b>B</b>	[84.0,87.0)
<b>B-</b>	[80.0,84.0)
<b>C+</b>	[77.0,80.0)
<b>C</b>	[74.0,77.0)
<b>C-</b>	[70.0,74.0)
<b>D+</b>	[67.0,70.0)
<b>D</b>	[64.0,67.0)
<b>D-</b>	[60.0,64.0)
<b>F</b>	[0.0,60.0)

Contact me personally, preferably via email, if the impacts of Hurricane Florence change your ability to complete assignments on time. Allowances will be made provided adequate justification and evidence.

**Homework assignments** are due at the time and date indicated. Late submissions will receive a grade of zero. You are encouraged to submit your assignments early.

There are no make-ups for assignments or quizzes that are late or incomplete for unexcused reasons (see the Attendance policy). Make-ups for *excused* absences will be addressed on a case-by-case basis.

CSC 242 Syllabus

**Exams:** Students who miss an exam will receive a **Course Grade of F**. No makeup examination will be given except for reasons of illness or other verified emergency.

**Regrading policy:** Requests to regrade must be made in person or via email no later than the next class day after the graded item is returned. For example, if class meets MWF and an exam is returned on Monday, the request to regrade must be made no later than Wednesday.

## University Learning Center

You are encouraged to visit the University Learning Center (ULC) if you need extra support in mathematics, writing, or managing your schedule. The ULC's mission is to help students become successful, independent learners. Tutoring at the ULC is NOT remediation: the ULC offers a different type of learning opportunity for those students who want to increase the quality of their education. ULC services are free to all UNCW students and include the following:

- [Learning Services](http://uncw.edu/ulc/learning/) [\(http://uncw.edu/ulc/learning/\)](http://uncw.edu/ulc/learning/)
- [Math Services](http://www.uncw.edu/ulc/math/index.html) [\(http://www.uncw.edu/ulc/math/index.html\)](http://www.uncw.edu/ulc/math/index.html)
- [Supplemental Instruction](http://www.uncw.edu/ulc/si/index.html) [\(http://www.uncw.edu/ulc/si/index.html\)](http://www.uncw.edu/ulc/si/index.html)
- [Writing Services](http://www.uncw.edu/ulc/writing/index.html) [\(http://www.uncw.edu/ulc/writing/index.html\)](http://www.uncw.edu/ulc/writing/index.html)










## Students with disabilities











Students with diagnosed disabilities should contact the Office of Disability Services (910-962-7555). Please submit to me a copy of the letter you receive from Office of Disability Services detailing class accommodations you need. If you require accommodation for test-taking, make sure I have the referral letter no fewer than three days before the test.

## Violence and harassment









UNCW practices a zero tolerance policy for any kind of violent or harassing behavior. If you are experiencing an emergency of this type contact the police at 911 or UNCW CARE at 962-2273. Resources for individuals concerned with a violent or harassing situation can be located at [the UNCW Title IX website](http://uncw.edu/titleix/) [\(http://uncw.edu/titleix/\)](http://uncw.edu/titleix/).

## Course Summary:

Date	Details
Thu Aug 23, 2018	 <a href="https://uncw.instructure.com/calendar?event_id=2488&amp;include_contexts=course_9318">First day of class (CSC-242-001)</a> (https://uncw.instructure.com/calendar?event_id=2488&include_contexts=course_9318) 9:30am
	 <a href="https://uncw.instructure.com/calendar?event_id=2489&amp;include_contexts=course_9318">First day of class (CSC-242-002)</a> (https://uncw.instructure.com/calendar?event_id=2489&include_contexts=course_9318) 3:30pm to 4:45pm
Tue Aug 28, 2018	 <a href="https://uncw.instructure.com/courses/9318/assignments/21890">CSC 242 Survey</a> (https://uncw.instructure.com/courses/9318/assignments/21890) due by 11:59pm
Wed Aug 29, 2018	 <a href="https://uncw.instructure.com/calendar?event_id=2490&amp;include_contexts=course_9318">Last day to add/drop</a> (https://uncw.instructure.com/calendar?event_id=2490&include_contexts=course_9318) 12am
Fri Oct 12, 2018	 <a href="https://uncw.instructure.com/courses/9318/assignments/29897">Assignment 1 - Combinational Circuits</a> (https://uncw.instructure.com/courses/9318/assignments/29897) due by 11:59pm
Thu Oct 18, 2018	 <a href="https://uncw.instructure.com/courses/9318/assignments/32839">Midterm Exam 1</a> (https://uncw.instructure.com/courses/9318/assignments/32839) (CSC-242-001) due by 9:30am
	 <a href="https://uncw.instructure.com/courses/9318/assignments/32839">Midterm Exam 1</a> (https://uncw.instructure.com/courses/9318/assignments/32839) (CSC-242-002) due by 3:30pm
Mon Oct 22, 2018	 <a href="https://uncw.instructure.com/calendar?event_id=2492&amp;include_contexts=course_9318">Last day to withdraw</a> (https://uncw.instructure.com/calendar?event_id=2492&include_contexts=course_9318) 12am
Wed Oct 24, 2018	 <a href="https://uncw.instructure.com/courses/9318/assignments/32843">Project 1 - Implementing Logic Circuits</a> (https://uncw.instructure.com/courses/9318/assignments/32843) due by 11:59pm

Date	Details	
Thu Nov 1, 2018	 <a href="https://uncw.instructure.com/courses/9318/assignments/32842">Assignment 2 - Number Systems</a> ( <a href="https://uncw.instructure.com/courses/9318/assignments/32842">https://uncw.instructure.com/courses/9318/assignments/32842</a> ) (CSC-242-001)	due by 9:30am
	 <a href="https://uncw.instructure.com/courses/9318/assignments/37070">Take-Home Quiz 1 - Integer number system videos</a> ( <a href="https://uncw.instructure.com/courses/9318/assignments/37070">https://uncw.instructure.com/courses/9318/assignments/37070</a> ) (CSC-242-001)	due by 9:30am
	 <a href="https://uncw.instructure.com/courses/9318/assignments/32842">Assignment 2 - Number Systems</a> ( <a href="https://uncw.instructure.com/courses/9318/assignments/32842">https://uncw.instructure.com/courses/9318/assignments/32842</a> ) (CSC-242-002)	due by 3:30pm
	 <a href="https://uncw.instructure.com/courses/9318/assignments/37070">Take-Home Quiz 1 - Integer number system videos</a> ( <a href="https://uncw.instructure.com/courses/9318/assignments/37070">https://uncw.instructure.com/courses/9318/assignments/37070</a> ) (CSC-242-002)	due by 3:30pm
Sun Nov 11, 2018	 <a href="https://uncw.instructure.com/courses/9318/assignments/32844">Project 2 - Arithmetic Chips and the ALU</a> ( <a href="https://uncw.instructure.com/courses/9318/assignments/32844">https://uncw.instructure.com/courses/9318/assignments/32844</a> )	due by 11:59pm
Thu Nov 15, 2018	 <a href="https://uncw.instructure.com/courses/9318/assignments/32840">Midterm Exam 2</a> ( <a href="https://uncw.instructure.com/courses/9318/assignments/32840">https://uncw.instructure.com/courses/9318/assignments/32840</a> ) (CSC-242-001)	due by 9:30am
	 <a href="https://uncw.instructure.com/courses/9318/assignments/32840">Midterm Exam 2</a> ( <a href="https://uncw.instructure.com/courses/9318/assignments/32840">https://uncw.instructure.com/courses/9318/assignments/32840</a> ) (CSC-242-002)	due by 3:30pm
Thu Nov 22, 2018	 <a href="https://uncw.instructure.com/calendar?event_id=2493&amp;include_contexts=course_9318">No class (Thanksgiving Break)</a> ( <a href="https://uncw.instructure.com/calendar?event_id=2493&amp;include_contexts=course_9318">https://uncw.instructure.com/calendar?event_id=2493&amp;include_contexts=course_9318</a> )	12am
Thu Nov 29, 2018	 <a href="https://uncw.instructure.com/courses/9318/assignments/32845">Extra Credit Assignment - Sequential Chips and Memory</a> ( <a href="https://uncw.instructure.com/courses/9318/assignments/32845">https://uncw.instructure.com/courses/9318/assignments/32845</a> )	due by 11:59pm
Fri Nov 30, 2018	 <a href="https://uncw.instructure.com/courses/9318/assignments/32846">Project 3, Part 1 - Machine Code and CPU Execution</a> ( <a href="https://uncw.instructure.com/courses/9318/assignments/32846">https://uncw.instructure.com/courses/9318/assignments/32846</a> )	due by 11:59pm



Date	Details	
Thu Dec 6, 2018	 <a href="https://uncw.instructure.com/calendar?event_id=5796&amp;include_contexts=course_9318">Last day of class (CSC-242-001)</a> ( <a href="https://uncw.instructure.com/calendar?event_id=5796&amp;include_contexts=course_9318">https://uncw.instructure.com/calendar?event_id=5796&amp;include_contexts=course_9318</a> )	9:30am to 10:50am
	 <a href="https://uncw.instructure.com/calendar?event_id=5797&amp;include_contexts=course_9318">Last day of class (CSC-242-002)</a> ( <a href="https://uncw.instructure.com/calendar?event_id=5797&amp;include_contexts=course_9318">https://uncw.instructure.com/calendar?event_id=5797&amp;include_contexts=course_9318</a> )	3:30pm to 4:50pm
	 <a href="https://uncw.instructure.com/courses/9318/assignments/32847">Project 3, Part 2 - Assembly Language</a> ( <a href="https://uncw.instructure.com/courses/9318/assignments/32847">https://uncw.instructure.com/courses/9318/assignments/32847</a> )	due by 11:59pm
Tue Dec 11, 2018	 <a href="https://uncw.instructure.com/courses/9318/assignments/43098">Take-Home Quiz 2 - Representing other types of data</a> ( <a href="https://uncw.instructure.com/courses/9318/assignments/43098">https://uncw.instructure.com/courses/9318/assignments/43098</a> )	due by 11:59pm
	 <a href="https://uncw.instructure.com/courses/9318/assignments/48108">Take-Home Quiz 3 - Multiprocessors, Multicores, and Buses</a> ( <a href="https://uncw.instructure.com/courses/9318/assignments/48108">https://uncw.instructure.com/courses/9318/assignments/48108</a> )	due by 11:59pm
Wed Dec 12, 2018	 <a href="https://uncw.instructure.com/courses/9318/assignments/32841">Final Exam</a> ( <a href="https://uncw.instructure.com/courses/9318/assignments/32841">https://uncw.instructure.com/courses/9318/assignments/32841</a> ) (CSC-242-001)	due by 8am
	 <a href="https://uncw.instructure.com/courses/9318/assignments/32841">Final Exam</a> ( <a href="https://uncw.instructure.com/courses/9318/assignments/32841">https://uncw.instructure.com/courses/9318/assignments/32841</a> ) (CSC-242-002)	due by 3pm
	 <a href="https://uncw.instructure.com/courses/9318/assignments/53040">Course Grade</a> ( <a href="https://uncw.instructure.com/courses/9318/assignments/53040">https://uncw.instructure.com/courses/9318/assignments/53040</a> )	

**Peer Observations**

Fall 2017 Peer Observation

**Peer Observation Report  
Fall 2017 Observation*****Dr. Lucas Layman***

Dr. Lucas Layman was observed by Dr. D. Simmonds teaching CSC 242 in Bear 165, on 16 October 2017. Approximately 30 students were present. Dr. Layman arrived early and was addressing student needs informally and started the class at 9:00 AM by reminding students of due date of the current assignment, and overviewing the days class activities. He provided motivation for why that class material was important and where the lesson fitted in the overall understanding of programming and how computers operate. Dr. Layman then entered the lecture for the day. The planned lecture topics included the roles of and relationships between high level programming languages, compilers and interpreters, virtual machines, assemble language, machine language, hardware and microarchitecture. These topics were addresses in the first part of the class. In the second segment of the class, Dr. Layman allowed students to work of assigned projects as he availed himself to answer questions and provide guidance.

The class progressed well and the class material was clearly focused, well-chosen and presented, and the overall organization was thoughtfully prepared. Dr. Layman employed an interactive style and students participated freely. One highlight of the observation was his very clear articulation, his writing on the whiteboard in large letters, and his skillful use of practical references and explanations.

Dr. Layman's lecture was logically delivered and paced well for students. In addition, he demonstrated strong knowledge of the subject matter. There was good interaction with the students who seemed to be encouraged and positive throughout the learning experience. Dr. Layman seemed destined to be an excellent instructor.

# Publications

## Toward Predicting Success and Failure in CS2: A Mixed-Method Analysis

Lucas Layman  
University of North Carolina,  
Wilmington  
Wilmington, North Carolina, USA  
laymanl@uncw.edu

Yang Song  
University of North Carolina,  
Wilmington  
Wilmington, North Carolina, USA  
songy@uncw.edu

Curry Guinn  
University of North Carolina,  
Wilmington  
Wilmington, North Carolina, USA  
guinncc@uncw.edu

### ABSTRACT

Factors driving success and failure in CS1 are the subject of much study but less so for CS2. This paper investigates the *transition from CS1 to CS2* in search of leading indicators of success in CS2. Both CS1 and CS2 at the University of North Carolina Wilmington (UNCW) are taught in Python with annual enrollments of 300 and 150 respectively. In this paper, we report on the following research questions:

- (1) Are CS1 grades indicators of CS2 grades?
- (2) Does a quantitative relationship exist between CS2 course grade and a modified version of the SCS1 concept inventory?
- (3) What are the most challenging aspects of CS2, and how well does CS1 prepare students for CS2 from the student's perspective?

We provide a quantitative analysis of 2300 CS1 and CS2 course grades from 2013–2019. In Spring 2019, we administered a modified version of the SCS1 concept inventory to 44 students in the first week of CS2. Further, 69 students completed an exit questionnaire at the conclusion of CS2 to gain qualitative student feedback on their challenges in CS2 and on how well CS1 prepared them for CS2.

We find that 56% of students' grades were lower in CS2 than CS1, 18% improved their grades, and 26% earned the same grade. Of the changes, 62% were within one grade point. We find a statistically significant correlation between the modified SCS1 score and CS2 grade points. Students identify linked lists and class/object concepts among the most challenging. Student feedback on CS2 challenges and the adequacy of their CS1 preparations identify possible avenues for improving the CS1-CS2 transition.

### CCS CONCEPTS

• **Social and professional topics** → **CS1; Student assessment.**

### KEYWORDS

CS1, CS2, student assessment

#### ACM Reference Format:

Lucas Layman, Yang Song, and Curry Guinn. 2020. Toward Predicting Success and Failure in CS2: A Mixed-Method Analysis. In *2020 ACM Southeast Conference (ACMSE 2020)*, April 2–4, 2020, Tampa, FL, USA, <https://doi.org/10.1145/3374135.3385277>.

*Conference (ACMSE 2020)*, April 2–4, 2020, Tampa, FL, USA. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/3374135.3385277>

### 1 INTRODUCTION

CS1 and CS2 are important first courses for those interested in computing. CS1 is often an introduction to computer science and programming, and CS2 is often an introduction to data structures and algorithms, though the subject matter varies across institutions [11]. Students' performance and experience with the introductory programming sequence have a major impact on retention in the Computer Science major and has been often studied (e.g., [17, 28]). While students in CS1 may be dipping their toe in the water, students in CS2 have committed to a curricular path and failure in the CS2 course can be more impactful. We focus our research on the *transition from CS1 to CS2*. We want to improve student outcomes in CS2 by both identifying challenges with CS2 topics in particular, and ensuring that CS1 is best preparing students for CS2.

Both CS1 and CS2 at the University of North Carolina Wilmington (UNCW) are taught in Python with annual enrollments of 300 and 150 respectively. Approximately 15% of students enrolled in CS2 at UNCW do not earn the requisite 'C' grade to proceed to the next course in the sequence. Prior research has identified indicators of CS2 final grade, including overall Grade Point Average (GPA), number of absences, and performance in prerequisite courses including CS1 [5, 9], but these indicators are not necessarily actionable.

Our long term research goal is to investigate successful and unsuccessful transitions from CS1 to CS2 to identify interventions and pedagogical improvements in those courses. We begin with the following research questions:

- (1) Are CS1 grades indicators of CS2 grades?
- (2) Does a quantitative relationship exist between CS2 course grade and a modified version of the Second Computer Science 1 (SCS1) concept inventory [18]? The SCS1 evaluates students' understanding of sequencing, selection, iteration, and other CS1 concepts.
- (3) What are the most challenging aspects of CS2, and how well does CS1 prepare students for CS2 from the student's perspective?

For (1), we perform a quantitative analysis of 2300 course grades in CS1 and CS2 at UNCW from 2013–2019 to evaluate the relationship between CS1 and CS2 grades. For (2), we quantitatively evaluate SCS1 assessment [18] scores of CS1 skills against student grades for two sections of CS2 offered in Spring 2019. For (3), we analyze responses to a student questionnaire at the end of CS2 that obtained feedback on challenges faced in the course and on the

adequacy of CS1 in preparing them for CS2. We present the most common responses from students and discuss their implications on avenues for pedagogical and curricular improvements.<sup>1</sup>

The rest of the paper is organized as follows: Section 2 discusses prior research on CS2 performance, Section 3 describes the CS1 and CS2 courses at UNCW, Sections 4 and 5 present quantitative analyses of CS1 vs. CS2 grade and modified SCS1 score vs. CS2 grade, Section 6 presents our qualitative questionnaire findings, and we conclude in Section 7.5.

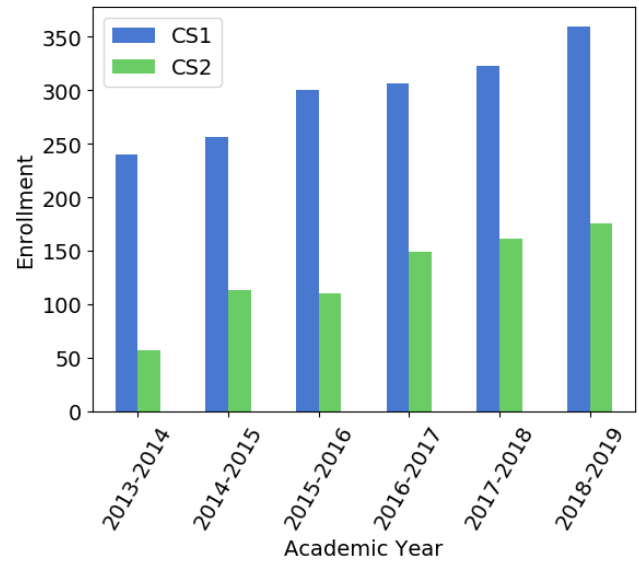
## 2 RELATED WORK

Our CS1 and CS2 courses are taught in Python. Koulouri et al. [13] report statistical findings that students better learned introductory programming concepts in Python (a "syntactically simple" language) than in Java, however Alzharani et al. found that beginning students struggle with assignments in Python as much or more than in C++ [1]. Educators have argued that that Python is a good choice for a first programming language [14, 22] because data typing, memory management, and object references are implicit, and that beginning programmers may find such concepts confusing. These concepts are usually important in a CS2 data structures course, though Enbody et al. [9] found no statistical difference in a C++ CS2 data structures course between students who took CS1 in Python vs. C++.

We are interested in indicators of *unsuccessful* transitions from CS1 to CS2. In this paper, we use *failure to pass CS2* as an indicator of an unsuccessful transition as it can be studied at scale – more fine-grained measures, such as failing early CS2 homeworks or exams, are warranted in future work. A host of literature reports the many personal [4, 17] societal [15], institutional [7], and pedagogical [29, 32] drivers that lead to student success and failure in CS courses in general. Overall GPA, CS1 grade, and perceived difficulty of prerequisite courses [5, 9] have been correlated with CS2 final grades, but these indicators are not actionable. Carter et al. [6] used measures of compilation, debugging, and execution in the development environment coupled with the frequency of message board interaction to develop a model that accounted for much of the variance in students' final CS2 grade in a C++ data structures course. Falkner and Falkner [10] find that students who turn in their first assignments on time statistically perform better across the CS curriculum.

Prior studies [23, 31] suggest that common CS2 topics are so-called *threshold concepts* [16] that transform one's understanding of computing and programming, including data structures, classes and inheritance, abstraction, and pointers. In general, research on CS1 concept acquisition greatly outnumbers research on CS2 concept acquisition, though CS2 is receiving increased focus (e.g., [12, 19]). Zingaro et al. [33] recently presented the most comprehensive study of data structure misconceptions based on qualitative analysis of 279 students' responses to final exam questions. Their study and Porter et al.'s [20] report on data structures learning goals led to the recent publication of the Basic Data Structures Inventory (BDSI), a multi-language, validated mechanism for assessing student understanding of linked list, array, binary tree, and binary search tree concepts [21].

<sup>1</sup>This study is approved by UNCW's Institutional Review Board: #19-0129.



**Figure 1: Enrollment History for CS1 and CS2 since Adopting Python**

Unfortunately, the BDSI was published after it could be incorporated into our current study.

Our work expands and adds to this body of literature in several ways. First, our study of the distributions between CS1 and CS2 grade is a large sample size ( $n=614$ ), providing quantitative evidence on a scale that can overcome biases such as instructor pedagogical effects. Second, no studies have been reported that quantitatively evaluate derivatives of the SCS1 concept inventory [18] (discussed in Section 5) as an indicator of CS2 success. Finally, we ask students to qualitatively reflect on CS2 challenges similar to Zingaro et al. [33], but also to consider how well CS1 prepared them for CS2. Educators should try to identify the students who are likely to have unsuccessful transitions from CS1 to CS2 as early as possible. By identifying which factors contribute to failure in CS2, we will be able to focus on how to improve aspects related to both CS1 and CS2 to reduce dropout rates.

## 3 DESCRIPTION OF CS1 AND CS2 COURSES AT UNCW

UNCW is an R2 Doctoral university per the Carnegie Classification. The Department of Computer Science offers undergraduate degree programs in Computer Science (~420 majors), Information Technology (~110 majors), and Digital Arts (~60 majors) as well as graduate programs in Computer Science and Information Systems (~35 students) and Data Science (~45 students). Our undergraduate major in Computer Science has received ABET accreditation since 2010-2011. Figure 1 shows enrollment in CS1 and CS2 each semester.

In this paper, CS1 and CS2 correspond to the terminology expressed in the ACM Computing Curriculum [2] guidelines where CS1 introduces basic algorithm design and programming concepts while CS2 introduces more advanced data abstractions and data

structures. Both CS1 and CS2 are four credits (200 minutes per week) equally divided between lecture and computer lab activities. No online sections are offered. All lectures and labs are led by a faculty member who is assisted by a graduate student who provides lab assistance, office hours, and grading help.

CS1 is CSC 131 – Introduction to Computer Science. Six to eight sections are offered each semester (1-3 sections in the summer) with approximately 20-25 students per section. The prerequisite for CS1 is a course in college algebra; no programming experience is assumed. CS1 has been taught using the Python programming language since Fall 2013 (Java was the language before that year). Typically, instructors introduce the IDLE development environment as well as command-line interpretation. The official learning outcomes for CS1 include a demonstrated ability to: understand and implement basic programming concepts (data types, conditionals, function definition), apply problem-solving techniques, use program control structures, practice modular programming, and use file input/output used to solve a variety of problems. The teaching methods and course structure are left to each instructor so long as they address the learning outcomes. All instructors implement a mix of homework and exam, and all courses are taught in a computer lab with programming exercises during most (if not all) sessions.

CS2 is CSC 231 – Introduction to Data Structures. Three to four sections are offered each semester (one in the summer) with approximately 20-25 students per section. The prerequisite for CS2 is CS1 (grade of C or higher) and Discrete Mathematics which is listed as a pre- or co-requisite. CS2 has been taught using the Python programming language since Spring 2014 (Java was the language before that year). Typically, instructors introduce a more advanced IDE such as PyCharm or Visual Studio Code. The official learning outcomes for CS2 include: learning multiple data structures for ordered and unordered data (e.g., lists, trees, hash tables), developing algorithms that implement the main operations of those data structures, performing Big-O analysis, and implementing projects that solve problems using those data structures. Instructors use a variety of in-class programming and written assessments combined with homeworks. Some implementations of the class are taught strictly in a computer lab, while others have two traditional lecture sections and one longer laboratory meeting each week.

Importantly, students must earn a grade of C or higher to enroll in future courses for which CS1 or CS2 is a prerequisite. These courses are required for Computer Science majors and minors as well as Digital Arts majors. Students who earn less than a C may repeat a course to earn a higher grade.

## 4 ANALYSIS OF THE RELATION BETWEEN CS1 GRADES AND CS2 GRADES

Our first research question is: *Are CS1 grades indicators of CS2 grades?* We look to validate previous research that found a correlation between CS1 and CS2 grades [5, 9]. Historical grade distributions also provide context and identify potential biases for interpreting our other analyses.

### 4.1 Grade Point Data Preparation

We collected all registered course grades for CS1 and CS2 from Fall 2013 (when first offered in Python) through Spring 2019. We

mapped the letter grades of [F, D-, D, D+, ..., B, B+, A-, A], to a 4.0 grade point scale. We collected 1641 CS1 letter grades and 706 CS2 letter grades in total. The CS1 sample is larger because CS1 is a required course for other non-Computer Science majors and also satisfies a general University Studies requirement. CS1 and CS2 were taught by 16 and 10 different faculty respectively in this period. Earning a C or higher is required to progress from both CS1 and CS2, and students may repeat a course if they earn less than a C. Approximately 7% of students repeat CS1 and CS2. Students' grades from all course attempts are included in this section's analysis; we exclude withdrawals and audits.

### 4.2 CS1 and CS2 Grade Point Analysis

Figure 2 shows the historical distribution of grades. CS1 has a greater proportion of A and F grades, while CS2 has a greater proportion of Bs. The difference between frequency distributions of grades is statistically significant ( $X^2 = 49.659, p < 0.001, df = 11$ ). We note that grade points earned is subject to heterogeneity in grading schemas between courses, and student performance in an individual class may be attributable to total course load and out-of-class reasons. The sample size should mitigate some of the effects, but we do not have data on the size of such effects.

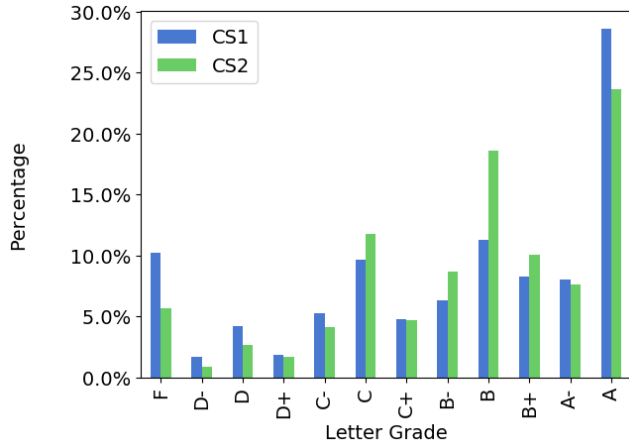
The outright failure rate of CS1 is 10.2% (167 Fs), whereas the failure rate in CS2 is 5.7% (40 Fs). The data suggests that students are more likely to excel or fail in CS1, whereas more students occupy the middle ground in CS2. The percentage of students who do not meet the C threshold required for dependant courses are 23.1% in CS1 and 15.0% in CS2. Our pass rates for CS1 are higher than published large-scale studies [3, 30]; large studies of CS2 pass rates are not available.

We examine the relationship between a student's most recent CS2 and CS1 course grades.<sup>2</sup> The scatterplot in Figure 3 shows no obvious relationship between a students' CS1 and CS2 grades, though a sizeable portion of students earn As in CS1 and similarly high grades in CS2.

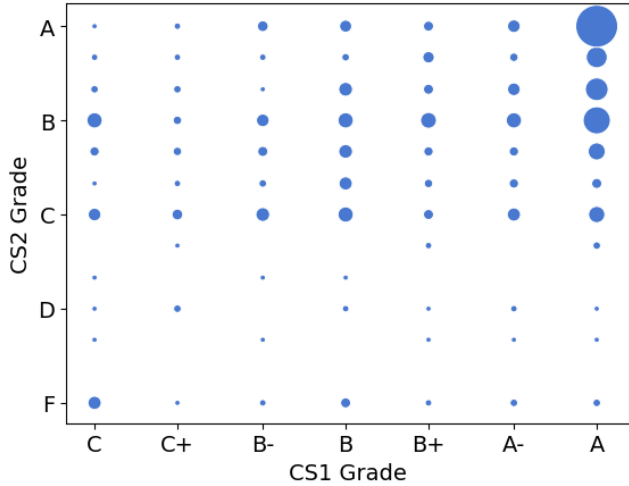
A more insightful analysis is to examine the *grade point difference* of individuals between CS1 and CS2 as shown in Figure 4. *Grade point difference* is calculated by subtracting CS1 grade points earned from CS2 grade points earned on the standard 4.0 scale. All students in this sample must have passed CS1 with at least a C (with many earning As), thus the distribution is biased to be skew left. The data indicates that most student grades (56%) dropped slightly. Only 18% improved their letter grades, and 26% earn the same grade bolstered by the 118 students who earned As in both courses. Figure 4 supports previous findings that CS1 grade is related to CS2 grade [5, 9] in the sense that 62.1% of grade point changes are within one standard deviation ( $\sigma = 0.949$ ). The change distribution is not normal (D'Agostino's  $K^2 = 67.29, p < 0.001$ ), possibly owing in part to the prerequisite grade of C in CS1.

Table 1 lists the letter grades earned in CS1 and the counts of students with those grades who then progress from CS2 by earning a grade of C or better. Over 97% of students who earn an A in CS1 also earn a C or better in CS2. However, of the students who earned a C in CS1, 27% do not earn a C or better in CS2. This

<sup>2</sup>We use the most recent grade, as opposed to an average grade across repeated attempts, to simplify the analysis.



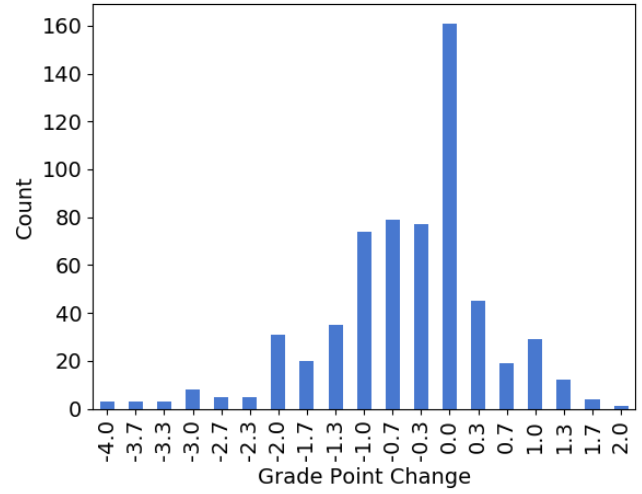
**Figure 2: CS1 and CS2 Course Letter Grade Distributions, Fall 2013–Spring 2019, CS1  $n = 1641$ , CS2  $n = 706$ . Percentages Are Shown to Normalize Values between Courses**



**Figure 3: Scatterplot of Students' Most Recent CS1 Grades vs. CS2 Grades,  $n = 614$ <sup>3</sup>**

amounts to only 14 students in our dataset, but they make up a disproportionate percentage of those who do not move on from CS2 ( $\chi^2 = 38.756, p < 0.001, df = 6$ ).

Those who do not progress from CS2 are of particular interest as they have demonstrated at least C-level competence in CS1. If the proportion of students who earned a C in CS1 remains disproportionately high, then we might consider whether a grade of C in CS1 measures the necessary aptitudes for success in the program. We are also interested in those students who performed excellently in CS1 (A and A-) but failed to earn a C in CS2. We do not have



**Figure 4: Grade Point Change from CS1 to CS2,  $n = 614$**

**Table 1: Proportions of Students Who Earned a C or Better in CS2 Grouped by CS1 Grade**

CS1 - Final Grade	C or better in CS2?		Total
	False	True	
C	14 (27%)	37 (7%)	51
C+	5 (10%)	24 (4%)	29
B-	4 (8%)	41 (7%)	45
B	9 (17%)	77 (14%)	86
B+	6 (11.5%)	51 (9%)	57
A-	6 (11.5%)	60 (11%)	66
A	8 (15%)	272 (48%)	280
Total	52	562	614

insights into what drives these students' performances yet, but that is the subject of future work.

## 5 SCS1 SCORE AS AN INDICATOR OF CS2 GRADE

The second research question we investigate is: *Does a quantitative relationship exist between CS2 course grade and a modified SCS1 concept inventory score?*

Our second research objective is to evaluate a modified version (explained below) of the *SCS1 concept inventory* [18] as a leading indicator of CS2 performance. The SCS1 is a validated instrument for evaluating students' understanding of select CS1 concepts. A relationship between our modified SCS1 (mSCS1) score and CS2 final grade may indicate that the mSCS1 can be used early in the CS2 semester to identify areas of weakness.

<sup>3</sup>The scatterplot shows students who earned grades in CS1 and CS2 at UNCW; students who earned CS1 credit at other institutions are excluded which is why the number of data points (614) is less than the number of CS2 grades collected (706).

## 5.1 The SCS1 and Modifications for this Study

*Note:* We are not at liberty to disclose the SCS1 or mSCS1 per the license of the SCS1 authors, but a request for free educator access to the SCS1 can be made via its Google Group located at <https://goo.gl/MiYOFk>.

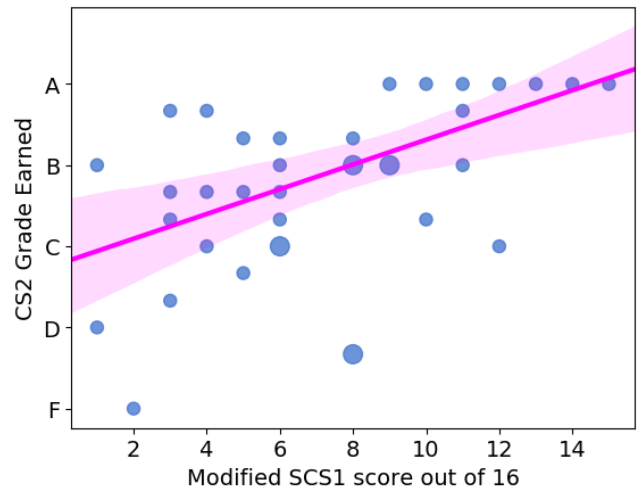
The Second CS1 Assessment (SCS1) [18] is derived from the Foundational CS1 (FCS1) Assessment [26]. The FCS1 was shown to have a significant, moderate correlation with CS1 Final Exam Scores [26, 27] in a large, multi-institutional, multi-programming language study. The SCS1 is an "isomorphic version" of the FCS1 available to educators while the FCS1 is copyrighted by its author. Both assessments are written in pseudocode and inventory the CS1 concepts of arrays, basics, for-loops, function parameters, function return values, if-else, logical operators, recursion, and while-loops. Each of the 27 questions has five multiple choice answers, and each topic has a definition-oriented, code tracing, and code completion question. Both the FCS1 and SCS1 were intended for a 60-minute exam period. We chose the SCS1 as a candidate for leading indicators because it is language agnostic, has been validated at scale, and is freely available to educators.

We modified the SCS1 for our study in response to community feedback on the SCS1 Google group and to fit our institutions CS1 learning outcomes and language (Python). Questions on recursion were dropped as recursion is not part of CS1 learning objectives at our institution. We translated the questions to Python as we found the pseudocode non-intuitive in places and did not wish to burden the students with interpreting the pseudolanguage. Finally, one question for each concept was dropped to encourage completion of the assessment in a 50-minute lecture session based on feedback from students and instructors in the FCS1 study [27] and on the SCS1 Google Group states that 60 minutes is not enough time to complete the original 27 questions. Questions with very low correct response rates (<20%) reported in [27] were dropped along with those that did not directly translate to Python.

Two sections of CS2 were administered the mSCS1 assessment consisting of 16 questions translated to Python. The assessment was given closed-book in a 50-minute class session. Scrap paper was permitted. A total of 44 students were administered the test, and 39 of those completed the semester.

## 5.2 Course Letter Grade and mSCS1 Score

Students' CS2 course grade points are plotted against their mSCS1 scores in Figure 5. D'Agostino's statistical test does *not* reject the null hypothesis that the mSCS1 scores are drawn from a normal distribution ( $K^2 = 2.794, p = 0.247$ ). However, the distribution of CS2 final grade is not normal ( $K^2 = 5.992, p = 0.049$ ). Spearman's non-parametric rank-order test yielded a moderate, statistically significant correlation ( $\rho = 0.568, p < 0.001$ ). This suggests that the written assessment may provide a leading indicator to overall performance in the course. The sample size is relatively small ( $n=39$ ), and we will replicate this analysis in future semesters. Future iterations of this work will examine student performance on the individual questions.



**Figure 5: Modified SCS1 Score vs. CS2 Grade Points Earned,  $n = 39$**

## 6 STUDENT REFLECTION QUESTIONNAIRE ON CS1 AND CS2

The final research question we investigate is: *What are the most challenging aspects of CS2, and how well does CS1 prepare students for CS2 from the student's perspective?*

In the final weeks of the semester, CS2 students were given a questionnaire that asked the following questions:

- (1) Rate your agreement with the following statement: "Overall, I think that CS1 (or equivalent) adequately prepared me for this course". (1-Strongly Disagree through 5-Strongly Agree)
- (2) Which concepts or topics from CS1 were most helpful to you in this course? (free text)
- (3) Which concepts or topics from CS1 would you like to have been better at prior to taking this course? (free text)
- (4) What were the most challenging aspects of this course? (free text)

Of the 79 students enrolled at the end of the semester, 69 (87%) completed the exit questionnaire. The questionnaire was not anonymous so that student responses could be linked to course performance; that analysis will be a part of future work, and we acknowledge that the lack of anonymity may alter the truthfulness of answers.

Of the 69 respondents, 59% agreed, 19% were neutral, and 22% disagreed with the statement that CS1 prepared them well for CS2. Ideally, student responses would be in the "agree" categories, but there are many personal [4, 17] and pedagogical reasons [29, 33] why this would not be the case irrespective of any disconnect between exit criteria from CS1 and starting expectations in CS2.

Responses to Questions 2–4 were transcribed into an Excel file and *open coded* [25] by the first author to identify topics/concepts in each response. A single response could mention multiple topics. Unique responses or responses that did not readily identify a topic or concern were not coded. Tables 2, 3, and 4 show codes (topics

and concepts) appearing in more than 10% of the responses to each question.

Question 2 asked, "Which concepts or topics from CS1 were most helpful to you in this course?" The student responses (Table 2) reflect the skills required to interact with data structures implemented as Python classes. Reading data from files, iterating over lists, and list manipulation are skills used in nearly every assignment. Data structures are implemented as Python classes with accompanying methods, e.g., a Stack class with `push()`, `pop()`, `top()`, etc. Thus, a strong grasp of function definition, call, and return is beneficial for students. Students who were exposed to recursion in CS1 found it to be useful when it is introduced in CS2. Question 3 asked, "Which concepts or topics from CS1 would you like to have been better at prior to taking this course?" The students' responses (Table 3) again reflect the focus of CS2 on Python classes, reading files for data content, and manipulating lists. File I/O, Python lists, and function definitions are covered extensively in our CS1 curriculum, whereas recursion and class definition are not part of the CS1 learning outcomes. The responses to Questions 2 and 3 are useful for informing discussions of what *could* be covered in CS1, but more importantly, these responses identify the concepts that should be reviewed, reinforced, and possibly quantified as potential leading indicators of performance at the beginning of CS2.

Question 4 asked "What were the most challenging aspects of this course? (open-ended)" concerning CS2. The responses to this question were predictably diffuse, but students did agree on some issues. Linked lists and its emphasis on pointers and class/object definition are challenging topics, reinforcing findings in Simon et al. [24] and Yeomans et al. [31]. Responses discussing the Linked

**Table 2: Questionnaire: Which Concepts or Topics from CS1 Were Most Helpful to You in this Course?**

Topic	Responses
Foundations (basic expressions, boolean values, syntax, vocabulary)	29 (43%)
Iteration, for-loops, or while-loops	25 (37%)
File manipulation (reading, writing, loading into a data structure)	15 (22%)
Recursion	15 (22%)
Lists and list operations	10 (15%)
Function call and definition	10 (15%)
if-else, conditionals	8 (12%)

**Table 3: Questionnaire: Which Concepts or Topics from CS1 Would You Like to Have Been Better at Prior to Taking this Course?**

Topic	Responses
Classes/objects	18 (26%)
File manipulation (reading, writing, reading into a data structure)	12 (17%)
Lists and list operations	11 (16%)
Recursion	11 (16%)
Dictionaries	8 (12%)

**Table 4: Questionnaire: What Were the Most Challenging Aspects of this Course?**

Topic	Responses
Linked list / doubly linked list	11 (16%)
Creating code from scratch	10 (15%)
Classes/objects	8 (12%)
Pace of instruction / work	7 (10%)

List and Class/Objects topics often reference challenges in translating the concepts into code. That is, students could understand the notion of Linked Lists and Objects but had difficulty implementing the data structures and applying them to problems. This likely relates to students' mental models of how the computer processes programs and points to a particular area where pedagogy can be improved. The "Creating Code from Scratch" topic refers to responses where students remarked on how past instructors required them to complete a half-finished Python script, but now they were required to create Python scripts from scratch or using a minimal code template. CS2 instructors should be aware of this paradigm. More importantly, student grades (particularly in CS2) may be predicated on their ability to decompose problems and initiate a solution with more independence than was required in the past. Thus, instructors may need to include these skills in course content and explicitly add them to course learning outcomes and assessments.

## 7 THREATS TO VALIDITY

We discuss this study's threats to validity according to the categories of Cook and Campbell [8].

### 7.1 Conclusion Validity

Conclusion validity refers to issues that affect the ability to associate treatment and outcome. *Low statistical power* is not a concern for our CS1-to-CS2 grade point comparison as the sample sizes are sufficiently large, however, the correlation between mSCS1 score and CS2 Grade Points Earned (Figure 5) had a sample size of 39, which is relatively low. Appropriate non-parametric statistical tests were used in all analyses as none of the data were normally distributed. The *reliability of the mSCS1* assessment has not been evaluated, and thus we cannot be certain if the outcomes of that assessment are replicable across contexts, especially outside of the Python language. Further, students' answers are likely influenced by the specific pedagogical style of the instructor, course content, and assignment content. The coding performed in this section was only performed by the first author due to time constraints, and thus interrater reliability is not available. We will continue administering the questionnaire in future semesters to improve the reliability of those responses.

### 7.2 Internal Validity

Internal validity concerns surround the causal inference between treatment and outcome should a relationship be detected. Some students took CS2 multiple times, introducing a potential *history effect* wherein their questionnaire answers and classroom performance



was perhaps different than those of students enrolled only one time. We do not isolate these students in our analyses and cannot quantify any such effects. Table 5 shows the number of repeats for each course; approximately 6% and 7% of students repeat CS1 and CS2 respectively. Only students' most recent grades are included in the calculation of Grade Change between CS1 and CS2 (Figure 4). Finally, our questionnaire in Section 6 was not anonymous, and thus students may not have been compelled to answer truthfully.

**Table 5: Individuals' Times Taken Excluding Withdrawals and Audits**

Times Taken	CS1 Count	CS2 Count
1	1434	609
2	91	44
3	7	3
4	1	-
Total	1533	656

### 7.3 Construct Validity

Construct validity concerns issues around whether the analysis results are connected to the driving theory, in our case, whether we adequately captured issues related to success in CS2 and the CS1-CS2 transition. One goal of our study (hopefully the first in a series) was to help better operationalize these constructs. Additional methods, such as semi-structured interviews and analyses of individual homework assignments and exam questions, will be used in the future to better capture which topics from CS2 are most challenging for students. Further, CS2 Grade is only one indicator of CS2 performance. In the future, we will also use the validated Basic Data Structures Inventory (BDSI) [21] to assess student understanding of CS2 concepts.

### 7.4 External Validity

External validity concerns the transferability of results outside the study context. We describe the content and environment of our CS1 and CS2 courses in Section 3, and it is up to the reader to decide if our Python-based courses resemble their own. One point of note is that our failure rate for CS1 is lower than that reported in other large scale studies, and thus our average CS1 grades may be inflated. No such data is available for CS2.

### 7.5 Conclusions and Future Work

Our long term research goal is to investigate *successful and unsuccessful transitions from CS1 to CS2*. As the first phase of that research, we provide a quantitative analysis of CS1 and CS2 grade point statistics—an evaluation of a modified version of the SCS1 concept inventory as an indicator of CS2 performance—and performed qualitative analysis of a questionnaire that obtained feedback from CS2 students on helpful CS1 preparation and challenges in CS2. In a sample of 614 students, we find that CS1 grades appear to have a statistical relationship with CS2 grades. This corroborates earlier findings [5, 9], but we provide a much larger sample size. *Further, the majority (56%) of students' CS2 grades were lower than*

*their CS1 grade, though 62% of the grade point changes were within one standard deviation*. A disproportionate number of C students from CS1 do not pass CS2 owing to the mean grade drop.

Our goal as educators is to ensure that students exiting CS1 have the necessary capabilities to succeed in CS2. To this end, we administered a modified SCS1 concept inventory [18] and correlated the scores with CS2 grade points earned. *We found a statistically significant correlation between mSCS1 score and CS2 grade points*, suggesting that the concepts on the mSCS1 and/or the way the mSCS1 tests those concepts measure useful information about beneficial skills in CS2. We believe, anecdotally, that strong performance on the mSCS1 requires solid mental models of *how* programs are interpreted by the machine. We believe that these mental models are central to both program debugging and abstraction. We will continue to investigate the individual questions and concepts captured in the mSCS1 for continued predictive power of CS2 grade and to identify possible improvements in CS1 structure.

From our questionnaire, the majority of students believed that CS1 adequately prepared them for CS2. The only data structure explicitly mentioned as challenging was *linked lists*. This may be a consequence of using Python as the programming language of choice. Unlike C/C++, Python (and Java) does not make the concept of pointers or memory references explicit. *Our findings are in line with studies have identified pointers and classes as threshold concepts* [23, 31] for students learning to program. Students indicated that coding a data structure to solve a problem, rather than filling in missing parts of an implementation, was challenging. This finding is congruous with previous papers who found that of *data structure application is a challenging addition to the finer points of data structure implementation* [20, 24].

A tremendous amount of computer science education research exists on success and failure factors in CS1, and a growing set of literature is being produced for CS2. We will continue to investigate cross-course indicators of success, specifically through more thorough investigations of the SCS1 inventory, the new Basic Data Structures Inventory [21], in-process code execution and class participation metrics (e.g., [6]), and qualitative feedback from the students themselves. We hope that we can improve students outcomes in CS2 (and beyond) through interventions driven by quantitative observation, and ensure that those students who commit to CS2 as part of a computing path receive the best opportunity to gain the skills and knowledge for career success.

### ACKNOWLEDGMENTS

Thanks to the students of CSC231 who contributed feedback to this study. Thanks also to Elham Ebrahimi, Clayton Ferner, Sridhar Narayan, Toni Pence, Geoffrey Stoker and other CS1 and CS2 instructors for their valuable participation and input in this study.

### REFERENCES

- [1] N. Alzahrani, F. Vahid, A. Edgcomb, K. Nguyen, and R. Lysecky. 2018. Python Versus C++: An Analysis of Student Struggle on Small Coding Exercises in Introductory Programming Courses. In *SIGCSE 2018 - Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. Association for Computing Machinery, Inc, Baltimore, MD, USA, 86–91. <https://doi.org/10.1145/3159450.3160586>
- [2] R. H. Austing, B. H. Barnes, D. T. Bonnette, G. L. Engel, and G. Stokes. 1979. Curriculum '78: Recommendations for the Undergraduate Program in Computer

- Science – A Report of the ACM Curriculum Committee on Computer Science. *Commun. ACM* 22, 3 (3 1979), 147–166. <https://doi.org/10.1145/359080.359083>
- [3] J. Bennedsen and M. E. Caspersen. 2007. Failure Rates in Introductory Programming. *ACM SIGCSE Bulletin* 39, 2 (6 2007), 32. <https://doi.org/10.1145/1272848.1272879>
  - [4] S. Bergin, R. Reilly, and D. Traynor. 2005. Examining the Role of Self-Regulated Learning on Introductory Programming Performance. In *Proceedings of the 1st International Computing Education Research Workshop, ICER 2005*. ACM Press, Seattle, WA, USA, 81–86. <https://doi.org/10.1145/1089786.1089794>
  - [5] H. Bisgin, M. Mani, and S. Uludag. 2018. Delineating Factors that Influence Student Performance in a Data Structures Course. In *2018 IEEE Frontiers in Education Conference FIE*. IEEE, San Jose, CA, USA, 1–9. <https://doi.org/10.1109/FIE.2018.8659300>
  - [6] A. S. Carter, C. D. Hundhausen, and O. Adesope. 2017. Blending Measures of Programming and Social Behavior into Predictive Models of Student Achievement in Early Computing Courses. *ACM Transactions on Computing Education* 17, 3 (8 2017), Article 12. <https://doi.org/10.1145/3120259>
  - [7] J. M. Cohoon. 2001. Toward Improving Female Retention in the Computer Science Major. *Commun. ACM* 44, 5 (2001), 108–114.
  - [8] T. D. Cook and D. T. Campbell. 1979. *Quasi-Experimentation: Design & Analysis Issues for Field Settings*. Rand McNally, Chicago.
  - [9] R. J. Enbody, W. F. Punch, and M. McCullen. 2009. Python CS1 as Preparation for C++ CS2. In *SIGCSE'09 - Proceedings of the 40th ACM Technical Symposium on Computer Science Education*. ACM Press, Chattanooga, TN, 116–120. <https://doi.org/10.1145/1508865.1508907>
  - [10] N. J. Falkner and K. Falkner. 2012. A Fast Measure for Identifying At-Risk Students in Computer Science. In *ICER'12 - Proceedings of the 9th Annual International Conference on International Computing Education Research*. ACM Press, Auckland, New Zealand, 55–62. <https://doi.org/10.1145/2361276.2361288>
  - [11] M. Hertz. 2010. What Do "CS1" and "CS2" Mean? Investigating Differences in the Early Courses. In *SIGCSE'10 - Proceedings of the 41st ACM Technical Symposium on Computer Science Education*. ACM Press, Milwaukee, WI, USA, 199–203. <https://doi.org/10.1145/1734263.1734335>
  - [12] K. Karpierz and S. A. Wolfman. 2014. Misconceptions and Concept Inventory Questions for Binary Search Trees and Hash Tables. In *SIGCSE 2014 - Proceedings of the 45th ACM Technical Symposium on Computer Science Education*. ACM Press, Atlanta, GA, USA, 109–114. <https://doi.org/10.1145/2538862.2538902>
  - [13] T. Koulouri, S. Lauria, and R. D. Macredie. 2014. Teaching Introductory Programming: A Quantitative Evaluation of Different Approaches. *ACM Transactions on Computing Education* 14, 4 (12 2014), Article 26. <https://doi.org/10.1145/2662412>
  - [14] R. P. Loui. 2008. In Praise of Scripting: Real Programming Pragmatism. *Computer* 41, 7 (7 2008), 22–26. <https://doi.org/10.1109/MC.2008.228>
  - [15] J. Margolis and A. Fisher. 2002. *Unlocking the Clubhouse: Women in Computing*. The MIT Press, Cambridge, Massachusetts.
  - [16] J. H. F. Meyer and R. Land. 2005. Threshold Concepts and Troublesome Knowledge (2): Epistemological Considerations and a Conceptual Framework for Teaching and Learning. *Higher Education* 49, 3 (4 2005), 373–388. <https://doi.org/10.1007/s10734-004-6779-5>
  - [17] I. O. Pappas, M. N. Giannakos, and L. Jaccheri. 2016. Investigating Factors Influencing Students Intention to Dropout Computer Science Studies. In *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*. ACM Press, Arequipa, Peru, 198–203. <https://doi.org/10.1145/2899415.2899455>
  - [18] M. C. Parker, M. Guzdial, and S. Engleman. 2016. Replication, Validation, and Use of a Language Independent CS1 Knowledge Assessment. In *Proceedings of the 2016 ACM Conference on International Computing Education Research (ICER '16)*. ACM, Melbourne, Australia, 93–101. <https://doi.org/10.1145/2960310.2960316>
  - [19] W. Paul and J. Vahrenhold. 2013. Hunting High and Low. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education - SIGCSE '13*. ACM Press, Denver, CO, USA, 29–34. <https://doi.org/10.1145/2445196.2445212>
  - [20] L. Porter, D. Zingaro, C. Lee, C. Taylor, K. C. Webb, and M. Clancy. 2018. Developing Course-Level Learning Goals for Basic Data Structures in CS2. In *SIGCSE 2018 - Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. ACM Press, Baltimore, MD, USA, 858–863. <https://doi.org/10.1145/3159450.3159457>
  - [21] L. Porter, D. Zingaro, S. N. Liao, C. Taylor, K. C. Webb, C. Lee, and M. Clancy. 2019. BDSI: A Validated Concept Inventory for Basic Data Structures. In *Proceedings of the 2019 ACM Conference on International Computing Education Research - ICER '19*. ACM Press, Toronto, ON, Canada, 111–119. <https://doi.org/10.1145/3291279.3339404>
  - [22] A. Radenski. 2006. "Python First": A Lab-Based Digital Introduction to Computer Science. In *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education 2006*. ACM Press, Bologna, Italy, 197–201. <https://doi.org/10.1145/1140124.1140177>
  - [23] K. Sanders, J. Boustedt, A. Eckerdal, R. McCartney, J. E. Moström, L. Thomas, and C. Zander. 2012. Threshold Concepts and Threshold Skills in Computing. In *ICER'12 - Proceedings of the 9th Annual International Conference on International Computing Education Research*. ACM Press, Auckland, New Zealand, 23–30. <https://doi.org/10.1145/2361276.2361283>
  - [24] B. Simon, M. Clancy, R. McCartney, B. Morrison, B. Richards, and K. Sanders. 2010. Making Sense of Data Structures Exams. In *ICER'10 - Proceedings of the International Computing Education Research Workshop*. ACM Press, Aarhus, Denmark, 97–105. <https://doi.org/10.1145/1839594.1839612>
  - [25] A. Strauss. 1987. *Qualitative Analysis for Social Scientists*. Cambridge University Press, New York. <https://doi.org/10.1017/CBO9780511557842>
  - [26] A. E. Tew. 2010. *Assessing Fundamental Introductory Computing Concept Knowledge in a Language Independent Manner*. Ph.D. Dissertation. Georgia Institute of Technology.
  - [27] A. E. Tew and M. Guzdial. 2011. The FCS1: A Language Independent Assessment of CS1 Knowledge. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education (SIGCSE '11)*. ACM, Dallas, TX, USA, 111–116. <https://doi.org/10.1145/1953163.1953200>
  - [28] E. H. Turner, E. Albert, R. M. Turner, and L. Latour. 2007. Retaining Majors through the Introductory Sequence. In *SIGCSE 2007: 38th SIGCSE Technical Symposium on Computer Science Education*. ACM Press, Covington, KY, USA, 24–28. <https://doi.org/10.1145/1227310.1227321>
  - [29] A. Vihavainen, J. Airaksinen, and C. Watson. 2014. A Systematic Review of Approaches for Teaching Introductory Programming and their Influence on Success. In *ICER 2014 - Proceedings of the 10th Annual International Conference on International Computing Education Research*. ACM Press, Glasgow, Scotland, 19–26. <https://doi.org/10.1145/2632320.2632349>
  - [30] C. Watson and F. W. Li. 2014. Failure Rates in Introductory Programming Revisited. In *ITiCSE 2014 - Proceedings of the 2014 Innovation and Technology in Computer Science Education Conference*. ACM Press, Uppsala, Sweden, 39–44. <https://doi.org/10.1145/2591708.2591749>
  - [31] L. Yeomans, S. Zschaler, and K. Coate. 2019. Transformative and Troublesome? Students' and Professional Programmers' Perspectives on Difficult Concepts in Programming. *ACM Transactions on Computing Education* 19, 3 (1 2019), Article 23. <https://doi.org/10.1145/3283071>
  - [32] D. Zingaro. 2015. Examining Interest and Grades in Computer Science 1: A Study of Pedagogy and Achievement Goals. *ACM Transactions on Computing Education* 15, 3 (7 2015), Article 14. <https://doi.org/10.1145/2802752>
  - [33] D. Zingaro, C. Taylor, L. Porter, M. Clancy, C. Lee, S. N. Liao, and K. C. Webb. 2018. Identifying Student Difficulties with Basic Data Structures. In *ICER 2018 - Proceedings of the 2018 ACM Conference on International Computing Education Research*. ACM Press, Espoo, Finland, 169–177. <https://doi.org/10.1145/3230977.3231005>

# Cry Wolf: Toward an Experimentation Platform and Dataset for Human Factors in Cyber Security Analysis

William Roden

University of North Carolina, Wilmington  
Wilmington, North Carolina, USA  
roden011@gmail.com

Lucas Layman

University of North Carolina, Wilmington  
Wilmington, North Carolina, USA  
laymanl@uncw.edu

## ABSTRACT

Computer network defense is a partnership between automated systems and human cyber security analysts. The system behaviors, for example raising a high proportion of false alarms, likely impact cyber analyst performance. Experimentation in the analyst-system domain is challenging due to lack of access to security experts, the usability of attack datasets, and the training required to use security analysis tools. This paper describes Cry Wolf, an open source web application for user studies of cyber security analysis tasks. This paper also provides an open-access dataset of 73 true and false Intrusion Detection System (IDS) alarms derived from real-world examples of *impossible travel* scenarios. Cry Wolf and the impossible travel dataset were used in an experiment on the impact of IDS false alarm rate on analysts' abilities to correctly classify IDS alerts as true or false alarms. Results from that experiment are used to evaluate the quality of the dataset using difficulty and discrimination index measures drawn from classical test theory. Many alerts in the dataset provide good discrimination for participants' overall task performance.

## CCS CONCEPTS

• **Security and privacy** → **Usability in security and privacy**; *Intrusion detection systems*.

## KEYWORDS

cyber security, human factors, IDS

### ACM Reference Format:

William Roden and Lucas Layman. 2020. Cry Wolf: Toward an Experimentation Platform and Dataset for Human Factors in Cyber Security Analysis. In *2020 ACM Southeast Conference (ACMSE 2020)*, April 2–4, 2020, Tampa, FL, USA. ACM, New York, NY, USA, 4 pages. <https://doi.org/10.1145/3374135.3385301>

## 1 INTRODUCTION AND BACKGROUND

Computer network defense is a partnership between humans and machines. Machines are necessary to process voluminous network data, and humans must investigate suspicious behaviors identified on the network. Intrusion Detection Systems (IDSes) inspect network activity for known attack signatures, violations of user-configured rules, and statistical anomalies [6]. IDSes then alert human cyber analysts of suspicious activity. However, IDS systems

often produce false alarm rates above 95%, requiring cyber analysts to evaluate hundreds or thousands of false alarms [10]. This behavior may lead to vulnerabilities at the analyst-system interface; research shows that low true event probability and high false alarm rate reduce human operator performance [2, 3]. While researchers strive to improve the accuracy of IDSes, comparatively little research has been published on how the *behaviors* of IDSes and other semi-automated cyber security systems impact the performance of human cyber analysts.

Experimentation on cyber security analyst performance is challenging. Cyber security professionals are difficult to access due to the sensitivity of their work. Further, security-focused datasets that may be used to simulate real attacks in an experimental environment (e.g., [4]) are often raw network captures and system logs rather than cyber defense tool output, and training study participants on real IDS and network monitoring systems is intractable. Consequently, few quantitative experiments have been published on analyst performance in cyber security tasks (e.g., [1, 7]).

This paper presents two open-access resources that help address the need for controlled experimentation of cyber analyst performance. Section 2 describes an IDS alarm dataset derived from real true and false alarms from the University of North Carolina, Wilmington (UNCW)'s IDSes. Section 3 introduces the open source Cry Wolf web application wherein users evaluate alarms from the dataset, answer a questionnaire on user expertise, and reflect on the task. Finally, Section 4 presents an initial evaluation of the dataset's quality using results from a controlled experiment conducted using the Cry Wolf platform.

## 2 THE IMPOSSIBLE TRAVEL DATASET

The first resource is a dataset of simulated IDS alerts derived from real *impossible travel* alarms. Impossible travel alerts are triggered when a user authenticates from two geographic locations within a period where physical travel between the two locations is impossible, e.g., authentications from London and Moscow with a time between authentications of 30 minutes. Physical travel in this time frame is impossible, but the authentications may be legitimate through technical means such as a Virtual Private Network (VPN).

The dataset contains *true alarms* where the impossible travel alert warrants further investigation for potential malicious activity, and *false alarms* where the alert is not cause for concern. The dataset contains 73 alerts in all: 30 true alarms and 43 false alarms and is available at <https://uncw-hfcs.github.io/ids-simulator-analysis/>. These particular numbers of alerts were needed for the experiment introduced in Section 4 and fully reported in [13]. Each alert contains the following data:

ACMSE 2020, April 2–4, 2020, Tampa, FL, USA

© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *2020 ACM Southeast Conference (ACMSE 2020)*, April 2–4, 2020, Tampa, FL, USA, <https://doi.org/10.1145/3374135.3385301>.

- **Cities of Authentication** — The two geographic locations from which the IDS detected an authentication.
- **Number of Successful Logins** — The number of successful authentications from each location in the past 24 hours.
- **Number of Failed Logins** — The number of failed logins from each location in the past 24 hours.
- **Source Provider** — The type of internet provider the authorizations came from at each location. Possible values are:
  - **Telecom** — traditional Internet Service Providers
  - **Mobile/cellular** — wireless carriers
  - **Hosting/server** — hosted service providers, e.g., VPNs, web hosts, and cloud computing
- **Time between Authentications** — The shortest time between authentications from the two cities in the past 24 hours. Reported in decimal hours. This is the field that triggers an alarm in a real IDS.
- **VPN Confidence** — A percent likelihood that the user utilized a VPN.

Examples are provided in §2.2. These data fields were chosen based on the first author's experience as a security operations analyst at UNCW. These fields were most often considered when deciding whether an impossible travel alert was normal from the university IDSes. The data values for events were based on real samples from UNCW's IDSes with random noise added. Location pairs were generated from combinations of 12 notable cities.

## 2.1 Evaluating Alerts – the Security Playbook

A "Security Playbook" accompanies the dataset and guides how to use the data fields to evaluate whether an alert is a true alarm or false alarm. The guidance reflects how an expert familiar with IDS behavior and the network being protected would evaluate the alerts. The playbook may be viewed at <https://git.io/JvE0I>. This section summarizes the main elements of the playbook as articulated to the reader.

One of the cities in each alert corresponds to legitimate access. Every city has a *concern level* based on a history of attacks: Moscow and Beijing are "High" concern cities, North American cities are "Low" concern, and all others are "Medium" concern. However, users travel legitimately and hosted services may connect from other countries, thus location should not be the sole deciding factor.

*Time between authentications* is the field that triggered the alert. Authentications from different locations in a short time could be an indication of account compromise. The Security Playbook provides a table of typical travel times between all pairs of locations. The *ratio of successful logins to failed logins* from each location may also indicate malicious activity. More failed logins than successful logins could be an indication of password guessing, but legitimate users may fail to authenticate as well, e.g., by using an expired password.

The *source providers* help interpret the other information. A telecom provider implies the login is originating from a user physically at the location, whereas a mobile/cellular or hosting/server provider does not necessarily mean the user is physically present. There is nothing inherently safe or malicious about any type of source provider. The *VPN confidence* percentage is the likelihood that the authentication attempts were made via a VPN.

The Security Playbook contains a "Things to Keep in Mind" section with additional considerations. This section reminds evaluators that IDSes can be inaccurate and that a few, many, or all of the alerts may be false alarms. Another consideration is that users visiting countries with restrictive governments will often use a VPN to circumvent that nation's firewall. Finally, the section states that it is not unusual for a mobile device to ping the country in which the mobile device is registered when the user is traveling abroad.

## 2.2 Scenarios

The impossible travel dataset covers seven scenarios encountered in real alerts from UNCW's IDSes.

**2.2.1 True Impossible Travel.** The dataset contains 19 impossible travel alerts that are true alarms. An example is shown in Table 1. In these alerts, the *time between authentications* is less than the time required for a person to travel between the locations as provided in the Security Playbook, and the *source providers* are set to *telecom* to indicate that the persons attempting the authentications are in those physical locations.

**Table 1: Event #66 – A True Alarm with Password Guessing**

City of Authentication	Seattle	Moscow
# Successful Logins	4	11
# Failed Logins	1	3
Source Provider	Telecom	Telecom
Time between Authentications	1.75	
VPN Confidence	0%	

**2.2.2 Password Guessing.** The dataset contains six true alarms where the number of *failed logins* from one or both cities exceeds the number of *successful logins* from that city. In contrast, the ratio of failed-to-successful logins is less than 1.0 in the rest of the dataset.

**2.2.3 Edge Case Travel.** The dataset contains 15 false alarms where the *time between authentications* is within 20 minutes of the typical time between the two cities. The Security Playbook emphasizes the travel time table shows "typical times" and not minima or maxima. All of the cities are "low concern" and the *source providers* are either "telecom" or "mobile/cellular" to encourage focus on the times.

**2.2.4 Eurotrip.** The dataset contains six false alarms with the same features as the "edge case travel" scenarios except that both cities are in Europe. All European cities are of "medium concern", which may lead some evaluators to escalate these scenarios.

**2.2.5 Mobile.** The dataset contains eight false alarms of authentication from a mobile device. Table 2 shows an example. A mobile device may initially route through its home country when traveling abroad as mentioned in the Security Playbook. For all the mobile scenario alerts, the *source provider* is "mobile/cellular" from a city in the USA, and the other city's *source provider* is "telecom". The idea is the user authenticates from a computer abroad while their mobile device is occasionally pinging their home wireless service.

**Table 2: Event #18 — A False Alarm from Mobile Usage**

City of Authentication	Miami	London
# Successful Logins	3	12
# Failed Logins	2	0
Source Provider	Mobile/Cellular	Telecom
Time between Authentications	0.90	
VPN Confidence	0%	

**2.2.6 VPN.** The dataset contains seven false alarms stemming from users employing VPN services in a location separate from the user's physical location. The *VPN confidence* for these alerts is >90%. One of the cities is "high concern" and has a *telecom* source provider to indicate the user is physically present. The other city is low or medium concern and has a *hosting/service* source provider that is intended to be the VPN service.

**2.2.7 Hosting/servers.** The dataset contains 12 false alarms that entail using a server or hosting service. These alerts involve only low or medium concern cities, and one of the *source providers* is "hosting/server" while the other is "telecom" to indicate the user is geographically situated in one place. The *VPN confidence* values are between 53-71% to distinguish these alarms from the higher confidence VPN scenario with high and medium concern cities.

### 3 THE CRY WOLF PLATFORM

The Cry Wolf web application provides a simulated environment for evaluating IDS alarms from the impossible travel dataset. Cry Wolf is written in the Flask micro-framework for Python. Source code and screenshots of Cry Wolf are available at <https://uncw-hfcs.github.io/ids-simulator/>. The webapp implements an experiment to evaluate the impact of IDS *false alarm rate (FAR)* on analyst performance in correctly classifying alerts as true or false alarms. The webapp provides the following experimentation structure:

- (1) A login page with experiment description and Institutional Review Board information. User logins are assigned by a human proctor, and participants are placed into a 50% or 86% FAR treatment group based on the login name.
- (2) A questionnaire that captures participant expertise in cyber security and networking for use in performance analysis.
- (3) A training page introducing the experiment's scenario, the Security Playbook, and five training alerts with rationale.
- (4) The main alert evaluation task, which displays a table of alerts and links to the alert details. The details are presented similarly to the examples shown in Tables 1–2.
- (5) A post-survey that administers the NASA Task Load Index questionnaire [9] and self-reflection questions on the IDS alert evaluation for use in analysis.

In part (4), the participant reviews the alert data against the Security Playbook and chooses to "escalate" or "don't escalate" the alert, which classifies the alert as a true or false alarm respectively. An analysis script compares participants' answers against an oracle of whether the alerts were true or false alarms, and generates a confusion matrix to derive classification performance measures of

sensitivity, specificity, and precision. The repository of analysis scripts is <https://github.com/uncw-hfcs/ids-simulator-analysis>.

The Cry Wolf platform has several benefits for user experimentation. Nearly all of the experimental procedure is automated in the application; the only intervention required by the experimenter is to obtain informed consent and assign a pre-generated login code. The platform captures precise data on when the participants view and classify each alert, enabling fine-grained analysis of time-on-task. Subjects can exit and resume the experiment as necessary using their assigned logins, and the web application can be deployed on a cloud server for maximum availability. Cry Wolf is open source, and experimenters knowledgeable of Python and web development can adapt it to variations of the experimental structure. Further details the Cry Wolf platform structure and implementation are provided in [13].

### 4 EVALUATION OF THE IMPOSSIBLE TRAVEL DATASET

This section presents an initial evaluation of the quality of the impossible travel dataset for use in controlled experimentation of cyber analyst performance. In Fall 2019, 51 individuals participated in an experiment using the Cry Wolf platform. The goal of that experiment was to evaluate the impact of IDS *false alarm rate* on analysts' abilities to *correctly classify IDS alerts as true or false alarms*. The participants were randomly assigned to one of two treatment groups: 25 participants were treated with a 50% false alarm rate (25 true and 25 false alarms), and 26 participants were treated with an 86% false alarm rate (seven true and 43 false alarms). Participants were shown alerts from the dataset and classified each as a true or false alarm. The experiment was not time-limited, and the median time to complete the classification of all 50 alerts was 15.6 minutes. The experiment, its initial findings, and threats to validity are fully reported in [13]. The remainder of this section examines the quality of the impossible travel dataset using data from that experiment.

#### 4.1 Measures of Dataset Quality

Classical test theory provides two measures to evaluate the quality of individual alerts: the alert's *difficulty index* and *discrimination index* [5]. These measures are traditionally used to evaluate the quality questions in psychometric and educational tests.

An alert's *difficulty index*,  $p$  is the *proportion* of participants who correctly classified an alert as a true or false alarm to the total number of people who classified the alert. Lord suggests a target of  $p \approx 0.85$  for items with a binary response to maximize validity of the overall test while accounting for random guessing [12]. An alert with  $p = 1.0$  indicates that the correct classification may be obvious to the participants, whereas an alert with a  $p \ll 0.85$  may indicate that the alert data is insufficient or misleading.

An item's *discrimination index*,  $D$ , measures the difference in responses to the item between high- and low-performers on the overall task. Each participant's number of correctly classified alerts is counted. Two groups are formed from the 27% of participants with the highest and lowest scores to minimize chance error [11]. To calculate an alert's  $D$ , subtract the number correct in the low group from the number correct in high group, then divide the difference by the size of the larger group. Values of  $D$  are in the range  $[-1.0, 1.0]$ .

**Table 3: Item Difficulty and Discrimination Index Statistics per False Alarm Rate (FAR) Treatment**

	50% FAR		86% FAR	
	$p$	$D$	$p$	$D$
mean	0.75	0.30	0.76	0.41
std	0.22	0.30	0.20	0.35
min	0.29	-0.29	0.16	-0.29
$Q_1$	0.60	0.00	0.62	0.14
$Q_2$ (median)	0.76	0.29	0.77	0.43
$Q_3$	0.95	0.57	0.92	0.71
max	1.00	0.86	1.00	1.00
participants	25		26	
true alarms	25		6	
false alarms	25		44	

Ebel states that items where  $D > 0.40$  are useful discriminators,  $0.20 \leq D \leq 0.40$  are in need of improvement, and  $D < 0.20$  are poor discriminators and should be eliminated or rewritten [8].

## 4.2 Analysis

Table 3 shows the item difficulty and discrimination index statistics calculated per treatment group. Item difficulty ( $p$ ) scores are similar across groups, which suggests the dataset is reliable across treatments. The mean and median discrimination index scores ( $D$ ) are in the range warranting improvement per Ebel [8]. The negative min  $D$  value shows that at least one alert is misleading because low performers answered it correctly more than high performers.

Table 4 bins the alerts according to their  $p$  and  $D$  scores. Approximately 40% of the alerts evaluated by each group have a good discrimination index per Ebel [8]. Approximately 30% of the alerts fall into the "too easy" category where many of the participants correctly classified the alert. In the Cry Wolf experiment, unlike in educational tests, such alerts are useful because they help build and reinforce the participants' notions of true and false alarms.

The alarms that are "too hard" are cause for concern as they may indicate that the Cry Wolf training exercises are insufficient or the alerts are unclear. Two *eurotrip* alerts have travel times that are nearly impossible but are false alarms – likely the participants were erring on the side of caution. Four alerts from the *mobile* scenario fell into the "too hard" category for the 50% FAR group. One possible explanation is that the participants did not read or remember the line near the end of the Security Playbook stating that mobile devices abroad often ping their home city first. The other alerts in the "too hard" group exhibit no obvious pattern. The

**Table 4: Aggregate Measures of Item Analysis. Values are the Number of Alerts**

	50% FAR	86% FAR
$D > 0.4$ (best)	24	27
$p \geq Q_3$ and $D \leq 0.4$ (too easy)	13	17
$p < Q_2$ and $D \leq 0.4$ (too hard)	7	2

$D$  score is sensitive to random noise given the small sizes of the high and low groups ( $n = 7$  in each group) for each treatment.

Overall, the difficulty and discrimination indices suggest that the impossible travel dataset is useful to discriminate between high- and low- performers. Further qualitative investigation on why the "too hard" alerts were mis-classified is warranted.

## 5 CONCLUSION AND FUTURE WORK

This paper presents a dataset of simulated IDS alarms, introduces the Cry Wolf platform for controlled experiments of cyber analyst performance, and provides an initial evaluation of the dataset's quality. The impossible travel dataset and Cry Wolf platform are open source, and the evaluation shows promising results for discriminating between high and low performers.

In future work, the impossible travel dataset will be expanded to include new alarm types. The Cry Wolf platform will be used in Summer 2020 to study the effects of *anchoring bias* [14] on cyber analyst performance. These studies are part of a research plan to investigate factors impacting cyber analyst performance. The long term goals of this research are to develop new training methodologies for coping with imperfect cyber defense systems, provide guidance for tool developers on user interface considerations, and develop defense strategies against analyst-machine vulnerabilities.

## REFERENCES

- [1] N. Ben-Asher and C. Gonzalez. 2015. Effects of Cyber Security Knowledge on Attack Detection. *Computers in Human Behavior* 48 (7 2015), 51–61. <https://doi.org/10.1016/j.chb.2015.01.039>
- [2] J. P. Bliss and M. C. Dunn. 2000. Behavioural Implications of Alarm Mistrust as a Function of Task Workload. *Ergonomics* 43, 9 (9 2000), 1283–1300. <https://doi.org/10.1080/001401300421743>
- [3] S. Breznitz. 1984. *Cry Wolf: The Psychology of False Alarms*. Lawrence Erlbaum Associates, Hillsdale, NJ.
- [4] Canadian Institute for Cybersecurity. 2019. Datasets. <https://www.unb.ca/cic/datasets/index.html>
- [5] L. Crocker and J. Algina. 1986. *Introduction to Classical and Modern Test Theory*. Wadsworth/Thomson Learning, Belmont, CA, USA.
- [6] D. E. Denning. 1987. An Intrusion-Detection Model. *IEEE Transactions on Software Engineering* 13, 2 (1987), 222–232. <https://apps.dtic.mil/dtic/tr/fulltext/u2/a484998.pdf>
- [7] V. Dutt, Y.-S. Ahn, N. Ben-Asher, and C. Gonzalez. 2012. Modeling the Effects of Base-Rates on Cyber Threat Detection Performance. In *Proceedings of the 11th International Conference on Cognitive Modeling (ICCM 2012)*. Universitätsverlag der TU, Berlin, Berlin, Germany, 88–93.
- [8] R. L. Ebel. 1979. *Essentials of Educational Measurement* (3rd ed.). Prentice-Hall, Inc., Englewood Cliffs, NJ.
- [9] S. G. Hart and L. E. Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research. *Advances in Psychology* 52, C (1 1988), 139–183. [https://doi.org/10.1016/S0166-4115\(08\)62386-9](https://doi.org/10.1016/S0166-4115(08)62386-9)
- [10] K. Julisch. 2003. Clustering Intrusion Detection Alarms to Support Root Cause Analysis. *ACM Transactions on Information and System Security* 6, 4 (2003), 443–471. <https://doi.org/10.1145/950191.950192>
- [11] T. L. Kelley. 1939. The Selection of Upper and Lower Groups for the Validation of Test Items. *Journal of Educational Psychology* 30, 1 (1 1939), 17–24. <https://doi.org/10.1037/h0057123>
- [12] F. M. Lord. 1952. The Relation of the Reliability of Multiple-Choice Tests to the Distribution of Item Difficulties. *Psychometrika* 17, 2 (1952), 181–194. <https://doi.org/10.1007/BF02288781>
- [13] W. T. Roden. 2019. *An Empirical Study of Factors Impacting Cyber Security Analyst Performance in the Use of Intrusion Detection Systems*. Master's thesis. University of North Carolina, Wilmington, Wilmington, NC.
- [14] A. Tversky and D. Kahneman. 1974. Judgment under Uncertainty: Heuristics and Biases. *Science* 185, 4157 (9 1974), 1124–31. <https://doi.org/10.1126/science.185.4157.1124>

# Toward Reducing Fault Fix Time: Understanding Developer Behavior for the Design of Automated Fault Detection Tools

Lucas Layman, Laurie Williams, Robert St. Amant  
*Department of Computer Science*  
*North Carolina State University, Raleigh, NC, USA*  
*lmlayma2@ncsu.edu, {williams, stamant}@csc.ncsu.edu*

## Abstract

*The longer a fault remains in the code from the time it was injected, the more time it will take to fix the fault. Increasingly, automated fault detection (AFD) tools are providing developers with prompt feedback on recently-introduced faults to reduce fault fix time. If, however, the frequency and content of this feedback does not match the developer's goals and/or workflow, the developer may ignore the information. We conducted a controlled study with 18 developers to explore what factors are used by developers to decide whether or not to address a fault when notified of the error. The findings of our study lead to several conjectures about the design of AFD tools to effectively notify developers of faults in the coding phase. The AFD tools should present fault information that is relevant to the primary programming task with accurate and precise descriptions. The fault severity and the specific timing of fault notification should be customizable. Finally, the AFD tool must be accurate and reliable to build trust with the developer.*

## 1. Introduction

Long *fault fix latency*, the time between fault injection and fault removal, could substantially increase the cost of fixing a fault. Research [6, 7, 19] indicates that the time a developer requires to fix a fault is positively correlated with *ignorance time* – the time between fault injection and the point at which the developer becomes consciously aware of the details of a reported fault. Increasingly, automated fault detection (AFD) tools provide developers with prompt feedback on recently-introduced faults, thereby reducing ignorance time. AFD tools examine source code using static and/or dynamic analysis techniques

to uncover potential faults in the code. Studies have shown that the use of AFD tools can increase software quality and developer productivity [28, 29]. Some examples of AFD tools are FindBugs [18], Check ‘n Crash [9], Continuous Testing [29], and the continuous compilation in integrated development environments (IDE) such as Eclipse.

Ideally, we want the developer to act upon an *alert*, the notification of a potential fault, as soon as it is displayed. However, alerts that are provided but not acted upon may be an indication that the alerts are being produced too often, are not informative, and/or may be distracting to the developer. Systems that automatically volunteer information can degrade rather than improve performance if their behavior is not closely matched to user needs and expectations; users may begin to view such systems as a constantly-ringing alarm clock and simply ignore them [23].

Typically, a developer will pick a certain point in a programming task to suspend his or her thoughts and investigate a fault. *The goal of this paper is to explore what factors are used by developers to decide whether or not to address a fault when notified.* These factors can be used to guide the design of intelligent fault notification systems that integrate AFD tools with programming environments to reduce ignorance time.

A controlled study was conducted with 18 developers of varying programming experience to discover why developers interrupt a programming task to debug a fault. The study participants performed several programming tasks in the Eclipse IDE. During the programming task, the IDE notified the participants that a potential fault was found in the code. The participants were then asked to discuss the decision factors that weighed on whether to address the alerts or not. The study sessions were audio recorded, transcribed and coded for analysis. Several themes emerged from our grounded theory [13]

approach to the qualitative analysis of the participant responses.

The remainder of this paper is organized as follows: Section 2 provides related work on memory and task interruption, Section 3 discusses the details of the study setup and execution, Section 4 contains the analysis of the participants' responses, Section 5 contains conjectures for further quantitative studies into usable design of automated fault notification systems during code development, and we provide conclusions and future work in Section 6.

## 2. Related work

In considering when to notify a developer of a potential fault, we address two fundamental areas that underlie our research: cognitive processing and task interruption. Understanding how and why an interruption can interfere with a working task provides a valuable starting point for examining developer interruption in a coding environment.

### 2.1. Interruptions and cognitive processing

Human attention is recognized to have a limited capacity. Limited cognitive resources require humans to be selective about the information they process [2]. Limitations in human memory and attention result in any interruption having the potential to cause *interference* with a working task. An interruption interferes with a working task by consuming cognitive resources initially used by the working task [11, 25, 27]. The amount of resources a task uses in the brain is the *cognitive load* of that task. The degree to which an interrupting task interferes with a primary task is dependent on several factors:

- the cognitive loads of the working and interrupting tasks [12, 26]
- the similarity of the two tasks [12]
- personal attributes of the developer [3]
- attributes of the tasks (e.g. complexity) [4, 5].

The cognitive load of debugging tasks varies according to task complexity and developer experience [1, 8, 33]. Interrupting a complex task with a complex debugging task may result in destructive interference, whereas interrupting a low complexity task with a low complexity debugging task may cause no interference. For example, debugging a recursive algorithm while in the midst of implementing a tightly coupled method may result in significant interference between tasks. However, there may not be interference if a developer needed to insert a semicolon in another line while writing a print statement.

### 2.2. Interruptions in human-computer interaction and decision theory

Human-computer interaction (HCI) studies have shown that the similarity of the interrupting task to the primary task and the interrupting task's complexity can affect user performance in environments that support multiple activities [4, 5, 10, 31]. Design guidelines for systems where user attention may be divided between multiple activities [17, 24] have also been published. McFarlane [22] asserts that a negotiated interruption style, where the system alerts the user but does not force attention away from the primary task is best in terms of user performance in most situations.

Horvitz has studied decision theory for using information about developer goals to guide the decision of an intelligent notification system [15, 16]. He describes that an agent takes action based on *utility* as a function of action or inaction given what the system can infer about a user's goals. Horvitz labels the critical threshold of action versus inaction as  $p^*$ . In this study, we are investigating what factors may contribute to a  $p^*$  value that defines the threshold between the user addressing an alert or not.

## 3. Study description

This section describes a controlled study of developers working with an AFD system, the Automated Warning Application for Reliability Engineering (AWARE) [14, 30]. To achieve the goal of our study, we examined factors that cause a developer to interrupt the task at hand and devote time to investigate a fault. All study materials, including the example program, task descriptions, interviewer scripts, and transcriptions and coding may be found at [http://agile.csc.ncsu.edu/aware/research/resources\\_LWS07.zip](http://agile.csc.ncsu.edu/aware/research/resources_LWS07.zip).

### 3.1. AWARE

AWARE is a plug-in for the Eclipse IDE that runs third-party AFD tools. AWARE also estimates the severity of a fault and ranks the fault according to the likelihood that it is *not* a false positive. For a more thorough discussion of AWARE, please see [30]. A screenshot of AWARE can be seen in Figure 1. The AWARE display shows a list of faults initially ordered by true positive probability. Each fault in the list contains the following information in order:

- a description of the problem, such as "possible null pointer" or "uninitialized variable"



- the folder, class file, and the line number at which the fault was detected in the code
- the probability at the fault is a true positive
- the severity of the fault from 1-3 with 3 being the most severe

The version of AWARE used in this study did not incorporate fault analysis tools, but instead displayed seeded fault notifications at scheduled times.

At the beginning of the study, AWARE's fault notification was not attracting the attention of the participants. AWARE was changed so that the fault notification window would change from white to yellow whenever a fault appeared. Six of the 18

subjects participated in the study before this change was made. Some participants still did not notice the fault notification after the change was made due to their engrossment in the programming tasks. We cannot provide a reliable analysis of any systematic difference in the responses of the two groups due to the variability in the responses. Anecdotally, we observe that more participants noticed the fault notifications after the yellow background change, but did not necessarily begin debugging more than the group prior to the interface change.

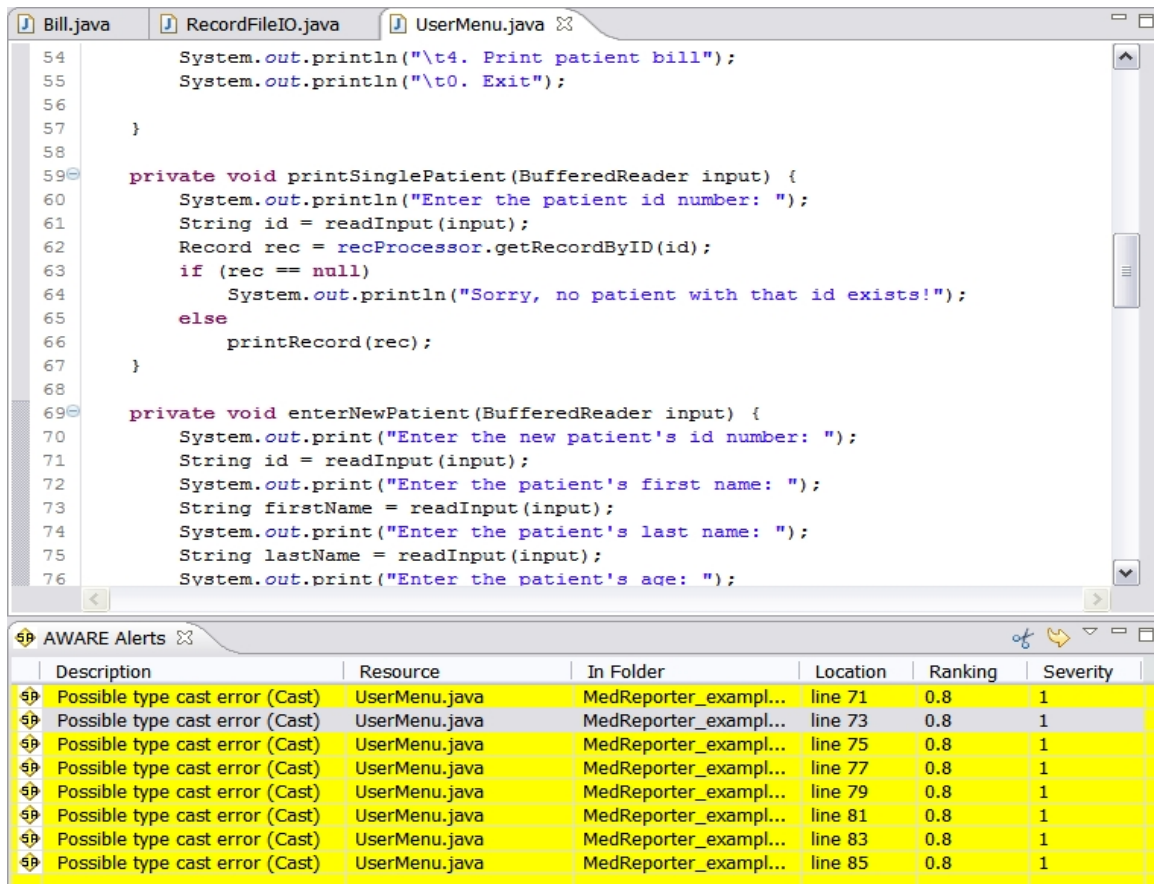


Figure 1. AWARE in the Eclipse IDE (cropped image)

### 3.2. Pre-study analysis

To guide our study, we needed to identify some potential factors that may cause a developer to interrupt a working task to address an alert. A literature search yielded little information on this topic, and so we performed a task analysis of developer behavior while using an advanced IDE that

displays alerts from AFD tools. The analysis was conducted with only one subject (the first author) and yielded a behavior model similar to Latorella's general model of task interruption [23]. The analysis yielded several factors that, in combination, may contribute to a developer's decision on when to address an alert. These factors were: a) the complexity of the primary programming task; b) the relevance of the fault to

primary task; c) the estimated cost of fixing the fault in terms of time; and d) the potential criticality of the fault as estimated by the system-assigned priority of the fault notification.

### 3.3. Study participants

Participants were solicited from the North Carolina State University Department of Computer Science through a graduate student mailing list and by posting fliers throughout the computer science building. A \$20 gift certificate to a location of the participant's choice was offered as incentive to participate in the study. The requirements to participate in the study were a working knowledge of Java and object-oriented programming, participation in a 45 minute live study session, and consent to be audio recorded. No experience with AFD tools or IDEs was necessary.

All participants completed an online survey to sign up for the study. The online survey collected the participants' contact information, gender, and times available for the live portion of the study. The survey also collected each participant's years of programming, Java, IDE, and professional development experience. Participants were also prompted to list any IDEs they may have used. Finally, the survey asked for the study participants' response on a scale from 1 to 9 in confidence in solving programming problems (1 = not confident, 9 = very confident) and their enjoyment of coding in general (1 = I hate coding, 9 = I love coding). In total, 28 survey responses were collected.

Twenty subjects participated in the study and the other eight missed their appointments. Of the 20 participants, one session was aborted because the participant had no Java experience, and one session was discarded because of problems with AWARE. Thus, 18 live sessions were used in the study analysis. The programming experience responses of the 18 participants are summarized in Table 1.

**Table 1. Subjects counts - years of experience**

	Programming	Java	IDE	Professional
None	0	0	2	5
0-1	0	6	5	3
1-3	3	5	3	6
3-5	8	4	4	1
5-10	4	3	2	3
>10	3	0	2	0

At the beginning of each session, the participant was asked to rate his or her fatigue at that time on a scale from 1-10 with 1 being rested and 10 being completed exhausted. The fatigue rating, programming confidence and coding enjoyment of the 18 participants is summarized Table 2. This information was used in assigning participants to the different treatments for the experiment, discussed in Section 3.5.

**Table 2. Subject counts - miscellaneous**

	1	2-3	4-6	7-8	9+
Programming confidence	0	1	6	11	0
Coding enjoyment	0	2	1	10	5
Fatigue	2	6	9	1	0

In general, the participant sample was widely distributed over the survey questions, though the sample size was too small to perform a statistical analysis. The limitations in using this sample of participants are discussed in Section 4.5.

### 3.4. Study programming tasks

The bulk of the live session required the participants to complete four programming tasks while interacting with AWARE. The programming tasks required participants to modify and to add to an existing example program – a simple medical reporting and billing system written by the first author. The example program was designed to be easily comprehended and contained enough classes and functionality (seven classes, 413 LoC) to simulate cognitively complex programming and debugging tasks.

The programming tasks were created to help determine what criteria a developer uses when deciding to interrupt their programming task to address an alert. At a pre-determined time after the start of each programming task, AWARE would alert the participants that a fault had been detected. These faults were purposefully injected into the example program beforehand. The faults and associated alerts exhibited all of the properties suggested by the pre-study analysis (see Section 3.2). All of the faults were designed to be relevant to the current programming task; that is, the fault would directly impair the proper functionality of the programming task. The faults also had a high criticality and could crash the example program. Finally, the faults required non-trivial investigations to uncover the root of the fault, thus increasing the developer effort required to fix the fault.

### 3.5. Study procedure

In the main portion of the study, participants met individually with the investigator (the first author) in a private meeting room to perform programming tasks and discuss alerts. These sessions were comprised of five parts.

**3.5.1. Part 1: Introduction.** To provide some context to the session, the investigator explained that the purpose of the study was to examine how developers interacted with advanced IDE environments. No further detail was provided. The participants were then given a brief demonstration of Eclipse and AWARE on a research laptop. Study participants were shown a sample program to demonstrate how Eclipse compiles the source code every time a file is saved and displays any resulting compiler errors or warnings. The study participants were then told AWARE works in a similar fashion, but uses different tools to find different types of faults. The subjects were also told that AWARE's analysis takes more time and runs in the background, so the timing of the alert displays was unpredictable.

**3.5.2. Part 2: Familiarization.** Since the participants were working on an unfamiliar program and using unfamiliar tools, they were given several familiarization tasks to reduce any learning effects. First, the participants ran the example program and used several of its features, including printing out patient data and entering patient information. Second, the subjects performed an informal code walkthrough of the same features to familiarize them with the general architecture of the example program.

**3.5.3. Part 3: Example tasks.** Two example programming tasks (as discussed in Section 3.4) were given to familiarize the participants with the main tasks in Part 4 of the study. The participants were given a written requirement to modify a feature in the example program. The participants were told that they must completely implement the requirement and correct all errors detected by AWARE, but that the ordering of these activities was unimportant. Finally, the participants were instructed to "think out loud" to verbalize their thoughts to the investigator while working on the task. The subjects were told that they will work on the task until it is completed or until stopped by the investigator.

The investigator began audio recording as the subjects commenced on the programming task. The participants were stopped by the investigator approximately one minute after the AWARE fault

notification was displayed. This one minute window allowed the investigator to observe whether or not the subject chose to interrupt the main programming task to address the alert. The investigator also noted the subject's start time, the time of interruption and any observations about the subject's behavior at the time of the alert. The participants performed two example programming tasks. Both of the programming tasks in Part 3 were injected with faults that were more trivial to fix than in the Part 4 of the study.

**3.5.4. Part 4: Main tasks.** This portion of the study involved two programming tasks with differing complexities. The simpler task required finding and changing numerical values in the code, and the more complex task involved making changes to several coupled methods. Again, AWARE displayed a fault notification at a scheduled time during each task and the same procedures were followed as in Part 3.

To reduce the effect of the ordering of the programming tasks, the subjects were divided into two groups that had similar numbers of students with IDE experience and varying degrees of programming experience. One group performed the more complex task first, and the other performed the simpler task first. However, due to the variability of the subjects' data, neither the ordering of the programming tasks nor the experience data were used in our analysis.

After the investigator stopped the subjects on the programming task, the participants were asked to explain their rationale for either addressing an alert or ignoring it. The investigator prodded the participants to continue explaining their rationale until they had no more information to share. Some participants commented that they did not notice the alerts at all.

**3.5.5. Part 5: Exit interview and debriefing.** After the programming portion had been completed, the participants were asked to postulate on any additional factors that might influence their decision to address an alert or not. Participants were asked to think of scenarios where they would stop working on a programming task to address an alert and scenarios where an alert would be deferred until later. After the study, the participants were thanked and given a more detailed explanation of the study's purpose.

## 4. Analysis and findings

The audio recordings from the 18 study participants were transcribed by the first author and combined with notes taken by the investigator during the recording sessions, yielding approximately 60 pages of information. The transcriptions were then

coded by the first author. Coding is the process that categorizes qualitative data into different themes via three steps: open coding, axial coding, and selective coding [32]. Open coding is the process of identifying the categories in the data and the properties of the different categories. Axial coding is used to connect the categories and find their interrelationships. In the last step, selective coding identifies one or two central categories and forms a conceptual framework. Typically, coding should be performed by multiple persons to ensure the reliability of the analysis. Resource constraints prevented more than one person performing the coding, and we accept this limitation since our study is exploratory and designed to help guide future work rather than draw final conclusions.

The coding process yielded 37 distinct themes organized into seven categories dealing with task interruption and fault assessment:

1. Strategies – describe developer behavior as relates to addressing faults
2. Fault assessment criteria – the factors used by developers to determine whether or not to interrupt the primary task to address an alert
3. Interruption points – specifically when in time the primary task will be interrupted
4. Environment – influences created by the programming environment itself
5. Individual differences – attributes of the developers
6. Perspectives – the impacts of developer understanding of the example program or AWARE tool that influenced interruptions
7. External influences – factors related to the experimental setup that influenced developer behavior

Once the themes were identified, a count was made of the number of participants who mentioned a particular theme. Those themes which were mentioned by five or more subjects are discussed below. A complete list of all 37 themes grouped by category may be found in Appendix A of Layman, et al. [21].

#### 4.1. Fault assessment criteria

The primary purpose of this study was to assess what factors would contribute to a developer interrupting their workflow to address a fault presented during the coding process. The attributes of the fault itself are critical components in the developer's decision to interrupt. Study participants identified several of these fault assessment criteria.

Nine participants commented that *the description of the fault* was critical in assessing the importance of a

fault. The fault description contained information about the nature of the fault, such as whether it was a potential null pointer exception, and array index out of bounds, or an uninitialized variable. For example, one subject noted, “the main thing that I’m going to look at is null pointer exceptions ... Something should not happen that could cause the entire program to crash – that is what I would look at first.” Speaking on the fault description, another subject observed “I wasn’t using the ranking and severity as much as I was using my own programming experience and instinct in deciding whether to inspect that error or not.”

Nine participants used the *ranking and severity* of the AWARE fault notifications as part of their fault assessment criteria. When AWARE displayed an alert during a programming task, one subject had the following reaction, “Array index too large – what’s this? Line 10: RecordProcessor.getSize(). I don’t see a reason why... oh, severity is 3, ranking is 0.9. Oh okay, so this could definitely be a problem.” In general, it appeared that the subjects used the fault ranking and severity when they did not assess the importance of the fault from description and personal experience alone. The subjects may also have been primed to look at this information due to the introduction of the AWARE tool earlier in the experiment.

Another important assessment criterion was the *relevance of the fault to the code currently being written*. When asked why she addressed an alert immediately, one subject responded, “Well it seemed connected to my problem. I’m losing some data, so I’m trying to figure out – maybe it’s not been initialized here.” Oftentimes, subjects stated that they were quick to dismiss faults that did not seem relevant to their current task. “If it’s something that’s not really relevant to what I’m doing now, I’m going to go back and finish what I was doing,” said one participant. The criteria for assessing the relevance of a fault to the current task varied from subject to subject. Some participants spoke on a high level about related tasks, while others specifically stated that they would address alerts in the current class file.

#### 4.2. Interruption points

Determining *when* to notify the developer of a fault is of commensurate importance to understanding *why* a developer would interrupt. Many subjects noted that they would *interrupt the primary programming task after they finished a thought*. When an alert popped up during a programming task, one participant stated, “I’ve got to finish this thought, but I see the warning there.” When asked why he deferred addressing a

fault, one subject responded, “I wanted to finish what I was doing and then investigate afterwards. I don’t want to lose my current train of thought of what I was working on.”

These statements reflect current theories of mental task management and task switching. When given the choice, people will tend to switch between tasks only at a convenient breaking point between high level mental tasks and not between low level details [10, 24]. One subject remarked, “If I had some logic in my head, maybe an if-statement with a lot of different attributes, different things that I wanted to get out of my head and onto the code, I would have done that before I interrupted.” These observations also go to the heart of our motivation for this study: while fault notifications may be beneficial during development, they should be done with care so as not to impede the mental workflow of programming.

Other subjects more precisely defined their interruption points. Many participants interrupted themselves after completing *the current line of code*. For some subjects, finishing the line of code was a convenient stopping point. Others wanted to finish the line of code to determine if the alerts were the result of an incomplete piece of code. For example, one subject observed, “I figured I am not done with [the code] yet, so once I might finish, the error might disappear, which happens a lot with Eclipse.”

Other subjects interrupted only between *sections of code*, which in some cases was an extended version of finishing a line to see if alerts go away: “Instead of fixing the line every time, [fix them] every now and then after just 20 lines or 30 lines. After 30 lines I can fix them and see these are the probable errors.” In other cases, finishing a section of code seemed to coincide with completing a thought. Said one participant, “Let’s say I figure out certain logic, I want to finish that and then see what the problem with it is.”

The variations in where to interrupt the programming task, whether at the end of a line or at the end of a code section, may derive from the complexity of the current programming task. Though the programming tasks were designed with varying complexities to test the importance of primary task complexity, the variability of the data precludes a more rigorous analysis.

### 4.3. Environment and perspectives

Several themes arose related to the participants’ general interactions with AWARE and Eclipse. These themes are grouped into two categories: Environment and Perspectives. While the themes in these categories do not always directly involve fault

interruption and interaction, the themes do present some important design implications.

Five of the study participants expressed that they needed to *trust the fault detection system*. Trust was earned in the form of accurate, reliable fault information. Many of the participants had several years of programming and tool experience. This experience led them to distrust some analysis tools because of poor accuracy, and these participants placed higher values on their own assessments of potential faults. According to one subject, “If I used [AWARE] regularly, and I saw this ranking of 1.0... If I did it say, twice, and each time it was 1.0 and it was definitely something that was an error, then I think I would definitely, certainly start looking at this.”

Other subjects were intrinsically interested in AWARE’s fault information because it inspired them “to think of something as potentially an error that I hadn’t thought of when I previously developed.” Both of these perspectives suggest that both developer experience and familiarity with the code may play an important role in the usage of AFD tools.

Another emergent category involved the difficulties some subjects had in interpreting the fault information. Some subjects *incorrectly believed that a fault was the cause of something directly related to the code* they were typing, when the actual cause of the fault was rooted elsewhere. Six subjects made such mistakes, though the investigator did not reveal these mistakes to them at any point. This mischaracterization of a fault was often the result of developer expectations: the participant was developing code that was incomplete and thus was expecting a fault to be detected. Then, by chance, an alert was displayed referring to a separate portion of the code. The subject then drew the conclusion that the coding and fault were related when in fact they were not. The reverse of this scenario happened four times when participants believed that a fault was *not* related to programming task. Similarly, six participants *could not make the connection between the fault and the programming task*. These participants observed the fault and investigated the source line but could not understand the problem enough to correct it. In some cases, the participants stated that they could not discern what variable or statement the fault description referenced. These problems may be symptomatic of the version of AWARE used in this experiment, which contained less precise descriptions of the faults. The aforementioned themes stress the need for concise and accurate fault information.

#### 4.4. Individual differences

The individual differences of the developers have some bearing on the use of the AWARE system. Six of the participants expressly stated that they were very interrupt-driven, and that when something pops up, they tend to address it right away. One subject stated, “Every time I get a new mail icon, I’ll just stop whatever I’m doing to go check. I’m just that type of person.” The same subject later added rationale to the interrupt-driven personality while programming, “I guess any time I see errors or warnings I try to go and address those before I do something new because they might have a ripple effect.” A developer’s proclivity for interruption may make the usage of an AFD system more challenging since they may be more prone to the destructive interference caused by interrupting tasks.

#### 4.5. Study limitations

The primary limitations of this study are external validity limitations concerned with the sample population and the study environment. Limitations regarding the changing of the AWARE environment and the coding procedure have been discussed in Sections 3.1 and 4 respectively.

All 18 subjects were drawn from a student population (though some had professional experience) and thus the results of this study may not generalize to professionals. Similarly, because of the controlled and time-limited nature of the experiment, we could not reproduce the project complexities and environmental factors of the professional workplace. Therefore, the responses of the sample subjects may not reflect the diversity of professional developers in a professional setting. However, since we are using this study to provide conjectures and to form a basis for future study, we do not believe that these limitations significantly diminish our findings.

Some experimental validity concerns arose during the study. With a few of the subjects, a Hawthorne effect may have been present. Since they had been told about the capabilities of AWARE, they purposefully waited for alerts to appear and may have investigated the alerts when they would not have under normal, unobserved programming conditions. Also, some subjects did not notice the alerts until they were asked by the investigator if they observed the alerts. For those subjects who did not initially notice the alerts, a learning effect occurred wherein they noticed the alert on the next task. However, while these subjects did subsequently *notice* the faults, they

did not necessarily interrupt the primary task to investigate them.

#### 5. Conjectures

Based on our analysis, we identify several conjectures to guide future quantitative research on the integration of AFD systems with IDEs to reduce fault ignorance time.

*Conjecture 1: Fault descriptions should be as informative and precise as possible.*

At least half of the subjects used the fault description to assess the importance of the fault and weighed on the decision to interrupt the programming task. Furthermore, some subjects had difficulty in identifying the exact location of a fault because of the imprecise nature of some fault descriptions.

*Conjecture 2: System-assigned fault severity should reflect the developer’s perceptions of fault severity.*

Developer assessment of fault severity was often subjectively based on the fault description. Developer’s perceptions of fault severity varied between subjects. Therefore, the system-assigned fault severity should be customizable (based on fault type) so that the AFD systems can more accurately estimate a developer’s decision to interrupt a programming task.

*Conjecture 3: Fault information should be presented when the fault is relevant to the current programming task.*

Creating a mechanism to assess the relevance of a fault to the developer’s current working context will be difficult. However, the relevance of the fault is central to some developer’s decisions to interrupt the programming task. The location of the fault relative to the currently active line of code, coupling between code sections and data and control flow analysis may provide avenues for estimating relevance.

*Conjecture 4: The point at which the AFD tool notifies the developer should be customizable by the developer.*

The developer should have ultimate authority in deciding when alerts occur. The developers can customize the interruptions to be displayed to suit their personal preferences, which may increase both the effectiveness and the perceived usefulness of the AFD tool. Some developers may wish to only be notified of faults at the end of typing a programming

statement, while others may only wish to know of certain classes of errors such as null pointers.

*Conjecture 5: The developer must trust that the fault information from the AFD tool is accurate and reliable.*

If the developer cannot trust the accuracy of the fault information provided by the AFD tool, the utility of the tool will drop significantly and may be ignored entirely. Trusting the accuracy of the tool seems to be particularly important when the developer is not familiar with the code. However, accurately identifying faults can be problematic for tools that employ static analysis, which is known to generate high false positive rates [20]. In AWARE, each detected fault is provided with a probability that the fault is a true positive. Concurrent research on AWARE is investigating techniques to improve the accuracy of the true positive probability. Several subjects believed that some faults were the result of incomplete code. Therefore, deferring fault notifications until a source statement is complete may increase trust in the system.

## 6. Conclusion and future work

By leveraging the fault detection power of AFD tools and integrating them with code development, developers can reduce the ignorance time of faults identified by AFD tools and lower the cost-of-fix of these faults. If these tools are to be utilized by developers, they must be of value in terms of both information and usability. Programming is a complex cognitive process, and developers must be notified of fault information carefully to avoid valueless disruption. We performed a controlled case study to better understand how to create an intelligent interface between the developer and AFD tools. Our study revealed several important factors that contribute to a developer's decision to interrupt a programming task to debug a fault when using AFD tools.

We have provided five conjectures to guide further study on developer switches from programming to debugging tasks. We will use the findings of this study to guide the design and refinement of AWARE's alert system. We will investigate an intelligent system for estimating developer's fault assessment criteria for identifying which faults are of most importance to the developer, and we will observe developers' actual decision criteria in live use of the system. We will also incorporate customizable notification options and learning algorithms based on developer interactions with AWARE to help refine its facilities. Our ultimate goal is to investigate

empirically the impact on fault fix latency and cost-of-fix when AFD tools are integrated with IDEs to reduce ignorance time.

## Acknowledgements

The authors would like to thank the North Carolina State University software engineering RealSearch group for their helpful comments on this paper. This work is supported by the National Science Foundation under the Grant No. 00305917. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

## References

- [1] B. Adelson, D. Littman, K. Ehrlich, J. Black, and E. Soloway, "Novice-Expert Differences in Software Design," proceedings of Human-Computer Interaction (INTERACT '84), 1984, pp. 187-192.
- [2] A. Allport, "Visual Attention," in *Foundations of Cognitive Science*, M. I. Posner, Ed. Cambridge, MA: The MIT Press, 1989, pp. 631-682.
- [3] J. W. Atkinson, "The Achievement Motive and Recall of Interrupted and Completed Tasks," *Journal of Experimental Psychology*, vol. 46, 1953, pp. 381-390.
- [4] B. P. Bailey, J. A. Konstan, and J. V. Carlis, "The Effects of Interruptions on Task Performance, Annoyance, and Anxiety in the User Interface," proceedings of Human-Computer Interaction (INTERACT 2001), Tokyo, Japan, 2001, pp. 593-601.
- [5] B. P. Bailey, J. A. Konstan, and J. V. Carlis, "Measuring the Effects of Interruptions on Task Performance in the User Interface," proceedings of IEEE International Conference on Systems, Man, and Cybernetics 2000 (SMC '00), Nashville, TN, 2000, pp. 757-762.
- [6] W. Baziuk, "BNR/NORTEL: Path to improve product quality, reliability, and customer satisfaction," proceedings of International Symposium on Software Reliability Engineering, Toulouse, France, 1995, pp. 256-262.
- [7] B. W. Boehm, *Software Engineering Economics*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1981.
- [8] R. Brooks, "Towards a Theory of the Comprehension of Computer Programs," *International Journal of Man-Machine Studies*, vol. 18, 1983, pp. 543-554.
- [9] C. Csallner and Y. Smaragdakis, "Check 'n' Crash: Combining Static Checking and Testing," proceedings of

International Conference on Software Engineering (ICSE '05), St. Louis, MO, 2005, pp. 422-431.

[10] E. Cutrell, M. Czerwinski, and E. Horvitz, "Notification, Disruption, and Memory: Effectors of Messaging Interruptions on Memory and Performance," proceedings of Human-Computer Interaction (Interact 2001), Tokyo, Japan, 2001, pp. 263-269.

[11] M. B. Edwards and S. D. Gronlund, "Task Interruption and its Effects on Memory," *Memory*, vol. 6, 1998, pp. 665-687.

[12] T. Gillie and D. Broadbent, "What Makes Interruptions Disruptive? A Study of Length, Similarity, and Complexity?" *Psychological Research*, vol. 50, 1989, pp. 243-250.

[13] G. Glaser and L. Anselm, *The Discovery of Grounded Theory: Strategies for Qualitative Research*, Aldine de Gruyter, Chicago, IL, 1967.

[14] S. Heckman, "AWARE Research Home Page," <http://agile.csc.ncsu.edu/aware>, accessed January 11, 2007.

[15] E. Horvitz, "Principles of Mixed-Initiative User Interfaces," proceedings of Computer-Human Interaction (CHI '99), Pittsburgh, PA, 1999, pp. 159-166.

[16] E. Horvitz, A. Jacobs, and D. Hovel, "Attention-sensitive Alerting," proceedings of Conference on Uncertainty and Artificial Intelligence (UAI '99), Stockholm, Sweden, 1999, pp.

[17] E. Horvitz, C. Kadie, T. Paek, and D. Hovel, "Models of Attention in Computing and Communication: From Principles to Applications," *Communications of the ACM*, vol. 46, 2003, pp. 52-59.

[18] D. Hovemeyer and B. Pugh, "Finding Bugs is Easy," *ACM SIGPLAN Notices*, vol. 39, 2004, pp. 92-106.

[19] W. S. Humphrey, *A Discipline for Software Engineering*, Addison Wesley, Reading, MA, 1995.

[20] T. Kremenek, K. Ashcraft, J. Yang, and D. Enger, "Correlation Exploitation in Error Ranking," proceedings of International Symposium on Foundations of Software Engineering (ISESE '04), Newport Beach, CA, 2004, pp. 83-93.

[21] L. Layman, L. Williams, and R. S. Amant, "Toward Reducing Fault Fix Time: Understanding Developer Behavior for the Design of Automated Fault Detection Tools, the Full Report," North Carolina State University, Raleigh, NC, TR-2007-2, January 15, 2007.

[22] D. C. McFarlane, "Comparison of Four Primary Methods for Coordinating the Interruption of People in

Human-Computer Interaction," *Human-Computer Interaction*, vol. 17, 2002, pp. 63-139.

[23] D. C. McFarlane and K. A. Latorella, "The Scope and Importance of Human Interruption in Human-Computer Interaction Design," *Human-Computer Interaction*, vol. 17, 2002, pp. 1-61.

[24] Y. Miyata and D. A. Norman, "Psychological Issues in Support of Multiple Activities," in *User Centered System Design: New Perspectives on Human-Computer Interaction*, D. A. Norman and S. W. Draper, Eds. Hillsdale, NJ: Lawrence Erlbaum Associates, 1986, pp. 267-284.

[25] D. A. Norman, "Categorization of Action Slips," *Psychological Review*, vol. 88, 1981, pp. 1-15.

[26] D. A. Norman and D. G. Bobrow, "On Data-limited and Resource-limited Processes," *Cognitive Psychology*, vol. 7, 1975, pp. 44-64.

[27] M. I. Posner and A. F. Konick, "On the Role of Interference in Short-term Retention," *Journal of Experimental Psychology*, vol. 72, 1966, pp. 221-231.

[28] D. Saff and M. D. Ernst, "An Experimental Evaluation of Continuous Testing during Development," proceedings of International Symposium on Software Testing and Analysis (ISSTA '04), Boston, MA, 2004, pp. 76-85.

[29] D. Saff and M. D. Ernst, "Reducing Wasted Development Time via Continuous Testing," proceedings of International Symposium on Software Reliability Engineering (ISSRE '04), Denver, CO, 2004, pp. 281-292.

[30] S. E. Smith, L. Williams, and J. Xu, "Expediting Programmer AWAREness of Anomalous Code," proceedings of International Symposium on Software Reliability Engineering (ISSRE '05), Chicago, IL, 2005, pp. 4.49-4.50.

[31] C. Speier, J. S. Valacich, and I. Vessey, "The Effects of Task Interruption and Information Presentation on Individual Decision Making," proceedings of International Conference on Information Systems (ICIS '97), Atlanta, GA, 1997, pp. 21-36.

[32] A. L. Strauss and J. M. Corbin, *Basics of Qualitative Research: Techniques and Procedures of Developing Grounded Theory*, Second ed., Sage Publications, Thousand Oaks, CA, 1998.

[33] W. Visser and J. M. Hoc, "Expert Software Design Strategies," in *Psychology of Programming*, J.-M. Hoc, T. R. G. Green, R. Samurcay, and D. J. Gilmore, Eds. New York, NY: Harcourt Brace Jovanovich, 1990, pp. 235-249.



**(Best Paper Award)**


---

## Technical Debt: Showing the Way for Better Transfer of Empirical Results

Forrest Shull, Davide Falessi, Carolyn Seaman, Madeline Diep, and Lucas Layman

---

### Abstract

In this chapter, we discuss recent progress and opportunities in empirical software engineering by focusing on a particular technology, Technical Debt (TD), which ties together many recent developments in the field. Recent advances in TD research are providing empiricists the chance to make more sophisticated recommendations that have observable impact on practice.

TD uses a financial metaphor and provides a framework for articulating the notion of tradeoffs between the short-term benefits and the long-term costs of software development decisions. TD is seeing an explosion of interest in the practitioner community, and research in this area is quickly having an impact on practice. We argue that this is due to several strands of empirical research reaching a level of maturity that provides useful benefits to practitioners, who in turn provide excellent data to researchers. The key is providing observable benefit to practitioners, such as the ability to tie technical debt measures to business goals, and the ability to articulate more sophisticated value-based propositions regarding how to prioritize rework. TD is an interesting case study in how the maturing field of empirical software engineering research is paying dividends. It is only a little hyperbolic to call this a watershed moment for empirical study, where many areas of progress are coming to a head at the same time.

---

F. Shull (✉) • D. Falessi • C. Seaman • M. Diep • L. Layman  
 Fraunhofer Center for Experimental Software Engineering, 5825 University Research Court,  
 Suite 1300, College Park, MD 20740-3823, USA  
 e-mail: [fshull@fc-md.umd.edu](mailto:fshull@fc-md.umd.edu); [dfalessi@fc-md.umd.edu](mailto:dfalessi@fc-md.umd.edu); [cseaman@fc-md.umd.edu](mailto:cseaman@fc-md.umd.edu);  
[mdiep@fc-md.umd.edu](mailto:mdiep@fc-md.umd.edu); [llayman@fc-md.umd.edu](mailto:llayman@fc-md.umd.edu)

## 1 Introduction

Software engineering is an exceedingly dynamic field. Since the term “software engineering” was coined at the 1968 NATO conference, the field has seen an explosion in terms of the number of applications and products that use software, an immense growth in the sophistication and capabilities of those products, multiple revolutions in the way software relates to the hardware and networks over which it runs, and an ever-changing set of technologies, tools, and methods promising more effective software development.

Similarly and not surprisingly, the field of empirical software engineering has been dynamic as well. In the decades in which empirical studies have been performed, we have seen evolutions in the objects of study, study methodologies, and the types of metrics used to describe those study objects. Also, as with software engineering in the large, empiricists have seen our own fads come and go, with different types of studies being introduced, becoming (over-)popular, and then settling into a useful niche in the field. Articles elsewhere in this book [1] have reflected on some of these trends. In this chapter, we discuss some recent progressions and opportunities in the area of empirical software engineering by focusing on a particular technology, Technical Debt, which ties together many recent developments in the field. We use Technical Debt to discuss recent advances that are providing empiricists the chance to make more sophisticated recommendations and to have more of an impact on practice. We also use this concept as a launching point to look at how some of these recent progressions may extend into the future.

---

## 2 Technical Debt: What Is It?

Before describing the opportunities that TD brings to empirical software engineering, let’s try to understand what TD is. First coined in 1992 [2], the underlying ideas come from the mid-1980s and are related to Lehman and Belady’s notion of software decay [3] and Parnas’ software aging phenomenon [4]. TD is a metaphor, and while it lacks a formal definition, it can be seen as “the invisible results of past decisions about software that negatively affect its future” [5]. The reference to financial “debt” implies that the notion of tradeoffs between short-term benefits and long-term costs is central to the concept.

Because TD is a metaphor, it can be applied to almost any aspect of software development, encompassing anything that stands in the way of deploying, selling, or evolving a software system or anything that adds to the friction from which software development endeavors suffer: test debt, people debt, architectural debt, requirement debt, documentation debt, code quality debt, etc. [6]. Research into TD often bears a superficial resemblance to earlier empirical work on defect identification; TD identification often takes the form of identifying deficiencies in software development artifacts (requirements, architecture, code, etc.).

While TD research builds upon many of the empirical lessons learned from defect identification, we argue that TD research introduces an important new dimension into empirical studies of software defects and software quality: context-dependent short-term versus long-term quality tradeoffs. Consider the progression of empirical work in software quality:

- *Approximating quality via defect counts:* Many studies of software quality have used defect counts as a proxy for a technique's impact on software quality. For example, in studies of software V&V methods, it is often assumed that the more defects identified by a technique, the bigger the resulting improvement to software quality that can result from its application. Some examples include [7, 8].
- *Value-Based Software Engineering:* It was always recognized that defect counts were simply a proxy for software quality, but the Value-Based Software Engineering (VBSE) paradigm gave the community more tools required for a sophisticated view of the problem. VBSE articulated the idea that value propositions in software development need to be made explicit, so that software engineers can determine whether stakeholder values are being met and, indeed, whether they can be reconciled [9]. Studies reflecting a VBSE point of view tend to weight defects differently in terms of their severity for different stakeholders, or according to the operational scenarios under which those defects would be detected. In short, the studies were designed on the assumption that the true impact on quality can vary greatly from one defect to another.
- *Quality is relative in time and context:* Work on TD extends the VBSE considerations even further. First, in TD, the issues needing rework are more tightly coupled to the team's specific quality goals. For example, deficient documentation may not represent a "defect" in the sense that it will lead to incorrect software, but this deficiency may represent TD for teams that prioritize reuse and maintainability. In contrast, deficient documentation would not be considered as TD by a team that is developing a throw-away prototype. Second, TD instances do not automatically represent deficiencies in the system; rather they represent a tradeoff that was made in order to achieve some other short-term goal. TD may even be healthy in the short term, such as trading off a perfectly maintainable design to add quickly a feature needed immediately by an important customer. The TD metaphor stresses that other considerations need to be taken into account by teams contemplating rework of a TD issue, such as how much effort it would take to correct that instance and how much it is "costing" to have that instance in the system.

TD research has inherited much from prior generations of empirical studies that looked at software quality: approaches to counting discrete instances of items for potential rework (whether they be instances of defects or TD); the need for taxonomies to categorize those instances and provide insights regarding root causes; and the goal of characterizing various manual and automated techniques in terms of the number and type of instances that they uncover.

TD encapsulates new aspects of empiricism by providing **a context-dependent way of thinking about software quality across lifecycle phases, and in a way**

**tractable to quantitative analysis and hence objective observations.** Thus, the primary contribution of TD to the empirical community is useful guidance for: (1) analyzing the cost tradeoffs of software engineering decisions; and (2) effectively transmitting the results of empirical research to practitioners by recognizing the existence and management of these tradeoffs.

---

### 3 Technical Debt: A Boundless Challenge

One important distinction is between unintentional and intentional debt [10]. Unintentional debt occurs due to a lack of attention, e.g., lack of adherence to development standards or unnoticed low quality code that might be written by a novice programmer. Intentional debt is incurred proactively for tactical or strategic reasons such as to meet a delivery deadline. Intentional debt has been further broken down into short-term debt and long-term debt, which represent, respectively, small shortcuts like credit card debt, and strategic actions like a mortgage. Based on this classification, Fowler created a more elaborate categorization composed of two dimensions—deliberate/inadvertent and reckless/prudent [11]. These dimensions give rise to four categories: deliberate reckless debt, deliberate prudent debt, inadvertent reckless debt, and inadvertent prudent debt. This classification is helpful in finding the causes of technical debt, which lead to different identification approaches. For example, to identify reckless and inadvertent debt, especially design debt, source code analysis may be required.

Technical debt can also be classified in terms of the phase in which it occurs in the software lifecycle—design debt, testing debt, defect debt, etc. [12]. Design debt refers to the design that is insufficiently robust in some areas or the pieces of code that need refactoring; testing debt refers to the tests that were planned but not exercised on the source code. This type of classification sheds light on the possible sources and forms of technical debt, each of which may need different measures for identification, and approaches for management. For example, comparison to coding standards may be required to identify and measure design debt, while testing debt measures require information about expected testing adequacy criteria. Other types of debt, based on the lifecycle phase or the entity in which the debt occurs, have been suggested in the literature, e.g., people debt, infrastructure debt, etc. Some types of TD grow organically, without any actions on the part of developers, because software and technology inevitably become out of date with respect to their environment.

Another important dimension along which instances of TD can be categorized is visibility to the customer and end-users. Instances of visible TD include poor usability and low reliability. Instances of invisible TD include violations of architectural rules and missing documentation. Some, but not all, definitions of TD exclude debt that is visible to the end-user.

TD has been recognized to exist in every sector of the software industry, and the research community has been active on this topic. Formal, scholarly investigation of TD is just beginning and is starting to produce usable improvements. The

number of TD research outputs is increasing rapidly. There have been to date three Workshops on Managing Technical Debt, the first one resulting in a joint research agenda [13] and the remaining two co-located with ICSE 2011 and 2012. Two more workshops are planned for 2013. A recent *IEEE Software* special issue contains several articles on the multifaceted concept of TD [5]. Unfortunately, these workshops and published papers have to date failed to produce a universally agreed-upon and used definition of TD. Thus, there are signs that the term TD has been overloaded and is losing its meaning. Every new software engineering technique or empirical investigation has (to a greater or lesser extent) an impact on, or is affected by, TD. For instance, an empirical study comparing the effectiveness of two V&V techniques could support mitigation of TD because future TD decisions about which V&V activity to implement can be based on such an observation. There is a noticeable trend in titling software engineering papers to relate them to TD, even if that relationship is tenuous. However, the lack of a concrete definition of TD makes it difficult to argue against such titling. Such broad labeling of published studies makes aggregation of results hard. Clearly, an ongoing challenge for the TD research community is to find a way to define the term broadly enough to encompass all relevant research, but concretely enough to draw a useful boundary and give guidance to authors.

---

## 4 Technical Debt Brings Empirical Opportunities

The concept of technical debt is one that resonates strongly with the developer community as evidenced by the number of practitioner-authored blog posts, presentations, and webinars conducted on the topic. It has been our experience that practitioners desire research results on this topic more so than on many other subjects of empirical research. What is the reason for this surge of interest? We argue that an important reason is maturity of the empirical research methods applied to TD—many of the most powerful and mature empirical methods have come together in a mutually-supportive way to study TD, to engage real-world problems, and to communicate useful results to practitioners.

### 4.1 Identifying and Predicting TD Costs Is Improved by Empiricism

In general, managing TD consists of estimating, analyzing, and reasoning about: (1) where TD exists in a system so that it can be tagged for eventual removal, (2) the cost of removing TD (i.e., the principal), and (3) the consequences of not removing TD (i.e., the interest). Regarding point (1), there is a large body of software engineering literature [6, 14] related to how to identify TD. Points (2) and (3) require a more careful discussion. In the context of TD, the term “principal” refers to the cost of fixing the technical problem (i.e., removing the debt). For instance, the principal related to the debt affecting a component of a system with high coupling and

cohesion refers to the effort necessary to refactor the component to achieve a lower level of coupling and cohesion (e.g., refactoring the component is estimated to cost \$500). Principal needs to be estimated, and the principal estimate will be more accurate and the resulting decisions more sound with a reliable knowledge base. However, in the absence of historical data, a rough estimate (e.g., high, medium, low) based on expert opinion is more helpful than no estimate at all.

In the context of TD, the term “interest” refers to the cost that will be incurred by not fixing the technical problem (i.e., the consequences of not removing the debt). For example, the interest related to a component of a system with high coupling and cohesion refers to the extra effort that will be necessary to maintain the component in the future. We note that, unlike the principal, the interest is not certain but has an associated probability of occurrence. In other words, you can be sure that refactoring a component will cost you something (i.e., \$500), but you cannot be certain about the consequences of not refactoring it. Therefore, estimating interest means estimating both the amount and its probability of occurrence. These estimates are difficult to make, and in practice a rough estimate of high, medium, and low is the best that can be obtained. Even these rough estimates are more reliable if they are based on historical data.

Managing TD encompasses several estimation activities that are clearly of an empirical nature. In practice, it is difficult to predict anything without a reliable knowledge base.

## 4.2 The Pivotal Role of Context and Qualitative Methods

TD concepts like principal and interest are context-specific. In fact, the same TD in one organization (e.g., a specific level of coupling and cohesion) can have a low or high principal (i.e., can be easy or hard to eliminate) and a low or high debt (i.e., can have a low or high impact in the future) depending on the project or even the subsystem within a project. Thus, there is a need for quantitative methods to elicit meaningful representations of interest and debt in context.

Unfortunately, we do not yet know how to determine when one project is similar enough to another to experience similar results. Even when working in the same exact context, the future will always differ from the past. For example, when working with the same industrial partner, they may experience employee turnover that threatens the applicability of past results. Even if employees do not change, their experience and performance inevitably changes over time. Moreover, almost every software engineering technique is dependent on others. Thus, it is questionable if the assessment of a given technology still holds when the related technologies changed.

In the context of TD, principal and interest clearly vary among environments and there is a need to know one’s customers and collaborate closely with them. Context factors can be elicited in a number of ways. Qualitative methods are needed when it is not clear which factors are relevant, or when there is a desire to discover new unknown context factors. Given the appropriate prompts, developers, managers, and



other stakeholders can all provide important context information through interviews, focus groups, or observation.

There was a time when the software engineering research community debated whether qualitative research methods are appropriate, and proponents had to advocate for more adoption [15, 16]. These qualitative methods are now an integral part of the TD work and one of the reasons why TD tech transfer has been so effective.

When managing or studying TD in particular, two important elements of context are: (1) the software qualities of interest (i.e., what are the most important success criteria?); and (2) the “pain points” (i.e., where has the interest on TD been felt most acutely?). For example, if an organization is most concerned with on-time delivery, then they would be most interested in dealing with TD that causes late-lifecycle surprises, such as inadequate testing. If an organization has a history of damaging cost overruns during maintenance on a large legacy system, then they would be most interested in controlling design debt by refactoring code that is brittle, overly complex, or hard to maintain. Such subtle elements of context are often not documented, and can only be discovered through asking the right questions.

As an example, in [17], practitioners were interviewed and shared a variety of long-term pains resulting from TD. These pains varied from fragile code to poor performance to the added complexity of problems found at the customer site. They also shared varying contexts that influence decisions made about TD, such as the difference between short-lived, non-critical software and software whose longevity is uncertain. The open-endedness of the interviews made it possible to elicit elements of context that had not previously been reported in the literature, and also to gain a richer understanding of previously-known factors.

Sometimes, context information that relates to quality goals and to “pain points” can be found by mining the data archives of a project. In [18], the authors used archival data to conduct an in-depth retrospective study of a particularly high-interest instance of TD. The interest was incurred due to a decision to delay an upgrade in the infrastructure software, then a decision to make a substantial change in the architecture, which then necessitated a greatly increased amount of work later when the upgrade could no longer be delayed. The data analysis revealed the historical sequence of events and decisions that made this TD more expensive in the long run than it initially appeared.

In another study [19], we examined the use of reference architecture across projects in a mid-sized software development company that focuses on database-driven web applications. By collaborating with practitioners, we found that technical debt arises when developers design their own solutions and avoid reuse, and that designing the system to be in compliance with the reference architecture leads to greater understandability and maintainability over time in the future. In this same context, we also observed that code smells and out-of-date documentation can be realistic indicators of technical debt.

Finally, by observing a team developing high-performance code for supercomputers, we noticed that they solve the challenge of optimizing the use of the parallel

processors by strongly separating calls to the parallelization libraries from the code doing scientific simulation, thereby allowing both the computer scientists and the domain experts to focus on what they know best [20]. In this context, the instances where this separation of concerns breaks down should be treated as technical debt—by detecting and fixing where the planned architecture of the system is not followed, we can help the developers create a more maintainable and flexible system.

We note that in all these industry collaborations, discussions with the whole team were essential to validating our concepts of TD in that context. For example, the idea of separation of concerns may not be applicable on a more homogeneous team that does not have to deal with multiple types of specialized complexity, and we do not expect that specific rules for identifying “code smells” would be applicable for every development team.

### 4.3 Tool Support Enables Empiricism

Much of the measurement “infrastructure” that is our legacy from the empirical software engineering research community (e.g., GQM, QIP) was originally defined as a set of methodologies without an explicit tool-supported component. Contemporary empirical studies, in contrast, rely heavily on tool support and automation in order to deal with the size and complexity of today’s software engineering products when collecting, analyzing, and exploring metrics data. Moreover, automated or computer-assisted approaches are necessary to get buy-in from project teams who are used to doing all of their other project activities online. TD is no exception.

To date, several methods and tools for detecting anomalies in source code (automated static analysis, code smell detectors, etc.) have been developed, and these tools show promise for the task of identifying TD [19]. However, these tools have not yet been integrated with developers’ day-to-day work practices and tools and, more importantly, with management’s day-to-day decision-making processes. These shortcomings have led to limited adoption of existing methods to manage TD in industry and a lack of understanding about what can be gained from managing debt.

A long-term vision for tool support for TD decision-making, which will always by necessity include a human component, is a set of integrated tools that continually monitor and diagnose the forms of TD that are accumulating and that threaten the goals of the project, providing a continual stream of actionable information to human decision-makers. Such a toolkit must be integrated so that one single, seamless interaction is available to practitioners for all steps involved in choosing and applying TD identification techniques, aggregating the results, analyzing the choices available for a particular decision, and recording the outcome of the decision itself. The toolkit must also be extensible to incorporate new technical debt identification techniques as well as other decision approaches as they mature. The toolkit must be accompanied by a methodology that describes not only the process of applying the toolkit and choosing among the available options, but also how the



use of the toolkit should fit into software maintenance project management practices already in place.

Such tool support is necessary for technology transfer in this area, not only because industry adoption depends on effective tools, but also because such tools will allow experimentation with our findings in development environments and assist in the collection of necessary data to support further research.

## 4.4 Smarter Dashboards

The idea of dashboards to provide an aggregated view over key metrics is not a new one. The recent trend in this area, however, is the development of a set of principles that exploit the typical data-rich development environment to provide more effective and useful guidance. Across a number of different software domains, some of these common principles include the need for metrics dashboards to be:

- *Built on automated data collection*: Making the collection of measures an extra step for developers who are already overloaded is rarely a recipe for success. Metrics programs with staying power use metrics that are already available for other reasons, e.g., the use of timesheet or time accounting systems to track effort and tasking, or the use of JIRA and other bug tracking tools to measure defect backlog and closure rate. Effective dashboards are those which are built upon integrating data streams from these existing sources, and tying these measures to the questions of interest.
- *Easily Changeable*: In software acquisition environments, data comes from a number of different sources and may change from period to period (within contractually mandated limits). Effective dashboards are those which can easily be adapted to new data schemas as necessary.
- *Trustability*: The data used by the dashboard must be checked to help provide confidence in the results presented. Usually, the quantity of data requires at least some level of automation to verify the quality of the data and the dashboard results.
- *Allowing details-on-demand*: The ability to roll-up low level data, such as data measured at the subsystem level, separate components, or external sources, is necessary if the dashboard is to provide high-level status monitoring. If not done properly, however, roll-up data may camouflage valuable insight. Recent advances in reusable graphical libraries ensure that data exploration can happen quickly and efficiently.

Having a dashboard that pulls together and visualizes TD information would be beneficial during the decision-making process, and hence work has been ongoing in this area [21]. The dashboard should present information that is relevant to both engineers and managers, at varying levels of abstraction. For example, the dashboard could show how many TD instances have been identified, and provide capability for the engineers to drill down to the source of each TD tied to the

development work products, e.g., location of the source code that could benefit in refactoring, problematic requirement statements, etc. Such information enables traceability between each TD instance and the affected documentation, as well as allowing the engineers to understand the extent and context of the TD. Additionally, business-level information, such as comparison between the principal cost and estimated interest of the TD, may be more relevant for the managers. The cost can be rolled up to show the collective impact of TD for the whole system/program. At the same, on demand, managers could drill down and understand how the impact of TD is distributed across the subsystems or program modules or other development work products.

More ambitiously, we envision future TD dashboards that include what-if analysis capability, where managers could play out various scenarios corresponding to the removal of one or more TD instances. For each scenario, the dashboard could visualize how the removal of a source of TD impacts the landscape of the remaining TD (the principal and the interest). Additionally, the dashboard could offer the view of the TD landscape over time—one TD instance's cost may increase at a more significant rate than others as the cost of principal and interest (likelihood and additional effort) may vary over time. All these different perspectives could assist in the prioritizing of TD removal effort.

---

## 5 Conclusion

In this chapter, we have shown how research in one of the fastest-growing areas of empirical study, Technical Debt, relies on a variety of results and methods from past empirical software engineering research. This represents an important development in its own right, since software engineering research has long suffered from an inability to build on previous results. We have also shown how the success of TD research is indebted to recent trends in all of those areas. In this sense, TD makes an interesting case study in how the current level of maturity in empirical studies is paying dividends—it is perhaps only a little hyperbolic to call this a watershed moment for empirical study, where many areas of progress are coming to a head together.

The TD metaphor itself is an important focus of further work since it provides a framework that is both compelling to practitioners and ties together research results on many different topics. This chapter provided our vision of where the research can go: producing an improved and context-specific approach to software measurement that provides tighter feedback loops and more information to developers when they can best use it. Just as importantly, the refinements to empirical research methodologies and principles in the TD work (e.g., accounting for context and business value) will be crucial to other areas of software engineering research by strengthening the ability to influence software development practice.

## References

1. Basili, V.R.: A personal perspective on the evolution of empirical software engineering. In: Münch, J., Schmid, K. (eds.) *Perspectives on the Future of Software Engineering: Essays in Honor of Dieter Rombach*. Springer (2013)
2. Cunningham, W.: The WyCash portfolio management system. *ACM SIGPLAN OOPS Messenger* **4**(2), 29–30 (1993). doi:[10.1145/157710.157715](https://doi.org/10.1145/157710.157715)
3. Lehman, M.M., Belady, L.A.: *Program Evolution - Processes of Software Change*, APIC Studies in Data Processing No. 27, Academic (1985)
4. Parnas, D.L.: Software aging. In: *Proceedings of 16th International Conference on Software Engineering*, pp. 279–287. IEEE Computer Society Press. doi:[10.1109/ICSE.1994.296790](https://doi.org/10.1109/ICSE.1994.296790) (1994)
5. Kruchten, P., Nord, R.L., Ozkaya, I.: Technical debt: from metaphor to theory and practice. *IEEE Softw.* **29**(6), 18–21 (2012). doi:[10.1109/MS.2012.167](https://doi.org/10.1109/MS.2012.167)
6. Sterling, C.: *Managing Software Debt: Building for Inevitable Change*. Agile Software Development Series. Addison-Wesley Professional (2010). ISBN-10: 0321554132
7. Basili, V.R., Selby, R.W.: Comparing the effectiveness of software testing strategies. *IEEE Trans. Softw. Eng.* **SE-13**(12), 1278–1296 (1987). doi:[10.1109/TSE.1987.232881](https://doi.org/10.1109/TSE.1987.232881)
8. Basili, V.R., Green, S., Laitenberger, O., Shull, F., Sørumgård, S., Zelkowitz, M.V.: The empirical investigation of perspective-based reading. Retrieved from <http://dl.acm.org/citation.cfm?id=241252> (1995)
9. Biffl, S., Aurum, A., Boehm, B., Erdogmus, H., Grünbacher, P.: *Value-Based Software Engineering* (Google eBook), p. 388. Springer. Retrieved from <http://books.google.com/books?id=CAIM6nNPcsgC&pgis=1> (2006)
10. McConnell, S.: 10x software development. Retrieved from <http://forums.construx.com/blogs/stevemcc/archive/2007/11/01/technical-debt-2.aspx> (2007)
11. Fowler, M.: Technical debt quadrant. Retrieved from <http://www.martinfowler.com/bliki/TechnicalDebtQuadrant.html> (2009)
12. Rothman, J.: An incremental technique to pay off testing technical debt. Retrieved from <http://www.stickyminds.com/sitewide.asp?Function=edetail&ObjectType=COL&ObjectId=11011&tth=DYN&tt=siteemail&iDyn=2> (2006)
13. Brown, N., Ozkaya, I., Sangwan, R., Seaman, C., Sullivan, K., Zazworka, N., Cai, Y., et al.: Managing technical debt in software-reliant systems. In: *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research – FoSER’10*, p. 47. ACM Press, New York. doi:[10.1145/1882362.1882373](https://doi.org/10.1145/1882362.1882373) (2010)
14. Zazworka, N., Shaw, M. A., Shull, F., Seaman, C.: Investigating the impact of design debt on software quality. In: *Proceeding of the 2nd Working on Managing Technical Debt – MTD’11*, p. 17. ACM Press, New York. doi:[10.1145/1985362.1985366](https://doi.org/10.1145/1985362.1985366) (2011)
15. Dybå, T., Prikładnicki, R., Rönkkö, K., Seaman, C., Sillito, J.: Qualitative research in software engineering. *Empirical Softw. Eng.* **16**(4), 425–429 (2011). doi:[10.1007/s10664-011-9163-y](https://doi.org/10.1007/s10664-011-9163-y)
16. Dittrich, Y., John, M., Singer, J., Tessem, B.: For the special issue on qualitative software engineering research. *Inf. Softw. Technol.* **49**(6), 531–539 (2007). doi:[10.1016/j.infsof.2007.02.009](https://doi.org/10.1016/j.infsof.2007.02.009)
17. Lim, E., Taksande, N., Seaman, C.: A balancing act: what software practitioners have to say about technical debt. *IEEE Softw.* **29**(6), 22–27 (2012). doi:[10.1109/MS.2012.130](https://doi.org/10.1109/MS.2012.130)
18. Guo, Y., Seaman, C., Gomes, R., Cavalcanti, A., Tonin, G., Da Silva, F. Q. B., Santos, A. L. M., et al.: Tracking technical debt – an exploratory case study. In: *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, pp. 528–531. IEEE. doi:[10.1109/ICSM.2011.6080824](https://doi.org/10.1109/ICSM.2011.6080824) (2011)
19. Schumacher, J., Zazworka, N., Shull, F., Seaman, C., Shaw, M.: Building empirical support for automated code smell detection. In: *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement – ESEM ’10*, p. 1. ACM Press, New York. doi:[10.1145/1852786.1852797](https://doi.org/10.1145/1852786.1852797) (2010)

- 
20. Hochstein, L., Shull, F., Reid, L.B.: The role of MPI in development time: a case study. In: 2008 SC – International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–10. IEEE. doi:[10.1109/SC.2008.5213771](https://doi.org/10.1109/SC.2008.5213771) (2008)
  21. Zazworka, N., Basili, V.R., Shull, F.: Tool supported detection and judgment of nonconformance in process execution. In: 2009 3rd International Symposium on Empirical Software Engineering and Measurement, pp. 312–323. IEEE. doi:[10.1109/ESEM.2009.5315983](https://doi.org/10.1109/ESEM.2009.5315983) (2009)