

Applying DevOps Practices to Course Material

Recommendations from Software Development to Develop and Manage Courses

Joel Coffman*

United States Air Force Academy
Department of Computer and Cyber Sciences
United States of America
joel.coffman@afacademy.af.edu

ABSTRACT

DevOps, a portmanteau of development and operations, is a proven way to develop and manage IT systems. Key benefits compared to traditional software processes include faster time to market, which enables businesses to adapt to changing market conditions more rapidly than their peers, while providing world-class dependability and security. Many DevOps principles and practices also apply to academia even though curricula may not evolve as quickly as modern IT services. This paper encourages the adoption of specific capabilities that drive these performance improvements, including nascent examples of their use to manage course material, to improve the efficiency and effectiveness of faculty members' daily work.

CCS CONCEPTS

• **Social and professional topics** → *Computing education*; Project management techniques; • **Software and its engineering** → *Software development methods*.

KEYWORDS

DevOps, course material, course development

ACM Reference Format:

Joel Coffman. 2025. Applying DevOps Practices to Course Material: Recommendations from Software Development to Develop and Manage Courses. In *The 27th Western Canadian Conference on Computing Education (WCCCE '25)*, April 28–29, 2025, Calgary, AB, Canada. 7 pages. <https://doi.org/10.60770/027f-f062>

1 INTRODUCTION

Faculty members increasingly confront the expectation to do more with less. From shrinking budgets to ballooning enrollments, faculty find themselves saddled by additional responsibilities that stretch their ability to satisfy the expectations of the traditional academic pillars of teaching, research, and service: “it is no longer a (thinly veiled) secret that in contemporary universities many scholars, both junior and senior, are *struggling* – struggling to manage their workloads; struggling to keep up with insistent institutional demands to produce more, better and faster; struggling to reconcile

professional demands with family responsibilities and personal interests; and struggling to maintain their physical and psychological health and emotional wellbeing” [45]. The experiences reported by many faculty [24] parallel those encountered in IT organizations where firefighting and heroic efforts are too often the norm. More generally, friction not only bedevils developers [18] but also faculty [40] and imposes significant costs, not only to an organization as a whole (e.g., opportunity cost) but also to individuals (e.g., burnout [38]). DevOps, a set of cultural norms and technical practices, offers a better way of working, improving a host of outcomes including software delivery performance and job satisfaction [16]. These concepts also apply to academia and promise similar benefits.

Despite its adoption by industry, DevOps lacks substantial traction in academia, including computing pedagogy. Lack of experience and perceived overhead deter the adoption of DevOps practices in academia [44] despite their dramatic and well-established benefits: top performers in industry are not only ahead of [3, 15, 17] but also pulling away from their peers [16]. The need to treat course material similar to other software projects—i.e., applying software development practices to the design, implementation, and maintenance of course material—has previously been recognized [27, 32, 33, 36]. As faculty wrestle with increased demands on their time, DevOps replaces the need for Herculean efforts with a proven path to improve outcomes.

DevOps complements evidence-based approaches to course design (e.g., backward design [14, 59]) and offers specific technical practices that improve the efficiency of developing and managing courses. Though the artifacts differ from those produced by IT organizations, there are parallels: courses evolve (e.g., [21, 33]) as faculty adjust policies, update topics, and revise assignments to improve students' mastery of the material. Moreover, faculty increasingly collaborate on everything from individual assignments to entire courses. The major contributions of this work are as follows:

- It introduces DevOps, a set of cultural norms and technical practices that may not be familiar to faculty, particularly those without recent experience in the software industry.
- It recommends the adoption of specific capabilities that drive performance improvements in IT organizations along with examples of their existing—albeit often limited—use to develop and to manage course material.
- It discusses the benefits and challenges of DevOps in an educational setting.

The goal is to encourage computing faculty to embrace an established industry trend, which offers a number of benefits for faculty members' day-to-day work and is particularly, though not exclusively, suited to managing courses at scale.

*Also with Johns Hopkins University, Engineering for Professionals.

This work is completed in its entirety by the author. All rights reserved. You may share digital or hard copies of this work for non-commercial purposes, provided that the work is not altered in any way, and that each copy includes proper attribution, the full citation, and this license notice. For all other uses, contact the author.
WCCCE 2025, April 28–29, 2025, Calgary, Alberta, Canada
© 2025 Copyright held by the author.
<https://doi.org/10.60770/027f-f062>

The remainder of this paper is organized as follows. Section 2 summarizes related work and describes DevOps. Section 3 provides recommendations for adopting DevOps capabilities to manage course material. Section 4 reflects on benefits and challenges. Finally, Section 5 concludes.

2 BACKGROUND

This section highlights related work and provides an overview of DevOps. This material addresses the need “to improve DevOps recognition in academia” [44], specifically among teaching faculty.

2.1 Related Work

“Time is an issue for all faculty” [35]. Exhaustion, stress, overload, and anxiety have become commonplace [22], which contributes to rising levels of burnout [20, 51]. DevOps can reduce and even to prevent burnout [16], and this work is the first to specifically propose the adoption of DevOps by faculty in the context of teaching.

Throughout the past decade, there have been repeated calls to manage course material similarly to software projects, yet adoption has been limited largely to the use of version control systems or platforms. Mandala and Gary [36] state, “we have to start thinking of courseware development as an engineering process, much like software development.” Similarly, Kloos et al. [32] call for “courseware engineering” that emphasizes the effectiveness, maintainability, and reusability of educational material. Haaranen et al. [27] chide themselves and other computing educators, “Even though we know about best practices in software development, they seem to be often forgotten when it comes to our own course development.” They suggest that limited time, an emphasis on developing new material quickly, and changing course staff (including faculty and TAs) contribute to the failure to follow established software development practices. DevOps directly improves the first two issues and indirectly addresses turnover by creating a safe system of work to recognize and resolve mistakes expediently.

“Courseware as Code” [49] builds on the foundation of DevOps for course management and improvement, including distributed version control (e.g., Git) and automated workflows to increase transparency and consistency of contributions. Edmison et al. [12] propose a workflow for assignment management that leverages version control and automation. Aquinas [46] applies the “everything-as-code” approach to the development and delivery of hands-on exercises. ClassOps [39] aims to streamline course delivery. Deshpande et al. [11] use DevOps practices for assignment validation and delivery. In comparison, this paper provides a systematic framework for the adoption of DevOps for course material.

2.2 DevOps

DevOps comprises cultural norms and technology practices that enable the fast flow of work from development into operation while providing world class reliability and security [25]. Technologies and tools vary across organizations and even across teams within an organization; DevOps focuses on key outcomes and specific capabilities to improve those outcomes, building on preexisting philosophies such as agile software development and lean.

Figure 1 illustrates the DevOps process. The left side is similar to agile frameworks with an emphasis on incremental and iterative

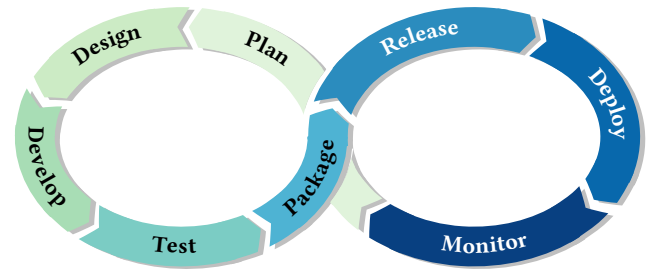


Figure 1: The DevOps process

planning. Although design, development, and testing are depicted sequentially, they may be interleaved. The right side focuses on operational aspects. Explicit packaging and release processes ensure that software runs correctly when deployed to production, and monitoring detects issues proactively rather than reactively.

High-performing organizations that embrace DevOps significantly outperform their peers across a variety of metrics [16]:

- Their *lead time*, the time spent between requesting work to completing it, is hundreds of times faster.
- Their *deployment frequency* is more than forty times faster.
- Their *change failure rate*, the percentage of changes that cause service failures or require rework, is 80% lower.
- Their *mean time to recovery (MTTR)* is more than one hundred times lower.

Although these metrics are critical to IT systems, the ideas also apply in academic settings. For example, what faculty member does not want to create new course material quickly while ensuring that the material integrates correctly with the rest of the course? How many faculty have released an assignment only to have a student find a mistake that requires immediate resolution?

The Three Ways. The outcomes that define DevOps stem from three core principles (see Figure 2). A critical concept is the *value stream*, “the sequence of activities an organization undertakes to deliver upon a customer request” [37] or whatever process delivers value [31]. Pragmatically, faculty should strive to create and to deliver learning activities as efficiently as possible.¹

Flow. These principles focus on the fast flow of work from development to operations. The goal is to decrease *lead time* while simultaneously increasing dependability. Knowledge work (e.g., a faculty member’s responsibilities) lack a physical representation; consequently, work should be visualized using a technique like kanban to see where work is flowing well and where it is stalled [10]. Strictly controlling the amount of work in progress is essential, for tasks that remain “in progress” not only have yet to deliver value but also introduce overhead due to context switching. Automation is a particularly powerful tool (consider, for example, automated assessment in computing education [43] and, more generally, faculty members’ desire to reduce the time spent on repetitive tasks [40]).

¹Instruction-centered practices, such as lecturing, disseminate information efficiently [23] but are relatively ineffective teaching methods [19]. More effective strategies, such as project-based learning [1], often require greater time commitment (i.e., investment by both faculty and students). Believing that instruction is either efficient or effective is a false dilemma; faculty strive to (and can) improve both.

- Flow** The fast flow of work from development to production
- Make work visible
 - Limit work in progress
 - Reduce batch sizes
 - Reduce the number of hand-offs
 - Identify and elevate constraints
 - Eliminate hardships in daily work
- Feedback** Rapid feedback at every stage of the process
- See problems as they occur
 - Swarm and solve problems to build new knowledge
 - Push quality closer to the source
 - Optimize for downstream work
- Learning** Continual learning through experimentation to improve flow and to provide feedback more quickly
- Institutionalize the improvement of daily work
 - Transform local discoveries into global improvements
 - Inject resiliency into daily work

Figure 2: DevOps principles [31]

In course design, flow is realized by the rapid delivery (i.e., deployment) of high quality course material. For example, a new assignment should integrate seamlessly into an existing course while increasing students’ mastery of concepts compared to prior learning activities. Major impediments to flow include multitasking; unnecessary hand-offs, such as multiple rounds of reviews by TAs [11]; and the need for heroics to achieve goals because individuals cannot sustain such efforts indefinitely.

Feedback. These principles emphasize the dependability of not only software but also the processes that manage development and operations. Feedback revolves around safe systems of work: individuals and teams need processes in place to detect and to recover from mistakes. Not surprisingly, timely feedback increases the likelihood of catching mistakes [6]. A simple example of such a mistake in an academic context is distributing a programming assignment with automated tests that do not work correctly [11]. An effective feedback loop requires identifying problems, solving them, and creating automated processes to prevent those problems.

In an academic setting, departments should have a curricular review mechanism to identify changes to course outcomes that would adversely affect students’ preparation for subsequent courses. This review might initially be a manual process, but ideally would be automated [33] so that dependencies between courses (e.g., recommending an algorithm on the basis of its big O analysis) are flagged when changes are proposed. Automating the dependency analysis saves valuable time compared to a generic review, reduces the likelihood of overlooking a dependency, and enables the affected faculty to collaborate on the solution to conflicts.

Learning. The principles of continual learning and experimentation shorten the virtuous cycle created by the principles of flow and feedback. Innovation requires a disciplined and scientific approach to taking risks. Not every change will work as expected in practice, which represents new knowledge to be shared with others.

Faculty implicitly embrace this principle in two forms. One is improving the efficiency of instruction. For example, a faculty member may record a demonstration of using a particular tool; the

Continuous delivery (CD)

- Use version control for all artifacts
- Automate the deployment process
- Implement continuous integration (CI)
- Use trunk-based development methods
- Implement test automation
- Support test data management
- Shift left on security
- Implement continuous delivery (CD)

Architecture

- Use a loosely coupled architecture
- Architect for empowered teams

Product and Process

- Gather and implement feedback
- Make the flow of work visible throughout the *value stream*
- Work in small batches
- Foster and enable experimentation

Lean Management and Monitoring

- Have lightweight change approval processes
- Monitor to inform decisions
- Check system health proactively
- Improve processes and limit work-in-progress
- Visualize work to monitor quality

Cultural

- Support a generative culture
- Encourage and support learning
- Support and facilitate collaboration among teams
- Provide resources and tools that make work meaningful
- Support or embody transformational leadership

Figure 3: Capabilities that drive improvement [16]

initial time investment is recouped as students ask fewer questions in office hours about how to use the tool. Another is improving the effectiveness of instruction. For example, a faculty member may replace existing course materials (e.g., lectures) with evidence-based teaching strategies (e.g., active learning). Such changes, and the assessment of them, form the basis of experience reports and computing education research.

3 RECOMMENDATIONS

Forsgren et al. [16] enumerate 24 capabilities that drive improvement in software delivery performance (see Figure 3). This section describes the capabilities that are most applicable in academia² in the form of specific recommendations and examples that faculty can adopt, incrementally if desired, in the courses that they teach. References to prior work illustrate nascent uses or existing use cases, and examples propose additional ways to embrace these concepts. The recommendations themselves are intentionally general because faculty should have flexibility, not dictates regarding the use of specific technologies, to manage their courses [33].

3.1 Continuous delivery (CD)

Continuous delivery (CD) is a software development practice in which software can be deployed at any time. CD reduces *lead time* while improving quality [29].

²Architectural and cultural capabilities, such as the need to support or embody transformational leadership, are often outside the control of individual faculty members.

3.1.1 Use version control for all artifacts. Comprehensive version control is essential, obviously for software but more broadly for any project that involves collaboration and recovering from mistakes, including the development and management of course material. Used correctly, version control provides a detailed history of *who* made a change, *what* changed, *when* it changed, and *why* it changed.

In comparison to an archive of course material, a version control system captures metadata about changes and simplifies comparisons (e.g., which learning objectives were modified). Cloud-based office productivity suites (e.g., Google Docs Editors and Microsoft 365) automatically version files and offer mechanisms for “suggesting” changes, but these tools do not offer the same support for merging changes when documents evolve independently and recording metadata, particularly the rationale, for changes. Similarly, learning management systems (LMSs) facilitate copying course content, but not a way to visualize and share specific changes as courses evolve.

Seminal work [7] describes the benefits of using version control “to reduce administrative demands and to support creative collaboration” among faculty; these benefits are echoed by faculty elsewhere [12, 26, 28, 34, 49]. Zagalsky et al. [61] summarize the motivations and benefits of using GitHub as a collaborative educational platform, including the visualization of changes as content is updated. Hosting course material, including lecture slides, code samples, and assignments, in GitHub or GitLab is one way to make it available to students [8, 26, 28, 30, 49, 50, 54, 61].

3.1.2 Automate the deployment process. Automating deployment ensures that updates to course materials are published as quickly as possible without requiring any steps to be performed manually.

The need to automate repetitive and tedious tasks, such as publishing assignments to an LMS and provisioning autograders [12], supporting large class sizes [33, 40], and preparing accreditation reports [13], has previously been recognized. A major benefit of deployment automation is that students always have the latest version of course material [8, 26] and dependencies (e.g., source code snippets embedded in a lecture) are up-to-date [33].

3.1.3 Implement continuous integration (CI). Continuous integration (CI) ensures that each change is tested—and any issues fixed—immediately, which epitomizes the DevOps principle of feedback. Practically speaking, CI gains prominence when multiple faculty members collaborate on course material [7, 26].

Lau et al. [33] provide a case study of the difficulty of supporting multiple versions of Python. CI shines in this context. As an example, starter code for a programming assignment and the solution may be tested against currently supported releases (see Figure 4). If, for example, the latest release reports new warnings or errors, those issues will be flagged automatically when the workflow runs, drawing attention to problems while saving valuable faculty time compared to manual testing against even one version of Python.

3.1.4 Use trunk-based development. Changes should be developed from the mainline and quickly merged back into the mainline. Short-lived branches that persist for less than a day are commonly used to implement this practice [16].

In an academic context, the antithesis of trunk-based development is copying content from an existing course, making substantial changes, and subsequently trying to incorporate updates from

```
jobs:
  test:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: ['3.11', '3.12', '3.13']
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v4
        with:
          python-version: ${ matrix.python-version }
      - name: Install dependencies
        run: pip install -r dependencies.txt
      - name: pytest
        run: py.test tests/
```

Figure 4: Excerpt of a GitHub Actions workflow. The test job executes the three Python versions independently.

or sharing changes with the original course. In general, a branch become more difficult to merge over time as both the trunk and branch evolve; this effort ultimately represents waste, either due to the difficulty of integrating changes or due to the absence of improvements (e.g., clarifications to an assignment’s instructions).

3.1.5 Implement test automation. Test automation alleviates the burden of manual testing and increases confidence that changes do not introduce problems. Tests must be *reliable*, meaning that a failure indicates a real issue and passing indicates that the change is safe to release, and should be run on every proposed change.

There are many examples of how to leverage test automation. Spell checking can be performed automatically by a pre-commit hook. A course website can be checked for accessibility features and broken links [8]. Code included in lectures can be compiled using a build tool (e.g., make) to avoid the embarrassment of omitting a semicolon [26]. Code blocks in tutorials can be tested to ensure they are not missing instructions [41]. Although each of the aforementioned tasks can be performed manually, doing so is error-prone, as it is easy to overlook something. Extending this idea, automated grading provides “significant savings of scarce teaching resources” while also improving student performance [60].

3.2 Product and Process

While agile is widely recognized, its principles are often ignored [2].

3.2.1 Gather and implement feedback. Feedback to faculty members may come in many forms: implicitly through students’ performance on assessments and explicitly in the form of course evaluations, classroom observations [42], and pilot offerings of new courses with subsequent refinements [58]. Feedback, regardless of its source, ideally drives improvements to learning activities by identifying what is and is not working in a course. Changes in response to feedback should be annotated in the version control history, providing a detailed record of why the change was made. This same concept applies to computing education research, as faculty test and evaluate the effectiveness of various teaching techniques.

Visibility into students’ struggles and problem-solving processes help faculty correct misconceptions and resolve sources of frustration [40]. Prior work [26, 28, 61] emphasizes student participation, either correcting mistakes or updating content. A decreasing number of issues when there are significant changes to course material indicates a reduced *change failure rate*.

3.2.2 Make the flow of work visible throughout the value stream. Faculty often labor in relative obscurity with respect to teaching activities, emerging at the end with little visibility into how they arrived at that point [40, 56], and it is likely that the effort invested by an individual faculty member is underestimated [53], which makes it difficult to recognize behind-the-scenes logistical work [33].

Techniques like kanban should be used to track all aspects of course design, from the adoption of a single “nifty assignment” to the development of an entire course. A visual workflow clearly identifies obstacles (e.g., administrative delays in approving curricular changes) and allows faculty to make informed decisions based on what is feasible for them to accomplish in the available time.

3.2.3 Work in small batches. A small batch size reduces work in progress, which predicts lead time, and lead time is correlated with quality, customer satisfaction, and employee happiness [31].

One application of this concept is making small incremental changes to a course [21] rather than combining those changes: larger changes delay the detection of mistakes (e.g., discrepancies between stated learning objectives and questions in a test bank) and increases the amount of rework when mistakes are found. Of course, every course eventually requires a major redesign, but even in this instance, the concept of the *minimum viable product* [48] is valuable, as it facilitates early feedback (e.g., from other faculty members) on the initial concept rather than deferring it.

3.2.4 Foster and enable experimentation. When a particular teaching strategy isn’t working for a group of students, faculty should be free to try out new ideas based on established pedagogical principles to improve students’ learning experience. In practice, this idea dictates flexibility to design and adapt courses while holding faculty responsible for the course’s established learning outcomes.



3.3 Lean Management and Monitoring

Lean creates value by improving efficiency and reducing waste.

3.3.1 Have lightweight change approval processes. Peer review coupled with CD (§3.1) improves software delivery performance. In teaching, faculty—particularly new instructors—may solicit feedback on changes from other faculty members. In comparison, requiring explicit approval from an external body (similar to a change advisory board in IT) may impose delay without improving quality.

Pull (or merge) requests are an excellent way to provide lightweight peer reviews. They offer an asynchronous review mechanism for faculty to weigh in on curricular issues where consensus is required, such as a proposed change to a course description. Such review does not detract from faculty’s academic freedom: public discourse increases transparency and provides a record of discussion and decisions that are likely to be forgotten as years pass. Fast feedback allows faculty to identify and to correct issues immediately rather than waiting until they become more difficult to address.

3.3.2 Monitor to inform decisions. Rather straightforward, take action when something is not working as expected. For example, frequent low-stakes assessments enable faculty to review what students are and are not learning to ensure competence with foundational skills prior to moving on to more advanced concepts.

Monitoring should be proactive, designed to identify issues before they become significant problems. For example, potential copyright violations for third-party images from Wikimedia Commons can be flagged when the image is removed from the media repository. Documentation rot and inaccessible resources, which contribute to non-executable tutorials [41], should be identified and corrected before students encounter them. Repository badges (e.g.,   [57] provide an unobtrusive way to alert faculty to software dependency problems prior to the start of a semester.

3.3.3 Improve processes and limit work-in-progress. Limiting work-in-progress keeps individuals and teams from becoming overburdened, which increases lead time, and exposes obstacles to the fast flow of work. Controlling work in progress ultimately improves quality, student satisfaction, and faculty members’ well-being.

Griffith and Altinay [24] describe a framework to evaluate faculty workloads, particularly teaching, because “many faculty members find themselves carrying higher workloads than their peers, feeling overburdened by increasing or changing institutional expectations, and failing to reach career goals.” Limiting work-in-progress by itself is insufficient: visualizing work and an effective feedback loop are critical to identify and eliminate obstacles [16].

4 DISCUSSION

DevOps is especially suited for large courses with hundreds of students and multiple faculty members (or TAs). Nevertheless, it should not be ignored by individual faculty members, including those without TA support. The DevOps principles related to flow have been shown to improve productivity and reduce burnout in IT organizations [16]; achieving a “flow state” and reducing cognitive load have also been shown to improve developer, team, and organizational outcomes [18]. Academia is no exception to the need to manage work effectively. Furthermore, institutional efforts, such as disability services and accreditation, implicitly involve all faculty members, even adjuncts teaching a single course. Technical practices like CD decrease the amount of unplanned work and rework, which leads to a nearly 30% increase in time for new work [16].

Ease of use and perceived usefulness, which is strongly correlated with estimated time saving [52], are key to adopting new technologies [5, 9]; barriers include lack of reliability [5]; structural constraints, such as lack of adequate support [4]; and, especially for faculty, time [44, 47, 55]. Industry-standard tools (e.g., Git) and platforms are widely used and reliable, and some platforms offer special support for academic faculty. With respect to time, adopting DevOps practices is not considerably different than adopting learner-centered design: both require up-front investment, and both have demonstrated benefits in the long run. Although the specter of a steep learning curve is daunting, incrementally adopting DevOps capabilities is a viable and worthwhile path forward.

Interviews with faculty indicate specific barriers to the use of third-party services, such as GitHub, including institutional policies and legal restrictions [61]. These issues are more critical in the

context of student work and “fair use” of copyrighted material in a classroom setting. Students’ contributions to course material are generally limited, such as correcting mistakes [26, 28]; even if public, statutes like the Family Educational Rights and Privacy Act (FERPA) in the United States restrict the release of “education records” by an institution, which does not apply to voluntary contributions by students [62]. Publishing copyrighted material can be avoided by making a repository private or linking to material hosted by an institutionally-supported platform (e.g., Dropbox, Google Drive, or Microsoft OneDrive) with appropriate authentication. These concerns can also be mitigated by using platforms managed institutionally with single sign-on (SSO) albeit with an increased administrative overhead compared to cloud-based services [34].

Faculty and non-academic staff, such as instructional designers, may be using DevOps practices without publishing about their experiences or using different terminology. Some of DevOps’s essential capabilities are being adopted piecemeal, such as the use of version control, as described previously. In contrast, Pang et al. [44] emphasize that faculty are “not motivated to learn or adopt DevOps” due to its perceived cost. While pressure—particularly demands on faculty time—may eventually force the adoption of new ways of working, the status quo may be “good enough” for years to come.

Finally, will DevOps, which has proven invaluable in the IT industry, realize similar benefits in academia? Anecdotal evidence, most frequently in the form of “experience reports” that address the aforementioned capabilities, indicates an affirmative response to this question (e.g., [11, 26, 28, 49]). For example, Hofstätter et al. [28] describe their experience as “truly time-saving and sustainable throughout a busy semester” with overwhelming positive feedback from students. This paper provides a systematic framework to place these ideas in context and argue for their adoption more widely.

5 CONCLUSION

Paraphrasing Heraclitus, “Change is the only constant.” Faculty in computing—more so than most disciplines—must adapt to the rapid obsolescence of technologies and evolution of course content [55]. DevOps has proved to be an effective approach to develop and manage IT systems; many of these capabilities also apply to the development and management of course material.

Two avenues for future work bear particular mention. First, widespread adoption of DevOps practices in the context of managing course material is likely predicated on the creation and sharing of effective DevOps workflows. Improving flow can be as simple as automating processes; existing CI services offer a way to share these reusable workflows, lessening their learning curve for faculty. Second, prior work in the IT industry (e.g., the “State of DevOps” reports³) provides a guide for rigorously measuring the benefits of DevOps in an academic setting, both for managing courses with a significant staff and courses taught by an individual faculty member.

ACKNOWLEDGMENTS

The views expressed herein are those of the author and do not necessarily reflect the official policy or position of Johns Hopkins University or the United States Air Force Academy, the Air Force, the Department of Defense, or the U.S. Government.

³DORA publications: <https://dora.dev/publications/>

REFERENCES

- [1] Phyllis C. Blumenfeld, Elliot Soloway, Ronald W. Marx, Joseph S. Krajcik, Mark Guzdial, and Annemarie Palincsar. 1991. Motivating Project-Based Learning: Sustaining the Doing, Supporting the Learning. *Educational Psychologist* 26, 3-4 (1991), 369–398. <https://doi.org/10.1080/00461520.1991.9653139>
- [2] Defense Innovation Board. 2018. *Detecting Agile BS*. Technical Report. Department of Defense.
- [3] Alanna Brown, Nicole Forsgren, Jez Humble, Nigel Kersten, and Gene Kim. 2016. 2016 State of DevOps Report. <https://dora.dev/research/2016/2016-state-of-devops-report.pdf>
- [4] Tom Buchanan, Phillip Sainter, and Gunter Saunders. 2013. Factors affecting faculty use of learning technologies: implications for models of technology adoption. *Journal of Computing in Higher Education* 25, 1 (April 2013), 1–11. <https://doi.org/10.1007/s12528-013-9066-6>
- [5] Darrell L. Butler and Martin Sellbom. 2002. Barriers to Adopting Technology for Teaching and Learning. *Educause Quarterly* 25, 2 (2002), 22–28.
- [6] Michael A. Campion and Carol L. McClelland. 1991. Interdisciplinary examination of the costs and benefits of enlarged jobs: A job design quasi-experiment. *Journal of Applied Psychology* 76, 2 (1991), 186–198.
- [7] Curtis Clifton, Lisa C. Kaczmarczyk, and Michael Mrozek. 2007. Subverting the Fundamentals Sequence: Using Version Control to Enhance Course Management. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '07)*. Association for Computing Machinery, New York, NY, 86–90. <https://doi.org/10.1145/1227310.1227344>
- [8] Joel Coffman. 2024. GitHub Pages as an LMS Alternative. In *Proceedings of the 26th Western Canadian Conference on Computing Education (WCCCE '24)*. Association for Computing Machinery, New York, NY, Article 18, 2 pages. <https://doi.org/10.1145/3660650.3660667>
- [9] Fred D. Davis. 1989. Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly* 13, 3 (Sept. 1989), 319–340. <https://doi.org/10.2307/249008>
- [10] Dominica DeGrandis. 2017. *Making Work Visible: Exposing Time Theft to Optimize Work & Flow* (first ed.). IT Revolution Press, LLC, Portland, OR.
- [11] Gururaj Deshpande, Shravan Cheekati, Shail Patel, Pranav Raj, Madhuri Singh, Mark Pindur, Nouf Al Soghyar, Bryan Zhao, Parisa Babolhavaei, Mohammad Taher, Krish Nathan, Will Spaeth, and Max Mahdi Roozbahani. 2024. Transforming CS Education with DevOps: Streamlined Assignment Validation and Delivery @ Scale. In *Proceedings of the Eleventh ACM Conference on Learning @ Scale* (Atlanta, GA, USA) (L@S '24). Association for Computing Machinery, New York, NY, USA, 259–264. <https://doi.org/10.1145/3657604.3664676>
- [12] Bob Edmison, Austin Cory Bart, and Stephen H. Edwards. 2021. A Proposed Workflow For Version-Controlled Assignment Management. In *Proceedings of SPLICE 2021 workshop CS Education Infrastructure for All III: From Ideas to Practice* (Virtual Event) (SPLICE '21). 3 pages.
- [13] Eugene Essa, Andrew Dittrich, and Sergiu Dascalu. 2010. ACAT: A Web-Based Software Tool to Facilitate Course Assessment for ABET Accreditation. In *2010 Seventh International Conference on Information Technology: New Generations*. IEEE, 88–93. <https://doi.org/10.1109/ITNG.2010.224>
- [14] L. Dee Fink. 2003. *Creating Significant Learning Experiences: An Integrated Approach to Designing College Courses*. Jossey-Bass, San Francisco, CA.
- [15] Nicole Forsgren, Jez Humble, and Gene Kim. 2018. 2018 State of DevOps Report. <https://dora.dev/research/2018/dora-report/2018-dora-accelerate-state-of-devops-report.pdf>
- [16] Nichole Forsgren, Jez Humble, and Gene Kim. 2018. *Accelerate: The Science Behind DevOps: Building and Scaling High Performing Technology Organizations*. IT Revolution Press, LLC, Portland, OR.
- [17] Nicole Forsgren, Jez Humble, Gene Kim, Alanna Brown, and Nigel Kersten. 2017. 2017 State of DevOps Report. <https://dora.dev/research/2017/2017-state-of-devops-report.pdf>
- [18] Nicole Forsgren, Eirini Kalliamvakou, Abi Noda, Michaela Greiler, Brian Houck, and Margaret-Anne Storey. 2024. DevEx in Action. *Commun. ACM* 67, 6 (May 2024), 42–51. <https://doi.org/10.1145/3643140>
- [19] Scott Freeman, Sarah L. Eddy, Miles McDonough, Michelle K. Smith, Nnadozie Okoroafor, Hannah Jordt, and Mary Pat Wenderoth. 2014. Active learning increases student performance in science, engineering, and mathematics. *Proceedings of the National Academy of Sciences* 111, 23 (2014), 8410–8415. <https://doi.org/10.1073/pnas.1319030111>
- [20] Virginia Gewin. 2021. Pandemic burnout is rampant in academia. *Nature* 591, 7850 (2021), 489–492. <https://doi.org/10.1038/d41586-021-00663-2>
- [21] Nasser Giacaman, Partha Roop, and Valerio Terragni. 2023. Evolving a Programming CS2 Course: A Decade-Long Experience Report. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) (SIGCSE 2023). Association for Computing Machinery, New York, NY, USA, 507–513. <https://doi.org/10.1145/3545945.3569831>
- [22] Rosalind Gill. 2009. Breaking the silence: The hidden injuries of neo-liberal academia. In *Secrecy and Silence in the Research Process*, Roisin Ryan-Flood and Rosalind Gill (Eds.). Routledge, London, UK, 228–244.

- [23] David Gooblar. 2019. *The Missing Course: Everything They Never Taught You about College Teaching*. Harvard University Press, Cambridge, MA.
- [24] Andrew S. Griffith and Zeynep Altınay. 2020. A framework to assess higher education faculty workload in U.S. universities. *Innovations in Education and Teaching International* 57, 6 (2020), 691–700. <https://doi.org/10.1080/14703297.2020.1786432>
- [25] Gary Gruver. 2016. *Starting and Scaling DevOps in the Enterprise*. BookBaby, Pennsauken, NJ.
- [26] Lassi Haaranen and Teemu Lehtinen. 2015. Teaching Git on the Side: Version Control System as a Course Platform. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education (ITICSE '15)*. Association for Computing Machinery, New York, NY, 87–92. <https://doi.org/10.1145/2729094.2742608>
- [27] Lassi Haaranen, Giacomo Mariani, Peter Sormunen, and Teemu Lehtinen. 2020. Complex Online Material Development in CS Courses. In *Proceedings of the 20th Koli Calling International Conference on Computing Education Research (Koli Calling '20)*. Association for Computing Machinery, New York, NY, Article 26, 5 pages. <https://doi.org/10.1145/3428029.3428053>
- [28] Sebastian Hofstätter, Sophia Althammer, Mete Sertkan, and Allan Hanbury. 2022. A Time-Optimized Content Creation Workflow for Remote Teaching. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education - Volume 1 (SIGCSE 2022)*. Association for Computing Machinery, New York, NY, 731–737. <https://doi.org/10.1145/3478431.3499421>
- [29] Jez Humble. 2018. Continuous Delivery Sounds Great, but Will It Work Here? *Commun. ACM* 61, 4 (March 2018), 34–39. <https://doi.org/10.1145/3173553>
- [30] John Kelleher. 2014. Employing Git in the Classroom. In *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*. IEEE, 1–4. <https://doi.org/10.1109/WCCAIS.2014.6916568>
- [31] Gene Kim, Jez Humble, Patrick Debois, and John Willis. 2016. *The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations* (first ed.). IT Revolution Press, LLC, Portland, OR.
- [32] Carlos Delgado Kloos, Ma Blanca Ibáñez, Carlos Alario-Hoyos, Pedro J. Muñoz-Merino, Iria Estévez Ayres, Carmen Fernández Panadero, and Julio Villena. 2016. From Software Engineering to Courseware Engineering. In *2016 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 1122–1128. <https://doi.org/10.1109/EDUCON.2016.7474695>
- [33] Sam Lau, Justin Eldridge, Shannon Ellis, Aaron Fraenkel, Marina Langlois, Suraj Rampure, Janine Tiefenbruck, and Philip J. Guo. 2022. The Challenges of Evolving Technical Courses at Scale: Four Case Studies of Updating Large Data Science Courses. In *Proceedings of the Ninth ACM Conference on Learning @ Scale (L@S '22)*. Association for Computing Machinery, New York, NY, 201–211. <https://doi.org/10.1145/3491140.3528278>
- [34] Joseph Lawrance, Seikyung Jung, and Charles Wiseman. 2013. Git on the Cloud in the Classroom. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education (Denver, Colorado) (SIGCSE '13)*. Association for Computing Machinery, New York, NY, 639–644. <https://doi.org/10.1145/2445196.2445386>
- [35] Stephanie Lunn, Maira Marques Samary, Susanne Hambrusch, and Aman Yadav. 2022. Forging a Path: Faculty Interviews on the Present and Future of Computer Science Education in the United States. *ACM Transactions on Computing Education* 22, 4, Article 51 (Sept. 2022), 24 pages. <https://doi.org/10.1145/3546581>
- [36] Srikesh Mandala and Kevin A. Gary. 2013. Distributed Version Control for Curricular Content Management. In *2013 IEEE Frontiers in Education Conference (FIE)*. IEEE, 802–804. <https://doi.org/10.1109/FIE.2013.6684936>
- [37] Karen Martin and Mike Osterling. 2014. *Value Stream Mapping: How to Visualize Work and Align Leadership for Organizational Transformation* (1st ed.). McGraw-Hill Education, New York, NY.
- [38] Christina Maslach, Wilmar B. Schaufeli, and Michael P. Leiter. 2001. Job Burnout. *Annual Review of Psychology* 52, 1 (2001), 397–422. <https://doi.org/10.1146/annurev.psych.52.1.397>
- [39] Samim Mirhosseini. 2023. *Addressing CS-Ed Course Material Preparation and Delivery Frictions Through ClassOps*. Ph. D. Dissertation. North Carolina State University, Raleigh, North Carolina.
- [40] Samim Mirhosseini, Austin Z. Henley, and Chris Parnin. 2023. What Is Your Biggest Pain Point? An Investigation of CS Instructor Obstacles, Workarounds, and Desires. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023)*. Association for Computing Machinery, New York, NY, 291–297. <https://doi.org/10.1145/3545945.3569816>
- [41] Samim Mirhosseini and Chris Parnin. 2020. Docable: Evaluating the Executability of Software Tutorials. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (Virtual Event) (ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, 375–385. <https://doi.org/10.1145/3368089.3409706>
- [42] Matt O’Leary. 2020. *Classroom Observation: A Guide to the Effective Observation of Teaching and Learning* (2nd ed.). Routledge, London, UK.
- [43] José Carlos Paiva, José Paulo Leal, and Álvaro Figueira. 2022. Automated Assessment in Computer Science Education: A State-of-the-Art Review. *ACM Transactions on Computing Education* 22, 3, Article 34 (June 2022), 40 pages. <https://doi.org/10.1145/3513140>
- [44] Candy Pang, Abram Hindle, and Denilson Barbosa. 2020. Understanding DevOps Education with Grounded Theory. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET '20)*. Association for Computing Machinery, New York, NY, 107–118. <https://doi.org/10.1145/3377814.3381711>
- [45] Maria do Mar Pereira. 2016. Struggling within and beyond the Performative University: Articulating activism and work in an “academia without walls”. *Women’s Studies International Forum* 54 (2016), 100–110. <https://doi.org/10.1016/j.wsif.2015.06.008>
- [46] W. Michael Petullo. 2022. Courses as Code: The Aquinas Learning System. In *Proceedings of the 15th Workshop on Cyber Security Experimentation and Test (Virtual Event) (CSET '22)*. Association for Computing Machinery, New York, NY, USA, 30–38. <https://doi.org/10.1145/3546096.3546099>
- [47] George M. Piskurich. 2006. *Rapid Instructional Design: Learning ID Fast and Right* (second ed.). Pfeiffer, San Francisco, CA.
- [48] Eric Ries. 2011. *The Lean Startup: How Today’s Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. Crown Publishing Group, New York, NY.
- [49] Julianna Rodriguez, Christopher Apsey, Sarah Rees, Todd Boudreau, and George Raileanu. 2018. Courseware as Code: Setting a new bar for transparency and collaboration. In *2018 IEEE Frontiers in Education Conference (FIE)*. IEEE, 4 pages. <https://doi.org/10.1109/FIE.2018.8658928>
- [50] Arnold Rosenbloom, Sadia Sharmin, and Andrew Wang. 2017. GIT: Pedagogy, Use and Administration in Undergraduate CS. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education (ITICSE '17)*. Association for Computing Machinery, New York, NY, 82–83. <https://doi.org/10.1145/3059009.3072980>
- [51] Zaynab Sabagh, Nathan C. Hall, and Alenoush Saroyan. 2018. Antecedents, correlates and consequences of faculty burnout. *Educational Research* 60, 2 (2018), 131–156. <https://doi.org/10.1080/00131881.2018.1461573>
- [52] Laura Schauer, Robert Stewart, and Manuel Maarek. 2024. Integrating Canvas and GitLab to Enrich Learning Processes. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training (Lisbon, Portugal) (ICSE-SEET '24)*. Association for Computing Machinery, New York, NY, USA, 180–190. <https://doi.org/10.1145/3639474.3640056>
- [53] Bruce Arne Sherwood and Jill H. Larkin. 1989. New Tools for Courseware Production. *Journal of Computing in Higher Education* 1, 1 (March 1989), 3–20. <https://doi.org/10.1007/BF02942603>
- [54] Aaron Daniel Snowberger and Choong Ho Lee. 2023. A Workflow for Practical Programming Class Management Using GitHub Pages and GitHub Classroom. *Journal of Practical Engineering Education* 15, 2 (Aug. 2023), 331–339. <https://doi.org/10.14702/JPEE.2023.331>
- [55] Cynthia Taylor, Jaime Spacco, David P. Bunde, Zack Butler, Heather Bort, Christopher Lynly Hovey, Francesco Maiorana, and Thomas Zeume. 2018. Propagating the Adoption of CS Educational Innovations. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (Larnaca, Cyprus) (ITICSE 2018 Companion)*. Association for Computing Machinery, New York, NY, USA, 217–235. <https://doi.org/10.1145/3293881.3295785>
- [56] Josh Tenenbergh and Sally Fincher. 2007. Opening the Door of the Computer Science Classroom: The Disciplinary Commons. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '07)*. Association for Computing Machinery, New York, NY, 514–518. <https://doi.org/10.1145/1227310.1227484>
- [57] Asher Trockman, Shurui Zhou, Christian Kästner, and Bogdan Vasilescu. 2018. Adding Sparkle to Social Coding: An Empirical Study of Repository Badges in the npm Ecosystem. In *Proceedings of the 40th International Conference on Software Engineering (Gothenburg, Sweden) (ICSE '18)*. Association for Computing Machinery, New York, NY, 511–522. <https://doi.org/10.1145/3180155.3180209>
- [58] Luther Tychonievich and Mark Sherriff. 2022. Engineering a Complete Curriculum Overhaul. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education - Volume 1 (Providence, RI, USA) (SIGCSE 2022)*. Association for Computing Machinery, New York, NY, USA, 453–459. <https://doi.org/10.1145/3478431.3499287>
- [59] Grant Wiggins and Jay McTighe. 2005. *Understanding by Design* (expanded 2nd ed.). Association for Supervision and Curriculum Development (ACSD), Alexandria, VA.
- [60] Chris Wilcox. 2015. The Role of Automation in Undergraduate Computer Science Education. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)*. Association for Computing Machinery, New York, NY, 90–95. <https://doi.org/10.1145/2676723.2677226>
- [61] Alexey Zagalsky, Joseph Feliciano, Margaret-Anne Storey, Yiyun Zhao, and Weiliang Wang. 2015. The Emergence of GitHub as a Collaborative Platform for Education. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing (CSCW '15)*. Association for Computing Machinery, New York, NY, 1906–1917. <https://doi.org/10.1145/2675133.2675284>
- [62] Micah Zeller and Emily Symonds Stenberg. 2016. Faculty Require Online Distribution of Student Work: Enter the Librarian. (Dec. 2016), 31–52.