Backward Design: An Integrated Approach to a Systems Curriculum

Michael S. Kirkpatrick, Mohamed Aboutabl, David Bernstein, Sharon Simmons
Department of Computer Science
James Madison University
{kirkpams,aboutams,bernstdh,simmonsj}@jmu.edu

Categories and Subject Descriptors

K.3.2 [**Computing Milieux**]: Computer and Information Science Education—*Computer science education*

General Terms

Human Factors

Keywords

ACM curriculum; systems fundamentals; backward design

ABSTRACT

This paper summarizes our experiences restructuring a core portion of our required courses for majors. Internal and external reviews of our program highlighted areas of concern in our "systems core," including inconsistent student outcomes, missing required material, and inadequate opportunities for programmatic assessment. To fix these problems, we initiated a curricular review to redesign these courses. Our novel approach employed a process known as *backward course design* that starts with desired student outcomes and works backward toward defining content coverage. By extending this design approach to our curriculum as a whole, we have defined a new systems core structure that has tightly integrated curriculum assessment opportunities. The result is a new systems core that changes almost 1/3 of the required courses for the major. This new structure provides increased student control over their learning goals while defining a consistent foundation of systems fundamentals; it also has tightly integrated program objectives and assessments. Applying the backward design philosophy to the curriculum, rather than to a single, pre-defined course, is both a rewarding and challenging experience. In this paper, we describe this approach, summarize the results of the process, and map the outcomes to the ACM 2013 curriculum. We also provide advice and lessons learned for others who may consider such an undertaking.

SIGCSE'15, March 4–7, 2015, Kansas City, MO, USA.

ACM 978-1-4503-2966-8/15/03 ...$15.00.
http://dx.doi.org/10.1145/2676723.2677264.

1. INTRODUCTION

Each program at James Madison University (JMU) is required to complete an internal assessment every year and an external review every five years. Additionally, we work with an external advisory group (EAG) to identify strengths and weaknesses of our program on an annual basis. In recent years, these reviews have highlighted certain aspects of our "systems core" courses (CS 350: Computer Organization, CS 450: Operating Systems, and CS 460: TCP/IP Networks) as areas of concern. Problems included overly broad specifications that led to different student experiences and insufficient programming experiences. Furthermore, our curriculum provided inadequate exposure to newer areas, such as parallel and distributed computing.

Over several years, we attempted a number of modest, incremental changes to address these problems. However, we found these changes failed to achieve all of our desired goals and a more extensive re-evaluation of our curriculum was necessary. Initial discussions of what topics should be covered revealed that we had different views on this matter, and we found that simply enumerating the desired topics by choosing pre-defined objectives from the ACM 2013 Knowledge Areas [1] failed to address our concerns.

Instead, we adopted a *backward design process* similar to an approach for course re-design [3]. In this approach, an instructor begins by identifying desired (or required) student learning outcomes and situational factors that influence the success of achieving these goals. From the outcomes, the instructor defines summative assessment strategies that provide evidence that the outcomes are achieved. Finally, the instructor creates activities and formative assessments that are designed to help students reach desired goals. Thus, assessment and content specifications are tightly integrated to ensure the course activities are aligned with desired outcomes. While our approach is based on Fink's work [3], backward design has been used by others in the education literature [10, 6].

Our work was also motivated by a desire to support learner-centered teaching [9] more explicitly. In this view, the instructor's responsibility changes from dictating the material that students must learn to guiding students through the learning process. In addition, the focus changes from learning the content for its own sake to *using and applying* it. Consequently, learner-centered teaching approaches emphasize active learning techniques [4, 8, 5]. The backward design process, in which content is specified to be deliberately aligned with student learning outcomes, is well-suited for supporting learner-centered teaching and active learning.

The contribution of this work is to describe our experiences with adapting the backward design process to a large-scale curriculum initiative. That is, the backward design process is normally applied to a single course with pre-defined learning outcomes. However, we applied this process to a significant portion of our major curriculum as a whole with no pre-defined learning outcomes that must be achieved. While the ACM 2013 curriculum recommendations provide general guidance for what should be covered, our goals were to re-design a core portion of our major requirements to be well-aligned with our students' backgrounds and career aspirations. Additionally, the backward design process explicitly creates a set of summative assessments that can be used as embedded assessments for systematic program evaluation.

In this paper, we provide background information on the courses that we targeted and how they fit into the major. We then describe our adaptation of the backward design process and our new systems core structure. We also provide advice for others who wish to use our approach to re-examine their curriculum. Overall, we find that the backward design process was a rewarding yet challenging experience, and we feel more confident in the structure of our curriculum and our ability to assess its efficacy. However, we also feel that certain aspects of the Fink model are rather awkward for CS outcomes, and other approaches based on the work of Wiggins & McTighe or Hansen could be more easily applied.

2. CURRICULUM BACKGROUND

CS 350 (Computer Organization) was criticized for inconsistent student experiences. A major contributing factor in this regard was our acceptance of transfer credit from different community colleges in the state. We observed that incoming transfer students were being offered vastly different experiences in the equivalent version of CS 350, particularly in regard to the amount and rigor of programming experiences. Some versions employed extensive C and assembly language assignments, while others had only Java programming or no programming at all. A lack of clear standards regarding the expectations of this course (and acceptable equivalents) was a primary cause.

Recent changes to CS 450 (OS) exacerbated the problems that arose from the CS 350 inconsistencies. While CS 450 traditionally operated as an introduction to systems programming (*e.g.,* developing command-line systems utilities and shells), some recent offerings focused on implementing advanced kernel code. Students who completed CS 350 without gaining exposure to C found themselves unprepared for the added rigor of these newer sections of CS 450.

The criticisms of CS 460 (TCP/IP Networks) arose from the unique history of the course. In the early 2000s, this course became cross-listed with another department, due to decreased enrollments in both, and an agreement was put in place that it would involve no programming component and would focus exclusively on the definition, modeling, and use of protocols. Over time, our faculty found this policy undesirable, as we consider network-based programming to be a fundamental skill for our graduates. Some students got exposure to that material in elective courses, but most students did not.

A common criticism of all three courses was the lack of clear specifications, thus making systematic program assessment challenging. While all faculty were adhering to the official course description, we had no explicitly defined objectives to assess student outcomes and to ensure consistent offerings. Anecdotal evidence from student exit interviews indicated that students perceived significant differences depending on the instructor.

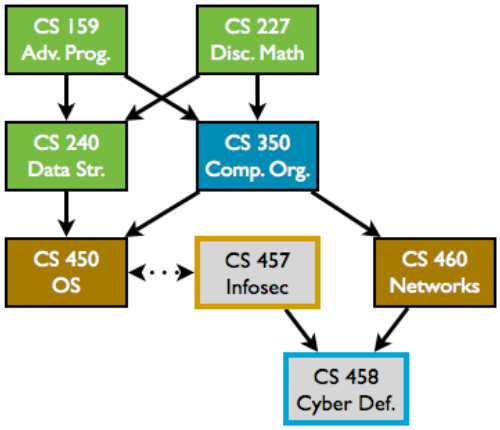

Figure 1: Existing systems core prerequisites

The timing of our courses also presented scheduling challenges for students. CS 350 was only offered in spring semesters, and was a required prerequisite for both CS 450 and CS 460, both offered in the fall. As a result, CS 350 acted as a bottleneck within our curriculum. Students who switched into the major after their freshmen year had difficulty graduating on time. Similarly, students who struggled with CS 350 (which was significantly more difficult than its prerequisite CS 159 – Advanced Programming) had to wait a year to retake the course. Thus, our new structure needs to avoid such a scheduling bottleneck.

Lastly, we felt that our existing course structure was inadequate for the rapidly evolving state of the field. Many concepts, such as principles of locality and communication, are common to several types of systems. In our view, the siloed approach (OS in one course, networking in another) obscured these common themes. Consequently, we desired a course structure that reflected the common foundations and themes of systems. We note that this view is aligned with that of the ACM, which introduced the Systems Fundamentals (SF) Knowledge Area in the 2013 curriculum [1].

3. BACKWARD DESIGN PROCESS

In this section, we will outline the major phases of the backward design process and highlight relevant observations of our experiences.

3.1 Situational Factors

A fundamental preliminary step in the backward design process is identification of situational factors that influence what can and cannot be achieved. We had two factors that restricted our options. First, our department maintains a "three-section, two-prep" teaching policy, where each faculty member teaches three 3-credit-hour sections per semester, with two sections being from the same course. Deviating from this policy would not be considered acceptable by the department.

The second restriction resulted from our relationship with the Virginia Community College System (VCCS). Through a combination of state and institutional policies, we must provide a feasible path for VCCS transfer students to com-

Foundational 1. What key information should our students understand and remember in the future? 2. What key ideas or themes are important for students to comprehend and describe?	**Human dimension** 7. What should students learn about themselves as software developers, users of technology, or as students? 8. What should students learn about understanding others and how they interact with others (including through developed software)?
Application 3. What kinds of thinking (critical, creative, practical) are important for our students to learn? 4. What software development skills do students need to gain? 5. What skills do our students need to learn about managing complex projects?	**Caring** 9. What changes and values should students adopt in regard to interests or ideas?
Integration 6. What connections should students recognize and make within the curriculum or to professional life?	**Learning how to learn** 10. What should students learn about how to work with computer systems? 11. What should students learn about becoming a self-directed learner?

Table 1: Guiding questions for defining learning outcomes

plete a degree in a timely manner. In our current structure, the *prerequisite depth* (*i.e.,* the minimum number of semesters from the first required course to the last required course) was 5, which is at the high end to offer a feasible path for timely completion. Thus, if our final proposal increased this depth, we would need to have a strong rationale and institutional acceptance of the curriculum would be unlikely. Figure 1 shows the relevant prerequisite structure (which is a subset of our overall curriculum) for our systems core. The courses in green are offered every semester, those in blue are spring only, and those in brown are fall only. The two courses in grey are electives, with CS 457 as a corequisite with CS 450.

Another situational factor arose from one of our department's strengths: As one of the seven original National Centers of Academic Excellence in Information Assurance Education (CAE/IAE), our department has a long history of offering multiple courses in information security and cyber defense. These courses require completing the systems core as prerequisites. Lengthening the duration of the core could make it more difficult for students to enroll in these courses, thus reducing the impact of a central departmental priority.

Lastly, the career aspirations of our students created another significant factor. Like many other institutions, our students overwhelmingly go into industry positions rather than graduate programs, though some opt for the latter. Furthermore, given JMU's proximity to Washington, D.C., and our emphasis on information security, many of our graduates go into positions with the federal government. This bias toward industry and government is reflected in our overall curriculum, which emphasizes applied software engineering experience. Thus, our goal for the systems core was to create a foundation of systems knowledge for all students while providing ample opportunities for exploring certain systems in greater detail, particularly for students with an interest in post-graduate work.

3.2 Systems Curriculum Goals

As a starting point for our curriculum re-design, we applied the Fink taxonomy [3] of learning goals to enumerate *all desired outcomes without consideration of feasibility.* That is, we began by asking what we felt all of our students needed to achieve without regard to how much time would be needed to achieve these goals. At this stage, we did not consider how our goals would impact the prerequisite depth issues for VCCS students or our security courses.

To define our desired learning outcomes, we adopted the Fink taxonomy [3]. This model defines 6 areas of learning outcomes as shown in Table 1. In Fink's model, 3 areas (Foundational, Application, Integration) roughly correspond to the cognitive domain concerns underlying Bloom's taxonomy [2]. Fink's model incorporates 3 additional areas that emphasize affective and metacognitive questions that have no direct mapping to counterparts in Bloom's taxonomy. We found defining learning goals for the cognitive areas to be a rather straightforward endeavor, as it is common practice to define cognitive outcomes. Through casual discussions, we had developed an informal understanding of our goals and values. However, enumerating them allowed us to identify mismatches in expectations and assumptions, thus making it possible to come to a consensus on these issues.

The affective and metacognitive questions led to considerably more discussion. Most members of the group were new to this process and unfamiliar with Fink's work; consequently, we found it necessary to spend time coming to an agreement on how to interpret terminology (*e.g.,* what it means to be a "self-directed learner" in computing, what constitutes "caring," what role should a CS department have in addressing the human dimension). In fact, disagreements in later discussions could often be traced back to different views about these questions. Furthermore, standard curriculum descriptions provide considerably less guidance in defining such objectives. For instance, the ACM 2013 has a short chapter describing "Characteristics of Graduates" at a broad level, but does not define specific measurable objectives similar to those defined in the Knowledge Areas. As a result, we found ourselves spending a significant amount of time grappling with this discussion, and others considering this process should be prepared for potential conflicts arising from these questions.

3.3 Guiding Principles and Themes

After enumerating our goals for the systems curriculum[1], we re-examined the list to define a set of higher-level principles. In this regard, we distinguished between goals that were specific to the systems curriculum (our focus) and those that were more universal to all upper-level CS courses. Principles in the former regard focused on the need to develop a system-level perspective and on how platform specifications can impact performance and other requirements.

The primary value of this stage of the process was to identify and prioritize our major goals and themes. In later stages, we would map assessments and modules to these themes to ensure alignment of course definitions with our overall goals. This mapping became particularly valuable when considering the number of hours allocated to topics. In some cases, we found that our initial assignment devoted significant amounts of time to some topics that provided insufficient support for our major goals and themes; our theme list then served as an indispensible tool for resolving these conflicts.

3.4 Assessment Strategies

To ensure a consistent set of experiences for students, we defined a list of modules and specified summative assessments for each. For instance, summative assessment of binary representation of information would consist of writing a program that read in a binary file and displayed the data in a variety of formats. Assessment of information exchange would consist of writing programs that use sockets, IPC, or shared memory.

It is important to emphasize one subtle aspect of this phase of the backward design process: *The modules themselves had not yet been defined.* That is, we were only concerned with specifying how students could demonstrate mastery of the goals and themes we had previously outlined. We had not yet discussed what specific topics needed to be introduced for each module. For instance, in the information exchange module, we had not mentioned any specific protocols, such as TCP/IP. This is the heart of backward design: **Student learning outcomes and assessments must be defined *before* there is any mention of topics to be covered.** The focus of the discussion is on what the students should *do*, rather than what they should *know*.

3.5 Module Definitions

Upon completion of the assessment strategies, we could then begin to define the modules in greater detail. Specifically, our goal at this stage was to construct a sequence of required courses that provided a strong systems foundation for all students, while also providing an opportunity for more detailed study of specific types of systems as desired or needed. That is, our approach was similar to the distinction between Core Tier-1 and Core Tier-2 in the ACM 2013 curriculum.

The module definitions started with the assessments specified previously, then we added the specific instructional objectives that must be met in order to support students' attempts at these assessments. In addition, the module definitions specified what material would be considered prerequisite, what themes were addressed by the module, and how much time was needed. Our initial attempt would have required slightly more than two 3-credit hour courses, so we had to perform multiple iterations of this process. *The previously stated guiding principles were important tools for prioritizing modules in this phase.*

[1]Due to space constraints, we cannot list our goals, principles, themes, strategies, and module definitions here. These will be made available online and can be found by contacting the lead author.

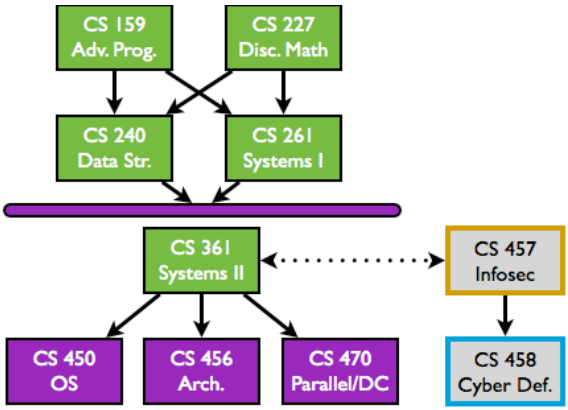

Figure 2: New systems core prerequisites

4. RESULTS

The end result of our work is a new systems core structure as shown in Figure 2. The modules defined as part of the backward design process were assembled into a two-semester Computer Systems I-II sequence. The purple courses denote new "advanced systems courses," that we will be introducing to complement the systems core. CS 240 and CS 261 are both required as prerequisites for CS 361 and CS 456, as denoted by the purple bar.

4.1 Computer Systems I-II

Table 2 shows the modules specified during the backward design process. For each module, we have specified summative assessments, hours of class time (approximate), skills expected, and assumed prerequisites. As shown in the mapping, these courses provide 100% coverage of Core Tier-1 topics in the Systems Fundamentals (SF), Architecture and Organization (AR), Networking and Communication (NC), Operating Systems (OS), Parallel and Distributed Computing (PD), and Computational Science (CN) Knowledge Areas. These courses also provide a foundation of several Core Tier-2 topics that we consider vital for all of our graduates.

Overall, the central themes of CS 261 (Computer Systems I) are the basics of instruction execution and concurrency. These are topics that generally correspond to traditional material in the AR and OS Knowledge Areas. CS 361 (Computer Systems II) focuses more on architectural design questions and communication between parallel software components, thus corresponding to the NC and PD Knowledge Areas.

4.2 Advanced Systems Courses

While our Computer Systems I-II courses provide a solid foundation in the area of systems fundamentals, this sequence alone is insufficient for our desired goals and student learning outcomes. Specifically, these courses provide a breadth of themes and concepts, but do not explore specific systems in detail. As such, we have also introduced three new courses in the areas of Architecture (CS 456), OS (CS

CS 261: Computer Systems I

Module name	*2013 mapping*	*Hours*	*Description*
C basics	n/a	6	Memory model and pointers, transition for students with Java experience
Compiler and debugger use	n/a	3	Introduction to GNU tools
CPU/memory organization	AR-Memory system (T2), PD-Architecture (T1/T2)	3	Registers and cache, principles of locality
Binary representation of data	AR-Machine level (T2)	4.5	Two's complement, IEEE 754, arithmetic instructions
von Neumann cycle	AR-Assembly level (T2)	3	Role of CPU and memory, load/store instructions
Basic circuits and datapath	AR-Interfacting (T2), SF-Paradigms (T1)	6	Logic gates, adders, ALU, control signals
Threads vs. processes	OS-Overview (T1), PD-Fundamentals (T1)	6	Fork vs. thread creation, memory space
Interrupts and OS principles	OS-Principles (T1), AR-Interfacing (T2)	4.5	Interrupts, system calls, kernel- vs. user-mode
System software design and evaluation	SF-Evaluation (T1)	4.5	Benchmarks, complexity, static vs. dynamic analysis

CS 361: Computer Systems II

Module name	*2013 mapping*	*Hours*	*Description*
Architecture analysis, design evaluation	SF-Reliability (T2), SF-Paradigms (T1)	3	Peer-to-peer, client-server, layered architectures
State models	SF-State machines (T1), OS-Concurrency (T2)	4.5	Notion of state, UML, FSM
Mathematical modeling	CN-Modeling (T1), DS-Probability (T1/T2)	3	Basic systems theory
Informaion exchange	PD-Communication (T1/T2), SF-Communication (T1), OS-Concurrency (T2)	6	Communication basics, IPC, sockets
Synchronization primitives	OS-Concurrency (T2)	6	Locks vs. semaphores, concurrency problems, deadlock
Parallel decomposition	PD-Decomposition (T1/T2), SF-Parallelism (T1)	3	Data vs. task, Amdahl's law, fork-join pattern, libraries
Protocol analysis, evaluation, and design	NC-Introduction (T1), SF-Cross layer communication (T1), PL-Reactive programming (T2), PD-Communication (T2)	9	Protocols and services, timing and statecharts, push/pull, flow control, reliability, handshaking, metrics
The Internet	NC-Applications (T2), NC-Reliable data delivery (T2), NC-Routing (T2), NC-LAN (T2), NC-Mobility (T2)	6	HTTP, DNS, DHCP, TCP, UDP IP, 802.3, 802.11, ARP

Table 2: Course definitions for new Computer Systems I-II courses

450), and Parallel and Distributed Computing (CS 470). CS 456 and the new CS 450 include a number of advanced topics that we previously covered in CS 350 and the old CS 450, but are not in the new Systems I-II sequence. These topics are identified as Core Tier-2 material in the ACM 2013 report. CS 470 is an entirely new course.

The key feature that distinguishes the advanced systems courses is the requirement of programming-in-the-large projects, particularly ones that require students to read and comprehend an existing code base. One example would be the Pintos OS projects [7]. Instructors will integrate the fundamental concepts from the Computer Systems I-II sequence throughout the systems core.

4.3 Course Offerings

This new structure offers a number of advantages over our previous curriculum. Whereas CS 350, CS 450, and CS 460 were offered only once per year (thus making CS 350 a bottleneck), CS 261 and CS 361 will both be offered every semester. Consequently, struggling in one or both of these courses will have significantly less impact on students' ability to progress in the major. This new structure also leaves the prerequisite depth at 5, and does not add any time to the security courses. In fact, by eliminating the bottleneck of CS 350, this new structure makes it easier for students to progress on time.

As a major requirement, students must complete one of the advanced systems courses[2], though students will be encouraged and advised to complete more than that. Pro-

[2]Achieving the ACM 2013 recommendation of 80% coverage of Core Tier-2 material would require completing at least three of the advanced systems courses. We consider such a goal infeasible given certain constraints of our curriculum. We note, though, that our previous curriculum also fell short

viding students with these options supports the learner-centered aim of granting students more control over their educational goals. These courses offer scheduling flexibility, with three sections (typically from two different courses) each semester. Furthermore, CS 456 may be taken concurrently with CS 361 for students on a compressed schedule.

5. DISCUSSION

Overall, we are pleased with the process and the end result of this curriculum initiative. Having explicitly defined summative assessments that can be used as embedded questions provides opportunities for systematic program assessment, thus allowing us to ensure consistent student experiences. The new curriculum structure provides greater student choice and avoids scheduling bottlenecks. This structure also maintains a feasible path for VCCS transfer students to complete the curriculum in a timely manner and places no additional barriers to the security courses. The Computer Systems I-II courses provide a breadth-first foundation in systems fundamentals, complemented by a number of courses to explore individual types of systems in depth. This new curriculum is much better aligned with our departmental goals and priorities.

While we strongly encourage backward design as a process for curriculum re-design, we feel less strongly about the details of the Fink taxonomy. In particular, we found the "human dimension" and "caring" areas to be challenging to work with and interpret. Wiggins & McTighe [10] use the terms "perspective" and "empathy" instead, and we find these terms more suitable for CS curricula, particularly for systems courses. That is, we find considerable overlap between these dimensions of both the Fink and the Wiggins & McTighe taxonomies, and our preference is for the latter. For instance, we find it easier to answer questions relating to the perspective of users vs. the perspective of developers, rather than questions of "learning about oneself," which we found to be more difficult to use for crafting measurable objectives. Alternatively, Hansen [6] refers to Big Ideas and Enduring Understandings, rather than a multidimensional taxonomy. While Hansen's model is more intuitive, it (like Bloom's) provides inadequate emphasis on metacognitive and affective goals. It is our view that these non-cognitive domains are critical and often overlooked. Perhaps the best approach would be to use the Wiggins & McTighe model to enumerate objectives, then use Hansen to define themes to unite modules.

Finally, it is important to note that adhering to this process requires significant time and effort. We performed this work over the course of an academic year, generally meeting twice a month. The process was initially slow-moving, as most members of the committee had no experience with the backward design process. As such, we spent considerable time discussing the process and whether or not a more streamlined approach would yield acceptable results. Ultimately, we found that the backward design process, while time-consuming and challenging, was worth the effort and our end result is superior to what would have been achieved by selecting pre-defined objectives. Thus, we would advise others who are re-examining their curriculum to consider the backward design process as an approach. The process facilitates discussions about departmental priorities, program assessments strategies, and instructional techniques, and has several beneficial side effects beyond just the curriculum changes.

of this 80% goal, and our new version is significantly improved in this regard.

6. CONCLUSION AND FUTURE WORK

In this work, we described our experiences using the backward design process to re-design a significant portion of our major requirements. This approach, which integrates assessment and content specifications, provides a framework for creating a learner-centered curriculum that ensures course instructional objectives are aligned with departmental goals and situational factors. We summarized our existing curriculum and how we applied the process to our re-design. We summarized our new course structure and provided advice for others considering this approach.

The current status of our work is that we have received preliminary approval from our department and we are working toward institutional approval. Our intent is to put this curriculum in place beginning with the 2016–17 academic year (with 2015–16 as a transitional step). In the mean time, we will be collecting baseline assessment data from students who complete the existing curriculum. In future work, we will compare these results with assessment data from the new curriculum.

7. REFERENCES

[1] ACM. Computer science curricula recommendations. `http://www.acm.org/education/curricula-recommendations`, 2013. [Online; accessed June 29, 2014].

[2] B. S. Bloom, J. D. Engelhart, E. J. Furst, W. H. Hill, and D. R. Krathwohl. *Taxonomy of Educational Objectives: The Classification of Educational Goals, Handbook I: Cognitive Domain.* David McKay, New York, 1956.

[3] L. D. Fink. *Creating Significant Learning Experiences.* Jossey-Bass, San Francisco, 2003.

[4] S. Freeman, S. L. Eddy, M. McDonough, M. K. Smith, N. Okoroafor, H. Jordt, and M. P. Wenderoth. Active learning increases student performance in science, engineering, and mathematics. *Proceedings of the National Academy of Sciences*, 2014.

[5] R. R. Hake. Interactive-engagement vs. traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses. *American Journal of Physics*, 66(1):64–74, 1998.

[6] E. Hansen. *Idea-Based Design: A Course Design Process to Promote Conceptual Understanding.* Stylus Publishing, Sterling, VA, 2011.

[7] B. Pfaff, A. Romano, and G. Back. The pintos instructional operating system kernel. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, SIGCSE '09, pages 453–457, New York, NY, USA, 2009. ACM.

[8] M. Prince. Does active learning work? a review of the research. *Journal of Engineering Education*, 93(3), July 2004.

[9] M. Weimer. *Learner-Centered Teaching.* Jossey-Bass, San Francisco, 2002.

[10] G. Wiggins and J. McTighe. *Understanding by Design.* Pearson, Upper Saddle River, NJ, 2005.