Using Outcomes-based Assessment as an Assurance Tool for Assurance Education

S. Older and S.K. Chin

Department of Electrical Engineering and Computer Science
Systems Assurance Institute
Syracuse University
Syracuse, NY 13244, USA
Email: sueo@ecs.syr.edu, skchin@syr.edu

Abstract

We discuss our efforts to deliver a graduate-level assurance curriculum with a strong emphasis on logic and formal methods. Specifically, we describe what we are teaching in two of our foundational courses, as well as what our students are learning. We also advocate the use of an outcomes-based approach when developing IA courses and curricula. We have found that focusing on the desired educational outcomes from the outset has made it easier to identify what is working and what is not, and we wish to share our experiences.

Introduction

The goal of Syracuse University's Certificate of Advanced Study in Systems Assurance (CASSA) program is to develop students who (1) comprehend the concepts underlying security and system assurance; (2) can apply those concepts to construct assured systems; and (3) can critically analyze and evaluate systems' conformance to their requirements. Because of this third requirement, a key component of the CASSA program is an emphasis on using formal mathematics and logic to provide a rigorous basis for the assurance of information and information systems. The CASSA program exists within the framework of the Computer Science and Computer Engineering Master's programs: students must satisfy all requirements of their home Master's program, as well as satisfying CASSA-specific requirements. In particular, all students must take a combination of courses that provides hands-on experience in both systems building and in using formal methods to analyze and evaluate system behavior.

Our purpose in writing this paper is twofold. The first is to report on our progress in delivering an Information Assurance (IA) curriculum with a strong emphasis on logic and formal methods. In (Older and Chin, 2002), we described our experiences in developing the CASSA program, as well as the challenges inherent in incorporating mathematical and logical rigor into an IA curriculum. Here, we provide more details about what we are doing in our courses, as well as our ongoing attempts to answer the following questions:

To what degree are we being successful? What are our students learning?

We discuss two specific courses that serve as elective courses in the CASSA program: *Modeling Concurrent Systems* (CIS 632) and *Principles of Network Security* (CSE 774). For each course, we describe the desired educational outcomes and what we have done to realize and measure those outcomes. We also discuss our observations of student achievement (both subjective and objective), and the changes we have made or intend to make as a result of

ISSN 1445-3312 print/ISSN 1445-3347 online

these observations. We hope that our experiences are informative for others interested in introducing formal methods and logical rigor into an IA curriculum.

The second and broader purpose is to advocate the use of an outcomes-based approach when developing IA courses and curricula. To be explicit, we do not wish to promote additional bureaucracy or the complex instrumentation of courses. Rather, we have found that focusing on desired educational outcomes from the start has made it easier to identify what is working and what is not, and we wish to share our experiences.

In our opinion, the outcomes-based approach is especially useful for developing IA curricula. IA is such a broad field that no individual program can cover everything. Focusing on desired outcomes helps identify which topics to include and in what depth. Furthermore, the greater specificity of desired outcomes allows for more detailed and precise discussions across disciplines. Ultimately, at the core of assurance is the mandate to guarantee with high confidence the quality of the resulting outcomes. We believe that we would be remiss if we failed to apply the underlying principles of assurance to all of our scholarly efforts. Just as we apply these principles with rigor to the design and evaluation of systems, we believe we should apply them to the design and assessment of our educational programs.

The rest of this article proceeds as follows. First, we describe the high-level educational outcomes of our CASSA program and how the requirements relate to these outcomes. We then illustrate how one of the high-level CASSA outcomes is addressed through two separate courses. The third section presents the course *Modeling Concurrent Systems*, an applied concurrency-theory course in which IA applications are an integral component. The fourth section focuses on *Principles of Network Security*: this course uses a variety of logical systems to describe, verify, and analyze properties related to network security. In the fifth section, we discuss our observations from these two courses, as well as some of the curricular adjustments that have arisen from these observations. We conclude with some final comments.

CASSA Educational Outcomes

The EECS Department has adopted an outcomes-based approach to curricula design (Syracuse 1998), in which we first formulate the outcomes we desire of our graduates and then use those outcomes to guide the development of rational curricula. This approach represents a shift from a traditional faculty-centered viewpoint of `*what do we teach?*' to a student-centered viewpoint of `*what do our students learn?*'

We made use of this approach in developing the CASSA program, focusing on our expectations for those students who successfully complete our program. No single curriculum can possibly address all of IA: concentrating on the desired educational outcomes helped us determine how to structure our program. In our case, the goal was to develop a coherent collection of courses to ensure the following outcomes:

1. Students comprehend the concepts underlying security and systems assurance.

2. Students can apply those concepts to construct assured systems.

3. Students can critically analyze and evaluate systems' conformance to their requirements.

Not surprisingly, these outcomes reflect the EECS department's long-standing emphasis on the use of mathematical and logical methods in engineering, computer science, and security.

These educational outcomes are broadly addressed by the CASSA program requirements. For example, to address Outcome 1, students must successfully complete both the Systems Assurance Seminar and a non-technical IA elective course (such as telecommunications policy, Internet law, or e-commerce). The seminar course serves as a gateway to more advanced assurance courses, introducing basic terminology and many of the nonmathematical issues related to information assurance. Furthermore, students must successfully complete a total of five courses from the *Foundations for Assurance* and *Assurance Applications* tracks, including at least two courses from each track. The *Foundations* track includes courses in cryptography, concurrency theory, and logical principles of network security. The *Applications* track includes courses such as Internet security, computer security, and security of wireless networks. Via this requirement, students are assured to receive hands-on experience both in building systems and in using formal methods, thus addressing Outcomes 2 and 3.

This setup raises an obvious question: what *specifically* can students do in the areas of constructing assured systems and critically analyzing and evaluating systems' conformance to their requirements? In this paper, we focus on the formal-methods aspect and describe our experiences assessing our students' learning in two separate *Foundations for Assurance* courses. We realize that, when the topics of outcomes and assessment are initially brought up, the typical reaction is resistance, due to a belief that it will be cumbersome, bureaucratic, and of little practical use. This has not been our experience. We have found that a good way to start is to follow Diamond's suggestion (1998, page 134):

> As an alternative to writing objectives in the abstract, ... [one can] develop strong, clear objectives by playing the role of the student and asking, "If I'm your student, what do I have to do to convince you that I'm where you want me to be at the end of this lesson, unit, or course?"

This approach is exemplified by the description of *Modeling Concurrent Systems* in the next section. One can also take a more structured approach suggested by Diamond (1998, page 132) and write objectives that include `a verb that describes an observable action' and `a description of the conditions under which the action takes place: "when given x, you will be able to ..." '. This approach is used in *Principles of Network Security,* which is described in the fourth section.

CIS 632: Modeling Concurrent Systems

The purpose of *Modeling Concurrent Systems* is to provide students with an in-depth understanding of the process-algebraic approach for specifying, modeling, and analyzing system behavior. Process algebras such as CSP (Hoare 1985) and CCS (Milner, 1980) provide a way to describe system behavior in terms of the events (i.e., abstract actions deemed *observable*) that can occur. The underlying theory also includes several useful notions of program equivalence and refinement, which are useful for compositional reasoning and analysis. There are automated and semi-automated tools available that allow one to apply the theory in practice to verify properties of nontrivial applications.

Educational Outcomes and Course Content for CIS 632

In the Fall of 2002, we focused on the use of CSP to specify and verify concurrent systems and to understand the emergent behavior of such systems. The educational outcomes appear in Table 1.

After completing this course, you should be able to:
• Use CSP or related calculi to write process descriptions and behavioral specifications.
• Use traces and failures refinement to analyze and relate system behaviors at multiple abstraction levels.
• Distinguish between the traces and failures models, and explain when each is appropriate to use.
• Use the model checker FDR to verify valid refinements or debug invalid refinements.
• Apply these techniques to specify and analyze nontrivial applications

Table 3: CIS 632 Educational Outcomes

We used Steve Schneider's CSP textbook (2000), supplemented with several exercises and examples from Bill Roscoe's textbook (1998). In the first portion of the semester, we introduce the basic operators of CSP, their operational semantics, and the trace and failures models, both of which induce notions of refinement. The trace model, for example, formally describes a process's behavior in terms of the set of traces—that is, sequences of events—that it can perform. These trace sets provide a basis for comparing, equating, and refining processes. Two processes are trace-equivalent when they have precisely the same sets of traces. A process Q refines P in the trace model (written $P \leq_T Q$) if every trace of Q is also a trace of P. Intuitively, if P corresponds to a specification of permissible behavior and Q refines P, then Q is guaranteed to exhibit only permissible behaviors.

Similarly, the failures model supports a notion of refinement based on a process's set of failures, which pair traces with sets of events. A process has the failure (t,X) if it can perform the trace t and then reach a state where its only possible actions involve events not in the set X. The failures model provides a finer notion of equivalence and refinement than the trace model does: it distinguishes processes not only by what they are able to do but also by what they are able to refuse to do. As a result, it supports reasoning about the potential for deadlock. A process Q refines P in the failures model (written $P \leq_{SF} Q$) if $P \leq_T Q$ and every failure of Q is also a failure of P.

We also introduced the model checker FDR2 (Formal Systems, 1997), which provides automated support for applying these notions in practice. The advantage of using a model checker is that one can analyze much larger systems than is possible by hand. In addition, model checkers can be useful in debugging designs: for example, when a desired refinement $P \leq_T Q$ fails, FDR2 produces a witness trace of Q that is not allowed by the specification P.

The final third of the course was spent on using CSP and FDR2 to analyze several different problems and protocols, such as the alternating-bit protocol and a distributed-database cache-coherency protocol. For three class periods, students worked in small groups of 4-5 students to specify and implement a one-lane bridge that safely supported bi-directional traffic. We also spent two lectures on the Needham-Schroeder key-exchange protocol and Gavin Lowe's (1996) use of CSP and FDR2 to uncover a previously unknown flaw in it. This analysis uses

CSP to model the system as the parallel composition of an initiator, responder, and intruder; CSP is also used to describe the desired authentication properties. FDR2 can then be used to try to validate refinement relationships between the system and the two desired properties. The result of FDR2's analysis is a witness trace that highlights a possible man-in-the-middle attack.

Assessment in CIS 632

Grades for this course were based on an equally weighted combination of homework assignments, quizzes, a final exam, and a final project. The purpose of homework was to keep students up-to-date with the material discussed in class and to familiarize them with the tools. The homework also served as simple formative-assessment tools: if several students had questions while working on the assignments or did poorly on particular questions, there was indication that certain topics needed to be reviewed again.

In total, there were six homework items. The first two homework items concentrated on sequential processes. The first homework required students to write CSP processes for a variety of scenarios, including one of their own choosing. For the second homework, students wrote machine-readable CSP and used the process animator ProBE (Formal Systems, 1998) to interact with their processes and to test their understanding of the CSP transition rules. The third and fourth homework items focused on the various parallel operators of CSP: the third homework assessed basic understanding of the operators, while the fourth homework challenged the students to use the operators to introduce constraints on a system. The final two homework items assessed students' understanding of the primary abstraction operators (i.e., hiding and renaming), as well as their mastery of the traces and failures models.

The quizzes and the final exam—all closed book and closed notes—served as summative-assessment tools, letting students demonstrate their understanding of the fundamental concepts. The quizzes were relatively lightweight and intended primarily as sanity checks. Typically, 60-75% of a quiz's points were for basic understanding of fundamentals (e.g., drawing transition graphs of processes, identifying a process's set of traces or failures, determining simple refinement relationships between processes). The remaining points tested students' deeper understanding of the concepts, such as writing a CSP process to model a scenario, validating or refuting claims about refinement relationships, and generating CSP processes that have (or fail to have) certain properties. The final exam placed more weight on these latter sort of questions: 34 points (out of 100) were for fundamentals, 30 were for writing CSP to model a scenario, and 36 were for validating/refuting a variety of claims about refinement relationships. Two examples of these claims follow:

- If $f(P) \leq_{SF} f(Q)$ and f is a renaming function, then $P \leq_{SF} Q$.
- For all processes P, Q, and R, and for all sets X and Y, the processes $P \underset{X}{\|} \left(Q \underset{Y}{\|} R \right)$ and $\left(P \underset{X}{\|} Q \right) \underset{Y}{\|} R$ are trace equivalent.

The first claim states that every failures-refinement relationship that holds between renamed processes (the process $f(P)$ behaves like P, except for a renaming of events via the function f) must also hold between the original processes. The second claim states that distinct parallel-composition operators are necessarily associative with one another. Both claims happen to be false.

On the final exam, students had very little difficulty with drawing transition graphs or identifying a process's traces and failures. The average grade (out of 10 students) for the combination of these questions was 85%, with a high mark of 100% (two students), a low of 62%, three students between 74% and 77%, and the rest between 85% and 97%. Students were also asked to write a CSP process to describe a banking account that allowed deposits, withdrawals (with limited overdraft protection), and queries, as well as requests to change the overdraft limit. Students then had to add a parallel constraint to the system to limit the number of rejected overdraft-change requests. On this question, the average grade was 74%, with a high of 93% and a low of 53%; five students received between 60% and 73%, and three received between 87% and 90%. The final question required students to judge the validity of various claims and to provide convincing explanations for their answers. Students had much more difficulty with this question: most students correctly identified the truth or falsity of the claims (worth a third of the points) but provided insufficiently convincing explanations. Here, the average was 63%, with a high of 89% (two students) and a low of 36%; the remaining students received grades between 39% and 86%, in a fairly even distribution.

The final project required students to use machine-readable CSP and FDR to model and reason about a nontrivial system or protocol. Students were required to submit annotated machine-readable CSP scripts containing descriptions of their system and of the desired behavioral properties, as well as the assertions (e.g., refinement or deadlock checks) necessary for validating those properties. Students were also required to submit 8-10 page project reports containing the following features: a high-level description of both the problem they were solving/analyzing and their CSP solution; a description of their use of refinements (e.g., explaining why trace refinements were used instead of failures refinements, or vice versa); and some analysis of their experience (e.g., unexpected results or design choices that were particularly good or bad). This paper was graded for grammar, spelling, and style, as well as for content. Students analyzed a variety of protocols and algorithms, including the Bully election algorithm, the Needham-Schroeder and TMN security protocols, the two-phase commit protocol for distributed transactions, and a distributed-sum algorithm.

CSE 774: Principles of Network Security

Principles of Network Security is an analytical course that uses predicate calculus, higher-order logic, and specialized logical systems to describe, specify, and verify the correctness and security properties of network security protocols, algorithms, and implementations. Students use formal logic to rigorously analyze cryptographic algorithms, key-distribution protocols, delegation, access control, electronic mail, and networks of certification authorities.

Because this course is fairly novel even among IA curricula and has no standard textbook, we describe its contents in more detail than the course in the previous section.

Educational Outcomes and Course Content for CSE 774

Both the educational outcomes and our discussion about this course make use of a modified Bloom's (1974) taxonomy to classify and distinguish the many kinds of knowledge and abilities. Some kinds of knowledge are at relatively low levels (e.g., *recalling* that $15_{16} = 21_{10}$), while others are at very high levels (e.g., *evaluating* whether an implementation meets a specification and requirement). The outcomes, which appear in the course's online syllabus, are listed in Table 2.

Comprehension

- Define the meaning of security services such as confidentiality, integrity, non-repudiation, and authentication
- Describe the characteristics of private key and secret key cryptographic systems
- Describe cryptographic algorithms for encryption, hashing, and signing
- Describe cryptographic protocols for session-based security such as Kerberos, and store-and-forward security such as Privacy Enhanced Mail
- Describe basic principles of trust topologies and networks of certification authorities

Application

- When given a block diagram or functional description of an implementation, you should be able to represent the implementation using predicate calculus
- When given protocol descriptions and trust hierarchies, you should be able to use specialized security calculi such as the logic of authentication for distributed systems to describe the protocol and trust relationships
- When given a trust topology, determine the necessary certificates for establishing trust in a key

Analysis

- When given a set of assumptions and a goal to prove, you should be able to prove, using formal inference rules, if the security goal is true or not
- When given a set of certificates, you should be able to formally derive whether a key is associated with a particular principal

Synthesis

- When given a description of a system or component and its specification and security properties, you should be able to construct a theory that describes both, and show if the security properties are supported
- When given a description of a trust topology, you should be able to create a formal description of certificates and trust relationships for the certification authorities

Evaluation

- When given a theory, inference rules, and a proof, you should be able to judge if the proof is correct
- When given a specification and implementation, you should be able to judge whether the implementation satisfies its specification

Table 2: CSE 774 Educational Outcomes

The first part of the course (outlined in Table 3) covers standard network-security fundamentals: basic security properties, cryptographic algorithms (e.g., DES and RSA), authentication, hash functions, digital signatures, protocols and certificates. The primary reference is William Stallings' classic text (1999).

Topic	Primary References
Basic Security Properties: confidentiality, authentication, integrity, non-repudiation, access control, availability; mechanisms; attacks	Stallings 1999 §1 Also: Saltzer et al. 1975, Lampson 1971
Conventional Encryption: DES, Electronic Code Book, Cipher Block Chaining	Stallings 1999 §3.1 - §3.3, §3.7
Confidentiality: placement of encryption, traffic confidentiality, key distribution	Stallings 1999 §5.1 - §5.3
Public-key Cryptography: principles of public-key cryptosystems, RSA, key management	Stallings 1999 §6.1 - §6.3
Message Authentication and Hash Functions: authentication requirements, authentication functions, message authentication codes, hash functions	Stallings 1999 §8.1 - §8.4
Hash Functions: Secure Hash Algorithm (SHA-1)	Stallings 1999 §9.2
Digital Signatures and Authentication Protocols: digital signatures, authentication protocols, digital signature standard	Stallings 1999 §10.1 - §10.3
Authentication Applications: Kerberos, X.509 authentication service	Stallings 1999 §11.1 - §11.2

Table 3: CSE 774 Topics in Security Fundamentals

The second part of the course (see Table 4) deals with reasoning about freshness of protocols, replay attacks, and role-based access control. We use the BAN logic (Burrows et al., 1990) to reason about freshness and potential replay attacks. We also introduce the formal definitions and properties of role-based access control (RBAC), as described by Ferraiolo and colleagues (1992, 1999, 2000). The knowledge expected of students is at the levels of comprehension, application, analysis and synthesis. Specifically, when given informal descriptions of key-exchange protocols, students are expected to be able to describe the protocol abstractly in the BAN logic; postulate initial beliefs about key associations, scope of authority, freshness of nonces, and so on; and show that, at the end of the protocol, belief in the distributed keys has been established. A similar set of skills is expected related to RBAC: when given an organizational structure of roles, students are expected to write down role-containment relations and derive the membership relations that are implied by a specific organizational structure.

Topic	Primary Reference
BAN Logic	Burrows et al. 1990
RBAC Definitions and Properties	Ferraiolo et al. 1999 (also: 1992, 2000)

Table 4: CSE 774 Topics in Belief Logics and Role-Based Access Control

The third part of the course (outlined in Table 5) focuses on authentication, delegation, and access control in distributed systems. Nine weeks is spent on this topic, the major focus of the course. The technical content of this part centers on the Abadi calculus (Abadi et al. 1993, Lampson et al. 1992) for reasoning about principals, their statements, and their beliefs.

Topic	Primary References
Underlying Semantics and Model	Howell and Kotz 2000 §1 - §3 Also: Abadi et al. 1993 §3.3 - §3.4
Axioms for Principals and Statements	Lampson et al. 1992 §3, Howell and Kotz 2000 §4.1 - §4.3 Also: Abadi et al. 1993 §3.1 - §3.2
Channels and Encryption	Lampson et al. 1992 §4
Group Names	Lampson et al. 1992 §5.3
Roles and Programs	Lampson et al. 1992 §6 Also: Howell and Kotz 2000 §4.4 - §4.5
Delegation	Abadi et al. 1993 §5 - §6.1 Also: Lampson et al. 1992 §7, Howell and Kotz 2000 §4.6
Interprocess Communication	Lampson et al. 1992 §8
Access-Control Decisions	Abadi et al. 1993 §6.2 Also: Lampson et al. 1992 §9
Reasoning about Credentials and Certificates	Wobber et al. 1994 §1 - §4.3
Extensions to the Logic	Howell and Kotz 2000 §6

Table 5: CSE 774 Topics in Authentication, Delegation, and Access Control

Assessment in CSE 774

Summative assessment is accomplished through four open-book, open-notes exams. Students receive written solutions as they turn in their exams. The questions are formulated with the educational outcomes at the various levels of knowledge previously described (comprehension, application, analysis, synthesis, and evaluation). While the course emphasizes the use of formal analysis, more weight is typically given to questions that ask students to set up the formal assumptions, definitions, and goals when given an informal problem description.

Fundamentals

The first exam occurred four weeks into the course. Two questions dealt specifically with comprehending cryptographic algorithms (DES and RSA): given particular inputs, students had to compute the outputs. Little class time was devoted to these topics, and the average grades on these questions (out of eighteen students) were 71% and 82%. At the levels of application, analysis, and synthesis, students were asked to model cryptographic algorithms in schemes such as electronic code book and cipher block chaining. Here, the results were less satisfying. For example, students were asked to model ECB encryption as a recursive function (where the particular encryption function and key are parameters) and prove that ECB inverts itself. In this case, the average score was 50%. However, the distribution of grades was bimodal: 25% of the students received full or close to full credit, 25% received no credit, and the remaining students received between 20% and 70%, with most of these students getting more than 50%. All of the students who received no credit had failed to take the prerequisite course in predicate calculus.

BAN Logic and Role-Based Access Control

The BAN material was also assessed in the first exam, at the application, analysis, and synthesis levels. Specifically, one question at the analytical level asked for a proof that, in the context of the X.509 protocol, one principal believed that another principal believed in a particular statement. The initial assumptions about keys and nonces were given. The average for this question was 76%, with seven students getting 100% and one student getting 0%.

Another question asked students to consider how beliefs would change if X.509 timestamps could not be checked accurately, and to justify their answers using the BAN logic. The average for this question was 70%: ten students got 100%, and three students got 0%.
RBAC was assessed in the second exam, which was given after seven weeks. By this time student enrolment had dropped from eighteen to fifteen. RBAC was assessed at the levels of application, analysis, and synthesis. Students were shown an organizational chart of company roles and asked to formally prove or disprove mutual exclusivity of various roles. The average on this problem was 66%, but the distribution was again bimodal. Five students earned 100%, six students earned between 70% and 95%, one student received 20%, and the remaining three students received 0%.

Authentication, Delegation, and Access Control

The underlying semantics of the principal calculus (Lampson et al. 1992, Abadi et al. 1993, Howell and Kotz 2000) is based on Kripke structures. A Kripke structure comprises a set W of *possible worlds*; an interpretation function I, which maps each propositional variable to a subset of W; and an interpretation function J, which maps each principal name to a binary relation over W. Intuitively, $I(p)$ is the set of worlds in which the propositional variable p is true, and $J(A)$ is the accessibility relation for principal A: if (w,w') is in $J(A)$, then principal A cannot distinguish between worlds w and w'.
Questions on the second and third exam focused on assessing the students' understanding at the comprehension and analysis levels. Specifically, they were given a particular Kripke structure and asked to evaluate the beliefs of principals in various worlds. The average grade on Kripke structures in the second exam was 58%, with three students receiving 100% and four students receiving 25% or less. These grades improved on the third exam to an average of 65% with five students receiving 100%, one student receiving 93%, and the remaining students receiving between 27% and 53%.

Students were also asked on both the second and third exams to prove an axiom of the calculus (e.g., `if $(A \wedge B)$ *says* s, then A *says* s and B *says* s'). The average grade for this question on the second exam was 48%, with two students receiving 100% and three students receiving 0%. On the third exam, grades improved to an average of 60%, with five students receiving 100%, one student receiving 80%, one student receiving 0%, and the rest receiving between 20% and 60%.

The fourth and final exam (given to fourteen students after fourteen weeks of class) focused on reasoning about certificates, delegations, and authority and on proving properties of credentials used in the Taos operating system (Wobber et al. 1994). This time, students were presented with a client/server system where the server receives a message `C_{Alice} *says RQ*' within the context of boot, delegation, and channel certificates. The students first had to set up a theory whereby the server could conclude the statement *((Machine as OS) for Alice) says RQ*, and then they had to carry out a formal proof to justify their conclusions. This problem was very similar to the extended example of (Lampson et al. 1992). The average score on these questions was 67%, with seven students scoring 80% and above (two at 100%, one at 97%, and four at 80%), and the remaining seven students with scores from 33% to 60%.

Observations and Adjustments

Based on our observations at the course level, we have identified both pedagogical principles and suggested adjustments, at both the course and curricular levels. We outline these findings here.

Course-level Observations for CIS 632

We have found it useful to introduce the automated tools early in the semester: it helps students view the process algebra as more like a programming language (and therefore *real*) than just a mathematical notation. While grading homework early in the semester, we noticed that students in the undergraduate version of the course seemed to write higher quality CSP code: the graduate students wrote code that was technically correct, but it had a very strong imperative flavor (e.g., large numbers of nested conditionals). We postulate that, because the undergraduates all had functional-programming experience, they were better prepared to think about computational solutions in a non-imperative way.

The results from the final exam suggest that students have good intuition about the concepts but need more practice in justifying answers rigorously: for example, many students would give a specific example to justify that a general claim was true.

We have also found that students have difficulty grasping the distinction between the *properties* one wishes to prove about a system and the CSP idiom of imposing *constraints* upon a system through parallel composition. The properties are described as CSP processes, and one can check whether the system satisfies those properties by checking whether a refinement relationship exists. Constraints are also represented by CSP processes, placed in parallel with the system to *enforce* certain properties. The confusion seems to arise because the same language is used both to express properties and to describe the system design: students have difficulty maintaining that distinction.

Having originally identified this problem in Fall 2001, we made a concerted effort in Fall 2002 to mediate it. In the in-class bridge project, two groups were charged with identifying the desirable properties (e.g., *all cars on the bridge are going in the same direction*) and describing them in CSP; the remaining groups were to model the bridge and a traffic-light system in accordance with a proposed solution. We would then, as a class, combine both parts in a single file and check the necessary refinement relationships. The expectation was that, by separating the tasks, students would concentrate on either properties or the system and be able to see how they relate to each other. However, both groups charged with writing properties repeatedly asked for clarification of their task: they had trouble comprehending how their properties were related to the overall system.

Consequently, we plan to use the process algebra CCS in the next offering of this course. The CCS notion of synchronization differs from the CSP notion in a way that avoids the constraints idiom. Furthermore, the available automated tools for CCS—such as the Concurrency Workbench (Cleaveland et al. 1993)— support the use of temporal logic for describing desirable properties. We believe that the use of different notations will help students distinguish between system-level descriptions and their behavioral properties.

Course-level Observations for CSE 774

In the Fall 2002 offering, we were interested in seeing how far and how deeply our students could learn to use the principal calculus of Abadi and colleagues. We made mid-course corrections after observing that many students were not facile with the use of discrete mathematics (sets, relations, algebraic properties). As a result, we devoted more time to reinforcing discrete mathematical notions at the expense of reasoning about additional protocols, such as electronic banking protocols. In the future, we plan on rectifying the lack of facility with discrete math by incorporating more applications such as RBAC and modal logic into the predicate-calculus course that is a prerequisite for this course.

In our view, limiting the time spent on cryptographic algorithms so as to devote time to the authentication logic was well worth it: students gained knowledge that enables them to think rigorously about authentication, certificates, delegation, and access control. Furthermore, the first-exam results demonstrate that students are capable of picking up the cryptographic material on their own.

Curricular-level Adjustments

The assessment results from both courses indicate that our students need practice in working simultaneously at two levels of abstraction: (1) they must be facile with the rules of a given logical system, and (2) they must be able to link those rules to a particular instance or application. In general, our students have demonstrated good intuition about concepts, but they lack experience in translating intuition into precise and rigorous statements that are amenable to analysis and verification.

In addition to the course-level changes, we are making curricular adjustments in our Computer Science (CS) and Computer Engineering (CE) programs to address these needs. First, we are redesigning our prerequisite courses to provide additional experience in relating the abstract to the concrete, so as to minimize the amount of remedial work necessary in advanced classes such as CIS 632 and CSE 774. Because the same person teaches both CSE 774 and its prerequisite course CSE 607 (*Logical Basis for Computing*), we were able to initiate this adjustment in Spring 2003.

CSE 607 is a required course for all CE Master's students. A primary objective of the course is to enable CE students to use predicate calculus to reason about specifications and implementations and how they are related, much as they use Laplace transforms to reason about communications systems. This course has a reputation among students as being hard (we suspect that a major reason is a lack of experience in relating the abstract and concrete), and a large percentage of students wait until their final semester to take it.

In Spring 2003, we introduced Kripke structures in CSE 607 as one application of the predicate calculus, set theory, and relations. Because this course is required for all CE students, the enrolment (79 students) was significantly higher and more diverse than that of CSE 774. Despite these apparent challenges, the average score on the Kripke-structure exam question was 68%, a marked increase from the average grade on the first Kripke-structure exam question in CSE 774. However, the average on this question was still lower than the average on the rest of the exam, which required students to state and prove properties about concrete relations (average grade of 75%), prove properties about an algebraic system (83%), and translate English arguments into predicate calculus and give a formal proof of validity (87%).

. The second curricular adjustment is larger in scale and in time horizon. We are moving towards a common core in both Computer Science and Computer Engineering that emphasizes logic, concurrency, and functional programming. In our view, knowledge of these three areas provides a platform for developing new theories and the means to specify and verify particular systems and applications. Logic provides a means for reasoning, concurrency is the building block for complex systems, and functional programming serves as a mechanized animation of these ways of thinking

Conclusions

We have described two courses that illustrate our efforts to introduce rigorous foundations for assurance into an IA curriculum. Specifically, we have presented not only the material that we cover, but also how we assess students' mastery of that material and how they are doing. The statistics that we have provided are not by themselves statistically significant. However, we believe they shed light both on how we are doing so far and on what is possible in a Master's-level IA curriculum. Furthermore, students have responded positively to these courses: in fact, multiple students have applied to our Master's CASSA program as a direct result of having analyzed security protocols in the concurrency course.

Based on our experiences, we advocate using an outcomes-based approach to develop courses and curricula. We are not assessment experts, but rather users who have found value in adopting practices put forward by experts in the field of Higher Education. We have tried to demonstrate that the process does not have to be painful or cumbersome. Even a lightweight approach as in CIS 632 offers a lot of benefits. A course's educational outcomes are suggestive of the appropriate assessment techniques, and both outcomes and assessment are in the instructor's control. Another advantage of writing outcomes is that information is communicated more precisely, both to students and to other faculty. Because we have thought about educational outcomes, several faculty across several courses have developed consensus regarding our Computing Engineering and Computer Science Master's programs. Because we have assessed our outcomes, our course and curricular changes are more precisely aimed at addressing shortcomings. We are moving towards a common core for these two programs that includes logic, functional programming, and concurrency. This common core would help prepare students for the two courses discussed here, as well as for the rest of our Master's programs.

Acknowledgements

This work is partially supported by the National Science Foundation under Grant Number DUE-0241856, as well as by the Information Assurance program of the New York State Center for Advanced Technology in Computer Applications and Software Engineering (CASE).

References

Abadi, M., Burrows, M, Lampson, B., Plotkin, G (1993) A Calculus For Access Control In Distributed Systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706-734.

Burrows, M. Abadi, M., Needham, R (1990) *A Logic Of Authentication*. SRC Research Report 39, Systems Research Center, Digital Equipment Corporation, Palo Alto, CA.

Bloom, B.S. (1974) *The Taxonomy of Educational Objectives: Affective and Cognitive Domains*. David McKay, New York.

Cleaveland, R., Parrow, J., Steffen, B (1993) The Concurrency Workbench: A Semantic-Based Tool For The Verification Of Concurrent Systems. *ACM Transactions on Programming Languages and Systems*, **15**:36-72.

Diamond, R.M. (1998) *Designing & Assessing Courses & Curricula: A Practical Guide*. Jossey-Boss, revised edition.

Ferraiolo, D.F., Barkley, J.F., Kuhn, D.R. (1999) A Role-Based Access Control Model And Reference Implementation Within A Corporate Intranet. *ACM Transactions on Information and Systems Security*, **2**,1: 34-64.

Ferraiolo, D.F., Kuhn, D.R. (1992) Role Based Access Control. In *Proceedings of 15th Annual Conference on National Computer Security*, Gaithersburg, MD, pp. 554-563. National Institute of Standards and Technology.

Formal Systems (Europe) Ltd (1997) *Failures-Divergence Refinement: FDR2 User Manual*, Oxford.

Formal Systems (Europe) Ltd (1998) *Process Behaviour Explorer: ProBE User Manual*, Oxford.

Ferraiolo, D.F, Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R. (2000) *A Proposed Standard For Role-Based Access Control*. Technical report, National Institute of Standards and Technology.

Howell, R., Kotz, D. (2000) *A Formal Semantics For Spki.* Technical Report TR 2000-363, Dept. of Computer Science, Dartmouth College, Hanover, NH.

Hoare, C.A.R. (1985) *Communicating Sequential Processes*. Series in Computer Science. Prentice Hall, London.

Lampson, B., Abadi, M., Burrows, M., Wobber, E. (1992) Authentication In Distributed Systems: Theory And Practice. *ACM Transactions on Computer Systems*, **10**,4: 265-310.

Lampson, B. (1971) Protection. In *Proceedings of the 5th Princeton Conference on Information Sciences and Systems*, Princeton, NJ.

Lowe, G. (1996) Breaking And Fixing The Needham-Schroeder Public-Key Protocol Using FDR. *Software Concepts and Tools*, **17**:93-102.

R. Milner (1980) *A Calculus of Communicating Systems*, volume 92 of Lecture Notes in Computer Science. Springer-Verlag.

Older, S., Chin, S. (2002) Building A Rigorous Foundation For Assurance Into Information Assurance Education. In *Proceedings of 6th National Colloquium for Information Systems Security Education*, volume 1. George Washington University Journal of Information Security.

Roscoe, A.W. (1998) *The Theory and Practice of Concurrency*. Series in Computer Science. Prentice Hall, London.

Schneider, S. (2000) *Concurrent and Real-Time Systems: The CSP Approach*. John Wiley & Sons.

Saltzer, J., Schroeder, M (1975) The Protection Of Information In Computer Systems. *Proceedings of the IEEE*, **63**,9:1278-1308.

Stallings, W (1999) *Cryptography and Network Security*, Second Edition. Prentice-Hall.

Syracuse University Department of Electrical Engineering and Computer Science (1998). *Developing Curricula To Meet The Needs Of The Next Millenium: Preliminary Report Of The EECS Curriculum & Course Development Committee*.

Wobber, E., Abadi, M., Burrows, M., Lampson, B. (1994) Authentication In The Taos Operating System. *ACM Transactions on Computer Systems*, **12**,1: 3-32.