Expanding the Frontiers of Computer Science: Designing a Curriculum to Reflect a Diverse Field

Mehran Sahami
Computer Science Department
Stanford University
Stanford, CA 94305, USA
sahami@cs.stanford.edu

Alex Aiken
Computer Science Department
Stanford University
Stanford, CA 94305, USA
aiken@cs.stanford.edu

Julie Zelenski
Computer Science Department
Stanford University
Stanford, CA 94305, USA
zelenski@cs.stanford.edu

ABSTRACT
While the discipline of computing has evolved significantly in the past 30 years, Computer Science curricula have not as readily adapted to these changes. In response, we have recently completely redesigned the undergraduate CS curriculum at Stanford University, both modernizing the program as well as highlighting new directions in the field and its multi-disciplinary nature. As we explain in this paper, our restructured major features a streamlined core of foundation courses followed by a depth concentration in a track area as well as additional elective courses. Since its deployment this past year, the new program has proven to be very attractive to students, contributing to an increase of over 40% in the number of CS major declarations. We analyze feedback we received on the program from students, as well as commentary from industrial affiliates and other universities, providing further evidence of the promise this new curriculum holds.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer and Information Science Education – *Computer science education, Curriculum.*

General Terms
Design, Documentation, Experimentation, Management.

Keywords
Curriculum, Concentrations, Tracks, Multi-disciplinary.

1. INTRODUCTION AND GOALS
While the discipline of computing has evolved significantly in the past 30 years, Computer Science curricula have been much slower to adapt to these changes. In particular, CS has relatively recently spawned a number of sub-fields that have rapidly grown in both intellectual and practical importance but are under-represented in the typical CS undergraduate curriculum. Furthermore, CS is having a pervasive impact on a wide range of other disciplines, a phenomenon that is likely to continue for many years, but opportunities for undergraduates to study these advances at the boundaries between disciplines remain relatively rare. At Stanford University, we undertook to bring the Computer Science curriculum in line with what is happening in the field, completely redesigning the undergraduate CS major to explicitly show the diversity of topics in Computer Science and provide greater opportunities for inter-disciplinary work. Although not the primary motivation, recent declines in CS program enrollments [12] added impetus to our desire to capture student interest by increasing awareness of the great diversity and intellectual challenge that computing presents.

SIGCSE'10, March 10–13, 2010, Milwaukee, Wisconsin, USA.

At Stanford, between 2001 and 2006, we witnessed a drop in CS enrollments generally in-line with national trends, as the number of annual CS major declarations dropped from over 150 to approximately 80. While enrollment stabilized and even began to grow modestly in 2007, the strong correlation between student enrollments and the health of the high-tech economy cannot be denied [10]. More recently, the media portrayal of "a supposedly horrific loss of [computer programming] jobs" due to off-shoring [9] has deterred potential students from CS. While recent analysis has found that job off-shoring has not resulted in a net loss of IT jobs in the US [1], it is instructive to see that students' decision making is significantly affected by *perceptions* of job prospects. More subtly, students choosing to not major in CS because of their belief that programming jobs are being off-shored indicates that students are equating majoring in CS with being a programmer as the eventual (perhaps, only) career outcome. A similar phenomenon has been cited in other work looking at enrollment declines [7] and has also been used as the basis of curricular revision which stresses the importance of *context* for computing [4]. In a similar vein, we believe that explicitly broadening the "footprint" of Computer Science to show its many intellectually challenging subfields and strong multi-disciplinary ties can cast a wider net in piquing student interest to pursue CS. Thus, the primary goals of our curriculum revision are aimed at:

- providing students greater awareness of the breadth of options in CS and opportunities to pursue these areas in depth,
- incorporating relatively new, but already mature, sub-fields of CS on par with more traditional topics within the curriculum,
- highlighting and promoting multi-disciplinary connections,
- establishing a structure with sufficient flexibility to allow for lightweight revision in response to the evolution of the field.

These goals (especially the first three) are in-line with recommendations on engineering education from the National Academy of Engineering [8] as well as a recent NSF workshop on Integrative Computing Education and Research [5].

Interestingly, while increasing enrollments was not a direct goal of our curriculum revision efforts, we were happy to see a significant increase in CS major declarations in the year since our new program was launched. A detailed analysis of the reasons for this increase is presented in Section 4.

2. CURRICULAR STRUCTURE

Immediately prior to our new curriculum, our program structure consisted of a large core of required classes (CS1, CS2, advanced programming, discrete math, automata/computability, algorithms, AI, and computer architecture), a few restricted choices (operating systems or compilers), and roughly three restricted CS electives. In the past two decades, this curriculum saw only minor revisions in adjusting a few particular course requirements—the essential structure remained untouched. During that same period, new frontiers had emerged in CS and multi-disciplinary ties to other fields had been established, but the large and inflexible major requirements limited the ability of students to take advantage of courses in those areas.

In alignment with our goals, we pursued a *track-based* model for our new curriculum, with the following general structure:

- *Core*: a common set of courses all CS majors are required to take to establish a shared intellectual foundation.
- *Track*: a chosen topical concentration area (selected from a menu), aimed at giving students greater depth in the area(s) of CS they find the most interesting.
- *Electives*: additional course options allowing students to pursue greater breadth, depth, multi-disciplinary ties, or combinations thereof.
- *Senior Project*: a capstone course with either a software development or research emphasis.

Being homed in the School of Engineering, general requirements such as mathematics and science still exist in the new curriculum, but we discuss them only in relation to the requirements above. We describe the Core, Tracks, and Electives presently.

2.1 The Core

The Core curriculum focuses on providing a common intellectual experience in CS, setting a foundation for use across a variety of depth areas. This raises the opportunity—indeed, the necessity—to have the Core include only essential elements common to the many directions in which CS has evolved. Given our program's historical context, agreeing to have a Core was not contentious. Rather, the challenge was to keep the Core as streamlined as possible. We gathered input from a number of faculty detailing their "must-haves" for any computer science student and modeled the Core based on the *intersection* of those lists rather than the *union*. Our Core is composed of six courses: three in "Theory" and three in "Systems", detailed below.

> *Systems I*: *Programming Abstractions*—a classic CS2 course emphasizing common data abstractions and structures as well as recursion.

While this is a CS2 course, it does not have our CS1 as a strict prerequisite (our CS1 course is in Java, whereas CS2 is in C++, lessening the mechanical dependency between the two courses). An analysis revealed that roughly half of our CS majors did not take CS1 (due to AP credit or prior programming experience), providing evidence that a reasonable number of students can begin the Systems Core immediately. Of course, we do not expect all students to do so, and offer an accessible CS1 course taken by a large percentage of the entire undergraduate student population that is meant to "funnel" students into the CS program. Our CS1 and CS2 courses were unchanged as a result of the curriculum revision. We note that the selection of Core topics is somewhat orthogonal to the detailed contents (e.g., programming language) of CS1, making such a structure more readily adoptable by others.

> *Systems II: Computer Organization and Systems*—a course with the theme of taking students "from the hardware up to the source code". Topics include machine architecture, memory models, data representation, and elements of compilation and concurrency.

Systems II is meant to provide students with a unified introduction to the lower levels of the machine. We feel that it is essential for students in any subfield of computing to understand the fundamental abilities and limitations of real computing systems. Even students with a theoretical focus benefit from seeing such material as applications of automata theory or program analysis/verification. This course is not a replacement for a full course in either compilers or machine architecture, but since it contains the foundational material we believe all students need in this area, it allows for streamlining the curriculum to no longer require a full course in either of these topics.

> *Systems III: Principles of Computer Systems*—the theme of this course is "modern computer systems and networks". Topics include processes and concurrency mechanics, file systems, virtualization, networking and distributed systems.

This course explains the facilities and paradigms that modern operating systems, networks and distributed systems provide. The need for this course is reflected in the fact that computing has and will continue to move toward a much more distributed paradigm (both multicore chips as well as large scale data centers and the web), and all students must be facile with such workings. Across such diverse areas such as HCI (where practitioners consider how distributed systems affect end-user interaction through issues such as latency or information consistency) or Computational Biology and Artificial Intelligence (where researchers develop algorithms for parallel, distributed systems), the direction in which computing is moving in the 21st century makes such a class important to all Computer Scientists.

Again, we stress that this course is not a replacement for a course focused on implementing an operating system or designing and implementing network protocols. The course is meant to be a foundation relevant to all students—it will be a reasonable endpoint for some, while providing a stepping stone for those who choose to pursue further work in systems design and implementation. As a result, we eliminated the requirement for all students to take a full course on operating system implementation.

> *Theory I*: *Mathematical Foundations of Computing*—this is a course on discrete math and automata. Topics include logic, induction, proof techniques, sets, functions, relations, and an introduction to automata and NP-completeness.

The goal of Theory I is to give students the mathematical language of computing, the ability to present cogent arguments

(i.e., proofs), and an understanding of what is possible to compute. The material is motivated *in situ* through real world applications, while also presenting topics, such as finite automata, which will be built upon in Systems II and III, reflecting the importance of theory to systems and vice versa.

> *Theory II*: *Probability Theory for Computer Scientists*—an introductory course meant to provide students with tools for probabilistic analysis and modeling in computing. The course also provides an introduction to Machine Learning.

Theory II is perhaps the most significant departure from a "standard" CS curriculum [6], but it is a reflection of where we believe the field is headed. Indeed, probabilistic analysis has become widespread as a tool in analyzing systems, constructing algorithms, and modeling uncertainty in user interactions. It forms the basis for many well known applications including Google's PageRank algorithm, modern email spam filters, and the analysis of biological and social networks. While many CS curricula (including our previous program) require a course in Probability, such courses are generally not taught in CS departments and are much more focused on theoretical results and proofs. They are generally devoid of any discussion of the real use of these models in computing and provide no examples of real-world software systems based on their use. For these reasons (among others), we chose to develop our own probability class in-house, allowing us to specifically target the use of probability in computing. Moreover, we also use the course as a vehicle to teach elements of Machine Learning and AI, showing applications both within computing as well as their use in other disciplines. Such applications can help students better understand the potential impact and social context of their work.

> *Theory III*: *Data Structures and Algorithms*—an advanced course in the design and analysis of algorithms, presenting various data models (e.g., algorithms for trees and graphs) and a range of algorithmic techniques (e.g. greedy and randomized algorithms, dynamic programming).

The focus of Theory III is to provide the algorithmic tools that we believe a facile computer scientist needs to know to think *computationally* in a variety of contexts. Such a course is common in many computing curricula, and we are in agreement with many of our peer institutions that this material is foundational and necessary for all computer scientists.

The prerequisite structure for the Core courses is shown in Fig. 1. We include our current CS1 (and potential future alternatives) in the diagram to show the variety of entry points into the Core.

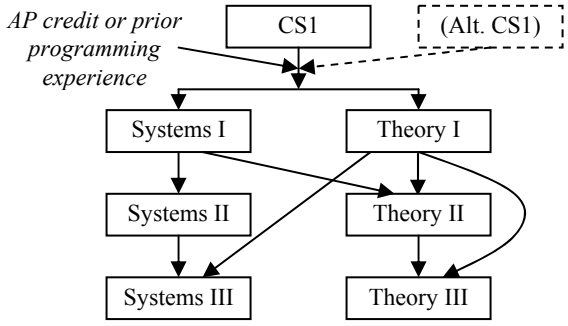

Figure 1. Prerequisite structure for Core courses.

Since our regular academic year is composed of three quarters, it is possible for students to complete the Core material in a single year, but we expect most students to space this material out during the course of their Freshman and Sophomore years.

While others have argued that a common core in a computing curriculum can be avoided [4], we have found that providing such a core offers several advantages. First, it provides a general foundation that students can use to pursue many areas in CS *after* they graduate. Indeed, the field of computing grows so quickly that it is simply not realistic to expect the education students receive as undergraduates to be sufficient for their whole careers, or that their interests will even remain in one subfield of the discipline. Thus, the Core aims to provide a foundation on which continued work in a variety of areas within CS may be pursued. Equally importantly, however, is that the Core provides students with a common *social* experience—the "bonding" that takes place as students undertake common challenges and have shared experiences. But perhaps most importantly, having a designated Core allows students to defer the selection of a concentration area (i.e., track) in CS until they have more exposure to the field. This is especially important given the findings that most students in high school do not know what CS is about [2]. The idea of a Core has been well-received by our students. As we show later in this paper, a recent survey of our CS undergraduates found 51% of respondents indicated that a streamlined set of core courses was one of the most appealing aspects of the new curriculum, while only 3% viewed it negatively.

2.2 Tracks

All CS students are required to complete the requirements for one track in our program. This provides the opportunity for students to better align their program with their interests in the field, and also pursue multi-disciplinary work related to that area. In its initial roll-out, our curriculum offers nine track choices:

- Artificial Intelligence
- Theoretical Computer Science
- Systems
- Human-Computer Interaction
- Graphics
- Information (management and application of data)
- Biocomputation
- Unspecialized
- Individually Designed

A track is generally composed of 4-5 courses, with requirements structured as follows:

a.) one or two designated *gateway* courses
b.) selection of one or two courses from a menu of courses highly related to the track area
c.) selection of additional courses from (b) and/or from a list of more broadly related courses to the track area

For example the Artificial Intelligence track looks as follows (shown with a much abbreviated sample of course offerings):

a.) Required: *Principles of Artificial Intelligence*
b.) Two courses from: *Intro. Robotics*, *Computer Vision*, *Agent Systems*, *Natural Language Processing*, *Machine Learning*
c.) One additional course from (b) or from: *Speech Synthesis*, *Advanced Robotics*, *Computation Genomics*, *Web Search*, *Decision Analysis*, *Stochastic Control*, *Information Theory*, *Modern Applied Statistics*

The designated "gateway" course(s) are the foundational course(s) in the track area (for example, "Principles of Artificial Intelligence" in AI or "Introduction to Human-Computer Interaction" in HCI). These courses provide a more in-depth introduction to the area, assuming students have already taken some portion (but not necessarily all) of the Core courses. The gateway courses serve two purposes. First, they provide a clear mechanism for students to sample a particular track area to decide if they wish to pursue it further (and such sampling can be done prior to the completion of the entire core, so students can generally take gateway courses as early as their Sophomore year). Second, all gateway courses are also designated as general CS *electives* (described in more detail later). This allows students who sample a gateway course and decide not to pursue that respective track to incur no programmatic penalty as the gateway course can simply be counted as one of their in-major electives.

We make special note of the last three tracks listed above, as their requirements are somewhat different from the other six. The *Biocomputation* track requires additional science courses (specifically, a year of Chemistry and two quarters of Biology) beyond the standard CS requirements (in Physics). To offset these additional requirements, students take fewer courses in other categories. Importantly, the track contains many requirements for a collegiate pre-medical program, allowing pre-med students an option to major in CS without significant undue burden.

The *Unspecialized* track is essentially composed of each of the gateway courses of the other tracks, with a few additional breadth options. This track serves two purposes. First, it provides a "breadth" track option for students who choose not to pursue one area in particular depth. Indeed, students who sample several gateway courses and cannot decide on a particular track can simply graduate under the Unspecialized track as they will have satisfied its requirements by taking five gateway courses from other tracks. Second, the Unspecialized track mimics the requirements of our previous curriculum. In this way, the new curriculum is largely a superset of our previous program requirements, allowing in-flight students to graduate under the new requirements without significant adjustment.

The *Individually Designed* "track" is simply a set of guidelines allowing students to propose their own coherent set of track requirements (requiring faculty approval). It provides an added level of flexibility to address individual needs or fast-emerging areas without the need for university-level program approval.

More generally, the designation of tracks allows students to explicitly see the diversity of options available to undergraduates in CS. Since our CS program no longer has one monolithic set of requirements, but rather explicitly lists a number of subfields, the program itself serves as an advertisement for the diversity of the field. Furthermore, by allowing students to pursue greater depth in the area of CS they find most appealing, we increase the frontier of the field that is accessible to undergraduates. This makes the overall "footprint" of our curriculum larger and casts a wider net to attract student interest.

Finally, an explicit goal of our curriculum redesign was to make future updates to the curriculum easier. Modularizing our program into the track-based structure accomplishes this goal, as it is now possible to add or remove tracks without affecting other tracks or the set of Core courses.

2.3 Electives

Beyond the Core and Track requirements, students complete their CS depth requirements by selecting from a set of elective courses. Students are required to take a minimum of seven courses, including their track requirements and electives. The electives come from two sources. First, a set of *General CS Electives* is available to choose from for students pursuing *any* track area. This list includes most of the undergraduate courses in the CS department. Each track then designates a set of *Track-specific Electives*—additional courses that are available as electives for students pursuing that track area. The track-specific electives are intentionally selected to include appropriate multi-disciplinary courses (including such diverse areas as economics, engineering management, linguistics, statistics, studio art, psychology, and philosophy, as well as other engineering disciplines) as well as advanced graduate courses in the track area. For example, the *Graphics* track provides electives such as *Design* and *Photography* from the Art department, *Introduction to Perception* from Psychology, and *Game Studies* from the program in Science, Technology and Society. Thus, the program can serve students interested in computational media, perceptual issues in digital art, and/or game development all within the same track.

Electives allow students to pursue additional breadth (by taking general CS elective courses), further depth (by pursing graduate courses), multi-disciplinary ties (by taking appropriate non-CS courses), or some combination thereof. This flexibility helps address a variety of student interests/career paths as well as highlights the growing multi-disciplinary nature of CS. Indeed, if we were to use AP Exams as a leading indicator of student interest, we would find that while CS numbers are stagnant, there are growing numbers of students interested in areas related to CS (e.g., Biology and Statistics, which had over 7 times and 5 times as many exam takers as CS, respectively, in 2008 [3]) who could be well served by a CS program that combines computing with other areas of interest (e.g., Biocomputation for Biology students, AI/Machine Learning for Statistics students, etc.).

3. PROCESS

The need for substantial curricular reinvigoration was widely recognized in our department (composed of roughly 50 faculty members), and our curriculum revision process was driven, unsurprisingly, by an 11-member departmental Curriculum Committee. Initially, we focused on determining a high-level structure for the curriculum, reaching consensus on the model of a Core, Tracks, and Electives before filling in the details. This provided a general framework—which was at times revisited—to determine the detailed program requirements. Importantly, this concrete initial proposal was put before the full faculty before moving on to the detailed work. Gaining unanimous departmental faculty support provided early buy-in that a curricular change was needed and gave the committee a mandate for significant reform.

The next task was defining the Core courses and their content. Two factors made this often contentious process successful. First, rather than having open-ended discussions about "what every computer scientist should know," we always maintained a concrete working document defining the latest revision of the Core courses and their detailed contents. This allowed for much more directed committee discussions and detailed analysis of the trade-offs in including or removing certain topics. More importantly, however, at least one of the likely instructors for each of the new Core courses was

intimately involved in the discussion of the course topics. As a result, there was a realistic assessment of the amount of material that could be covered in each course as well as a potentially willing instructor for it—a critical factor for eventually translating curriculum documents into real courses.

Once the Core was defined, additional committees were formed to develop each track (and track-specific electives) separately. The track committees were composed of a subset of members of the Curriculum Committee as well as additional faculty in the track area (in some cases including faculty from other departments). Again, a concrete proposal for the requirements for each track was always maintained to drive the discussion. The critical aspect of this process was that a large portion (well over half) of the entire CS faculty became involved in the curriculum development process. This cemented wide support for the new curriculum. When the requirements for each track were finalized, the result was reviewed by the Curriculum Committee for coherence and level-setting across all tracks. When the entire curriculum was defined, it was unanimously adopted by the CS department faculty and then approved by our School of Engineering. The curriculum was rolled-out in its entirety during the 2008-09 academic year.

4. ASSESSMENT AND CONCLUSIONS

To assess our new curriculum, we examined enrollment statistics in CS, surveyed student attitudes toward the program and actively sought feedback from industrial affiliates. Most notably, after the deployment of the new curriculum this past year, we witnessed a surge of over 40% in undergraduate CS major declarations (from 87 in 2007-08 to 123 in 2008-09). While CS enrollment increases have been reported recently at other universities, the magnitude of our increase is well above the numbers we have seen from peer institutions, and survey data suggests our new curriculum is the cause. A voluntary on-line survey (primarily multiple choice questions) of our students who declared CS as their major this past year (N = 58) shows that over 36% of respondents indicated that the new CS curriculum influenced their choice of major, with 7% stating explicitly that they would have majored in something other than CS if not for the new curriculum. Statistical extrapolation from survey respondents (47% of the population) to the entire population of students declaring CS in the past year reveals that between 9 to 44 *incremental* students declared CS as their major as a result of the curricular changes. Given that the total incremental increase in CS declarations we saw in 2008-09 was 36 students, it appears that a solid majority of the incremental increase is due to the revised curriculum. We also assessed the aspects of the new curriculum students (N = 96) found most/least appealing, shown in Fig. 2. Note that students could select multiple options so the percentages across categories do not sum to 100%. With regard to *most* appealing aspects, students could choose "Flexibility in requirements", whereas the analogous question for *least* appealing aspects was "New required courses" they would have to take. As can be seen in Fig. 2, students overwhelmingly found the new curriculum to be appealing, with the ability to focus on a particular track being the most compelling aspect of the program (77% positive response).

We also reviewed the curricular changes with many members our of industrial affiliates program and received uniformly positive responses from them. Discussion of our new curriculum has been

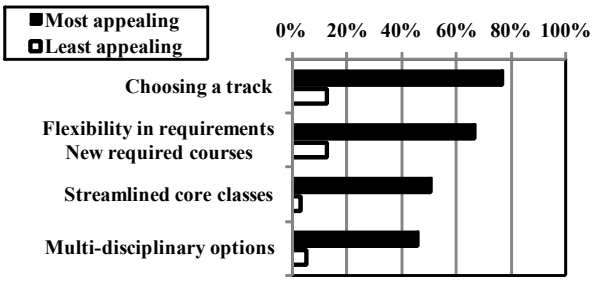

Figure 2. Aspects of curriculum most/least appealing (N=96).

taken up in several leading CS departments both nationally and internationally, giving us useful feedback on the program, including invitations to help two other universities restructure their CS curricula. While most feedback we have received has been positive, we have also received one important criticism: the lack of functional programming in our Core courses [11]. While our prior curriculum also lacked much functional programming, we now plan to use the functional statistical language R in a lab adjunct to our Theory Core II course to help address this point.

In conclusion, we believe our new curriculum helps capture student interest by explicitly reflecting the broadening frontiers of the discipline and allowing students to better align their academic programs with their interests. We would encourage others to consider a similar model, playing to strengths of their individual departments. For small departments, we recognize that having an extensive list of tracks may not be possible and we would encourage partnering with non-CS departments to provide more multi-disciplinary tracks as a means of highlighting the increasingly broad influence of computing in other areas. Smaller CS departments also have the potential to attract more students by targeting the larger departments at their institutions to work with in creating multi-disciplinary tracks. Indeed, in discussions with small CS departments, we have seen such opportunities arise, for example, in partnering with media studies programs.

5. ACKNOWLEDGMENTS

We are grateful to members of our Curriculum Committee and the many other faculty involved in our curriculum revision process.

6. REFERENCES

[1] Aspray, W., Mayadas, F., and Vardi, M. Y. (Eds). 2006. Globalization and Offshoring of Software: A Report of the ACM Job Migration Task Force.

[2] Carter, L. 2006. Why Students with an Apparent Aptitude for Computer Science Don't Choose to Major in Computer Science. In *Proc. of SIGCSE '06*.

[3] College Board. 2008. AP Program Size and Increments 1984–2008.

[4] Furst, M., Isbell, C., and Guzdial, M. 2007. Threads™: How to Restructure a Computer Science Curriculum for a Flat World. In *Proc. of SIGCSE '07*.

[5] Integrative Computing Education and Research (ICER). 2006. Final Report of the Northwest Regional Meeting.

[6] Joint Task Force on Computing Curricula. 2001. ACM/IEEE Computing Curricula 2001 Final Report. http://www.acm.org/sigcse/cc2001.

[7] Morris, J. H. and Lee, P. 2004. The Incredibly Shrinking Pipeline is Not Just for Women Anymore. *Computing Research News*, 16(3), May 2004.

[8] National Academy of Engineering. 2005. Educating the Engineer of 2020: Adapting Engineering Education to the New Century. Nat'l Academies Press.

[9] Reynolds, A. 2004. Offshoring Which Jobs? *Washington Times*, June 6, 2004.

[10] Sahami, M. 2007. The Google Education Summit. http://research.google.com/university/relations/eduSummit2007/MehranSahami.pdf

[11] SIGPLAN Proposal on Functional Programming. 2008. http://wiki.acm.org/cs2001/index.php?title=SIGPLAN_Proposal

[12] Vegso, J. 2008. Enrollments and Degree Production at US CS Departments Drop Further in 2006-07. *Computing Research News*, 20(2), March 2008.