[**Motivation**] CS is undergoing a change in its fundamental identity. Yes, there will always be a need for software engineers, and the field will always enthusiastically prepare students who choose that path. Yet with each year, software engineering recedes in relative importance for CS.

Not replacing but eclipsing the traditional "CS for Software" mindset are the opportunities of "CS for Insight," i.e., computing with the purpose of providing insight to other endeavors -- specifically, insight that would be impossible or difficult to obtain without the tools of modern machines, their interfaces, and their libraries. These are insights answering questions that software engineers would not ask -- because they are not software-engineering - or even CS - questions.

[**Student learning objectives**] Thus, this course would build upon CS5's mindset and skillset in order that students emerge feeling justifiably and demonstrably confident that they can tackle the following challenges:

(1) extracting the computational essence of non-CS problems (or correctly declaring that a problem does not have a computational component)

(2) proactively assembling -- from existing libraries, tools, and datasets, atop CS5's foundational skills -- a software system which sheds light on a problem beyond CS itself; this student-learning objective would include each student's creation of an online portfolio of their work in the class

(3) analyzing the output and design of such computing systems in light of (a) its algorithmic and data-structure efficiency, (b) its human-development and -maintenance efficiency, and (c) its ability to provide insight for the problem(s) at hand - and suggest productive new directions for research

(4) practice in presenting the rationale, design, and results of the three skills above

[**Possible audiences**] One audience excited about this course are CMC's economists. They believe that, in order to claim to be a "top-10%" economist in this day and age, a student needs the computing skills listed in (1)-(4). In contrast, such students do ***not*** need the computing skills in CS60 -- even if they acknowledge that those skills are important for a solid computer-science or software-engineering foundation. (As an aside: Eric Helland and Manfred Keil are proposing to CMC a computational track that will require CS5 or equivalent, so that "prerequisite" is not a problem either for their students or, of course, ours.)

This proposal, however, is decidedly **not** to build a "computing for economists" course; rather, it is a course that builds computing skillsets and mindsets for *all* students wanting to leverage computing's resources for deeper insights into a their own (non-CS) field of interest.

It's not clear whether CS or other HMC majors would be interested in having their students take a course such as CS41, but this course takes a student-focused, not program-focused tack. I would be excited, for example, if an undeclared student interested in X or a student majoring in X were to take CS41 and, with that experience, develop a nontrivial software system that provided insights (pedagogical or otherwise) into X. All the better if it overlapped another course in its content, purpose, and efforts (certainly taking care to count things fairly). I imagine CS41 could work for many possible X's, since the computational components are orthogonal to the problems themselves. In addition, those problems could be drawn from non-academic interests, should students choose to do so.

I would also be interested in organizing the course so that students could choose to take it for different numbers of units, e.g., a 1.5-unit or 3-unit course. This would allow students (or student pairs) to tailor their investment as their priorities allow.

[**Work to be undertaken**] Attached as an appendix is a more thorough first-draft description of the course's week-to-week student activities, topics, and organization, in the text format requested by the Curriculum Committee.

The work involved in developing this course would include fleshing out that week-to-week set of ideas into

- a set of presentation materials to use in the first offering of the course
- a set of weekly assignments to use in the first offering of the course, with versions for 1.5- and 3-unit varieties
- completed examples of those assignments, for reference and feedback
- an example online portfolio that would serve as a template for the online portfolios students would create as their final deliverable
- a set of alum contacts – and possible visiting dates: these alums would serve as guest speakers during the open-ended project phase of the course, telling their stories of how they have leveraged computational tools in their (non-software-engineering) paths after HMC

[**Possible details...**]

Beyond the above rationales, possible distinctive features could be

- a sampling of topics from CS60 and CS70, but from a CS-for-insight point of view
  - many topics NOT in 60 or 70 are covered too, as the outline below notes
- two small projects leading up to a large, final project
  - one will ask students to create a GUI for visualization and control
  - one will be an automatically-assessed machine learning challenge, e.g., which ones of these 10,000 images contain a can of SPAM (or, for a less-CS-idiosyncratic reference, contain a person or one or more trees...)
- the final 3-4 weeks will (strive to) feature at least one talk per week by a recent alum
  - this will give students both the chance to make connections and envision diverse post-HMC trajectories

- there will also be a final project of at least four weeks of each student (or student-team's) choice of topic
  - this can overlap (in appropriate ways) with other courses, fields of study, or interests

Here is a timeline sketch:

Week 1:
[Lectures] From software to insight: CS as a resource for exploration
[Lab and assignment]
- command-line skills, project-documentation skills (we'll use online progress logs throughout)
- libraries and their maintenance
- APIs, SDKs, and IDEs (big-O(my!))

Week 2:
[Lectures] What computers do best: lookup! Hashtables, et al.
[Lab and assignment]
- measure different hashtables' performance, big-O, part 1
- memoization as a natural application of hash tables
- dynamic programming as an extension of memoization + lookup tables

Week 3:
[Lectures] What humans do best: network! Graphs, et al.
[Lab and assignment]
- implement and use BFS, DFS, shortest-paths and max-flow, big-O, part 2
- use graphs as models for phenomena
- pull data from personal social graphs on which to run graph metrics
- more dynamic programming (FW, min-change)

Week 4:
[Lectures] The only bandwidth that matters: visualization bandwidth via libraries
[Lab and assignment]
- explore several data-visualization libraries
- practice the skill of "getting into a new library"
- review object-oriented programming paradigm and syntax, part 1
- choose small project #1 [from a flexible list of options]

Week 5:
[Lectures] The only bandwidth that matters, part 2: control bandwidth via GUIs
[Lab and assignment]
- build several small GUI applications (from a single library)
- again, practice the skill of "getting into a new library"
- review object-oriented programming paradigm and syntax, part 2
- continue to work on small project #1 - including at least a small GUI component

Week 6:
[Lectures] Presenting insights from computing: strategies and systems
[Lab and assignment]

- student teams' short presentations of small project #1
- formal but not burdensome feedback from other teams on the presentations
- finish work on small project #1 - submit project

Week 7:
[Lectures] Computing like humans? Machine learning, part 1
[Lab and assignment]
- try several different classifiers and compare performance
- use hashtables as classifiers
- bigger picture: the data is one model of itself (but not a very good one!)

Week 8:
[Lectures] Modeling and Machine learning, part 2
[Lab and assignment]
- explore and extend the repertoire of algorithms: SVN, linear/logistic regression
- connect with graphs + lookup tables: random forests et al.
- start small project #2 - machine learning challenge [informal intra-class challenge]

Week 9:
[Lectures] Web interfaces for input and output
[Lab and assignment]
- accessing web data
- creating applications with a web interface (using specified template and layout libraries)
- continue machine learning challenge (small project #2) - must include web components

Week 10:
[Lectures] Practical theoretical computation: regular expressions and parsing
[Lab and assignment]
- DFA/NFA/RE equivalence
- using RE libraries as a tool for modeling textual/syntactic data
- a glimpse at other parsing opportunities
- finish the machine learning challenge (automated results; not formal presentations)

Weeks 11-14: Final projects

Week 11:
[Lectures] Code maintenance and testing practices in industry w/ guest lecture(s)
[Lab and assignment]
- propose project
- demostrate access to the libraries that will be used
- start a source-code repository
- show an initial task breakdown and schedule

Week 12:
[Lectures] The softer and software sides of project manangement w/ guest lecture(s)
[Lab and assignment]
- continue work on final project
- weekly update in progress logs
- invite recent alums back from industry to speak about project management in practice

Week 13:
[Lectures] Guest lecture(s): computing for insight, not just an academic mirage!
[Lab and assignment]
- continue work on final project
- weekly update in progress logs
- invite recent alums back from industry to speak about their experiences

Week 14:
[Lectures] Student presentations of the final project
[Lab and assignment]
- presentations with formal, but not burdensome feedback from peers and, if possible, alums
- final deliverable

This may or may not be during finals week (there would be no pencil-and-paper final), depending on what the correct thing to do is, in light of the various forces at work (Dean Jacobsen's office, et al.)