Using Logic in Software Construction

Three courses developed by Rex Page that use logic in a hardware/software development context. All three are required of all Computer Science majors at the University of Oklahoma:
- Applied Logic for Hardware and Software (sophomore level)
- Software Engineering I (senior level, first semester)
- Software Engineering II (senior level, second semester)

SE-I and SE-II
The software engineering courses have three primary themes:
- software engineering processes
- design
- quality

The SE processes theme is covered in a standard way, using Humphrey’s approach (A Discipline for Software Engineering, Addison-Wesley). Students generally don’t like following the process because it requires keeping track of a lot of data (time logs, defect logs, software size and implementation time estimates, estimate-vs-actual correlation statistics, etc), but nearly all of them succeed with this part of the material.

The design theme is influenced by functional programming, starting with charts showing basic units and flow of data into and out of the units. Implementation is done in the purely functional ACL2 subset of Common Lisp. Nearly all students succeed in this effort. Some complain, but not as many as I expected when I started developing this material five years ago.

The quality theme focuses on low defect rates. Properties are stated as formulas in logic (using defthm in ACL2). Students develop test suites based on these properties, and push ACL2 through mechanical proofs of a few of them. Most of them succeed in stating proper theorems. Over half succeed with at least some proofs. About a quarter of them succeed with most of the proofs. The assignment describes the theorems they are to prove informally, and the required theorems are rigged to that if the students state them correctly, and the function definitions are not too arcane, ACL2 will prove the theorems without lemmas or hints in the early assignments, and with only a little work with lemmas or hints in the later assignments. (The Kaufmann/Manolios/Moore book, Computer Aided Reasoning: An Approach, is the required reading on this part of the material, and is also helpful for the design material.)

SE-I is about two-thirds individual work and one-third teamwork. SE-II is about one-third individual work and two-thirds teamwork and revolves around a software project of modest size (4,000 to 6,000 lines of code) with a dozen separate deliverables, spaced out across the semester, not quite uniformly.

Applied Logic for Hardware and Software
The applied logic course focuses on using logic to state and prove properties of digital circuits (mostly combinational circuits, including, near the end, a ripple carry adder) and of software expressed in the form of inductive definitions (that is, functional programs for algorithms ranging from list concatenation to sorting to AVL trees). A mechanical proof checker is used in the basic introduction to propositional logic (not ACL2) but is abandoned once we get to predicates (except where I show them a few ACL2 examples in lectures, to great affect, since the students have been struggling to do proofs by hand, and ACL2 just rips through them).

Students find this course very difficult because they have to do proof after proof after proof on homework and exams, but the success rate is high, and the material seems to help them in two follow-on courses: computer organization and data structures. I have statistical evidence for my claim that applied logic helps students succeed in data structures, but only anecdotes from individual students and professors about the connection with computer organization.

It is safe to say that by the time they complete Applied Logic, SE-I, and SE-II, most students have a permanently altered view of software correctness.