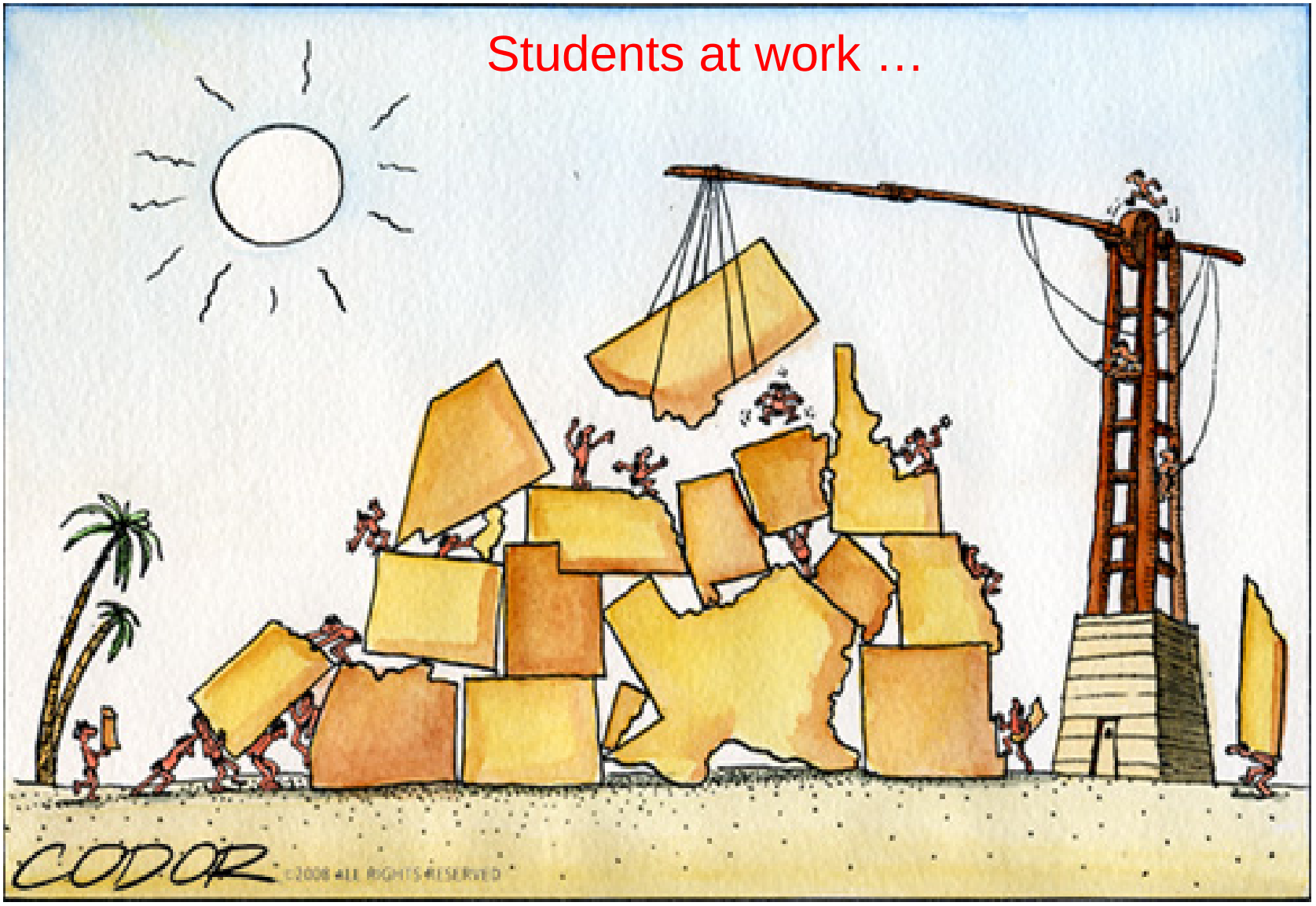
Colorado State University
Computer Science Department

Teaching Programming Students how to Model: Challenges & Opportunities

Robert B. France
Dept. of Computer Science
Colorado State University
USA
france@cs.colostate.edu

Students at work …



Student programmer perception of software modeling

- Modeling adds (accidental) complexity to software development
  - Tools are heavyweight: Learning and using documentation and analysis tools difficult
  - Difficult to determine model “goodness”
- Student concerns
  - How can development of modeling skills make programming easier?
  - Will modeling skills make me more marketable?

A perspective on student modeling problems

- Student models often reflect struggle with identifying and using abstractions
  - Problems may be a reflection of under-developed programming skills
    - They may be familiar with programming language syntax but still have weak programming skills
- Some students use models as an excuse for thinking informally about problems or solutions
  - Difficult for them to determine what details to include in a model

Teaching challenges

- Motivating software modeling
  - Why model software?
  - How can modeling enhance programming skills?
  - Will modeling skills enhance marketability?
- Understanding modeling aptitude
  - What makes Jane a “good” modeler and Johnny a “bad” modeler?
- Developing problem-solving skills
  - When and where in the curriculum should software modeling skills be developed?
  - How do you teach students to identify and leverage the “right” abstractions?
    - Do we understand the abstraction process well enough to teach it?

MOTIVATING SOFTWARE MODELING

The earlier you start to code the longer it takes to complete the program

Modeling as a software engineering enabler

- SE is “the discipline of resolving problems with software solutions”. (B. Blum).
  - Focus is on solving complex problems.
- Problem solving involves analysis and synthesis of solutions.
  - Analysis (Decomposition): What aspects and perspectives should be considered? What abstractions best capture relevant information in different aspects?
  - Synthesis (Composition): Use of compositional design construction and analysis techniques
- Model techniques provide the means to describe and analyze different aspects of a design
  - Modeling is an essential part of software engineering

Using models to bridge abstraction gaps

- Bridging wide abstraction gaps using manual techniques introduces significant accidental complexities
- Modeling techniques can be used to
  - reduce accidental complexity through use of abstractions that can be systematically realized, and
  - manage essential complexity through use of compositional analysis and construction mechanisms

In a nutshell …

Use of modeling techniques distinguishes a software engineer from a software developer (or programmer)

MODELING APTITUDE

Why do programming students find modeling difficult?

- Tools
  - Many existing modeling tools do introduce significant accidental complexity
  - Dissatisfaction with current toolset is sometimes the basis for dismissing modeling techniques
- Poorly developed abstraction skills
  - Significant effort invested on learning how "think" in terms of a programming language
- We know that
  - learning a modeling language is not enough;
  - students need to develop ability to identify the "right" abstractions

Finding the right abstractions

- Modeling must be purpose-driven

- Choose abstractions that support intended use of models
  - Using models to explore a problem or solution
  - Using models to analyze properties
  - Using models to generate implementations

Why Johnny can't model and Jane can

- Hypothesis: A good modeler is a good programmer; a good programmer is not always a good modeler
- Modeling requires programming and abstraction skills
  - Abstraction skills amplify development skills
    - programs produced by programmers with good abstraction skills should be of significantly better quality

DEVELOPING PROBLEM SOLVING SKILLS

Learning a programming language is easy, learning how to program is difficult

Problems students face

- How do we decompose a problem or solution?
- What information should be in a model and at what level of abstraction should it be expressed?
- How can we determine if the abstractions we use are “fit-for-purpose”?
- How can we determine if our model is of “good” quality?

"Traditional" approach to teaching modeling techniques

- Introducing modeling concepts using a 'waterfall' approach
  - Requirements modeling
  - Architecture modeling
  - Detailed design modeling
- Top-down approach reinforced by popular modeling textbooks
- Top-down modeling approach can overwhelm students whose previous experience base consists solely of developing small programs with fully specified requirements

An alternative bottom-up approach

- From modeling-in-the-small to modeling-in-the-large
  - Modeling-in-the-small: Focus on use of models to describe program designs
    - Bridging small abstraction gaps
  - Modeling-in-the-large: Extend focus to use of models throughout the development lifecycle
    - Managing wider abstraction gaps

When, where, what

- **Introductory Programming**: Illustrate OO programming concepts through models
  - Program structure: use class diagrams in introductory OO programming courses to illustrate program structure
  - Program behavior: use sequence diagrams to illustrate how objects interact in an OO design
- **Basic Programming** (basic data structures & algorithms): Using models to conceptualize program designs
  - Students required to develop initial models of their designs before coding solutions to small problems

Developing abstractions skills

- **Advanced Programming**: Using models to conceptualize more complex program designs
  - Present and discuss examples of good and bad program designs
- **Software Engineering**: Developing modeling-in-the large skills
  - Use of design studios to nurture abstraction skills
  - Present and discuss examples of good and bad modeling practices

It would be good to have …

- Modeling patterns and anti-patterns that distill expert modeling experience
- A repository of models that illustrate good and bad modeling practices (coming soon in ReMoDD)
- Text books that focus on developing modeling skills rather than on covering syntactic and semantic language concepts
- Lightweight modeling tools that tolerate incompleteness and support exploratory design.