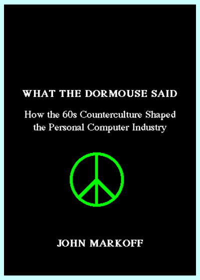
Data-Driven Programs

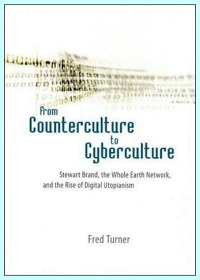
Jerry Cain
CS 106AJ
November 16, 2018
slides courtesy of Eric Roberts, with edits by me

Once upon a time . . .

Computing and the Counterculture

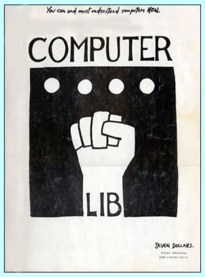
Two recent books argue that the personal computing revolution owes as much to the counterculture of the 1960s as it does to the technological strength and entrepreneurial spirit of Silicon Valley.

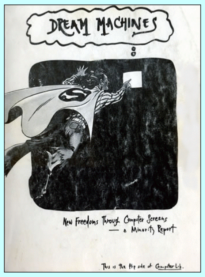

WHAT THE DORMOUSE SAID
How the 60s Counterculture Shaped the Personal Computer Industry
JOHN MARKOFF


From Counterculture to Cyberculture
Stewart Brand, the Whole Earth Network, and the Rise of Digital Utopianism
Fred Turner

Ted Nelson's Cyberspace Dreams

The countercultural vision comes across particularly clearly in the two-sided book *Computer Lib/Dream Machines* which was written by cyberspace visionary Ted Nelson in 1974.





Data-Driven Programs

Data-Driven Programs

- In most programming languages, data structures are easier to manipulate than code. As a result, it is often useful to design applications so that as much of their behavior as possible is represented as data rather than in the form of methods. Programs that work this way are said to be ***data driven***.
- In a data-driven system, the actual program (which is called a ***driver***) is usually very small. Such driver programs operate in two phases:
  1. Read data from a file into a suitable internal data structure.
  2. Use the data structure to control the flow of the program.
- To illustrate the idea of a data-driven system, we're going to spend most of this lecture building a programmed-instruction "teaching machine" of the sort that Ted Nelson discusses (mostly critically) in *Dream Machines.*

A New Approach to Files

- Older versions of the teaching machine program in the textbook read the series of course questions from a plain text file chosen by the user.
- Although the file-based model is standard practice in most other programming languages, it doesn't make sense for CS 106AJ. Security considerations make traditional file reading in JavaScript too cumbersome to use.
- Fortunately, JavaScript offers an elegant solution to the problem. All you need to do is store the necessary data in the `index.html` file, which JavaScript programs can always read.
- CS106AJ uses this strategy for both the teaching machine example and the Adventure assignment, which should make your lives much easier after the Thanksgiving holidays.

XML and the DOM

- The `index.html` file is written in the ***Hypertext Markup Language*** (HTML), which is a subset of the ***Extensible Markup Language*** (XML). In recent years, XML has become the industry standard for representing hierarchical data.
- Modern browsers parse all the XML data in the `index.html` file and store that information in an internal form called the ***Document Object Model*** (DOM).
- Much of JavaScript's undeserved negative reputation comes from the fact that the DOM is a complete mess.
- In general, the best strategy is to use as little of the DOM as possible. The three methods and one property outlined on the next page are all you need.
- To keep the user data from appearing on the page, you need to embed its XML in a `<div style="display:none;">` block.

Four Handy DOM Directives

`document.getElementById(`*id*`)`	Returns the element with the specified id attribute.
element`.getElementsByTagName(`*name*`)`	Returns an array of the elements with the specified tag name.
element`.getAttribute(`*name*`)`	Returns the value of the named attribute.
element`.innerHTML`	Returns the HTML under the jurisdiction of an element.

Sample DOM Method Calls

```
<div style="display:none;">
  <question id="Numbers1">
    True or false: Numbers in JavaScript may contain a decimal point.
    <answer response="true" nextQuestion="Finish" />
    <answer response="false" nextQuestion="Numbers2" />
  </question>
  <question id="Numbers2">
    True or false: Numbers can be negative.
    <answer response="true" nextQuestion="Finish" />
    <answer response="false" nextQuestion="Numbers1" />
  </question>
  <question id="Finish">
    You seem to have mastered JavaScript. Start over?
    <answer response="yes" nextQuestion="Numbers1" />
    <answer response="no" nextQuestion="EXIT" />
  </question>
</div>
```

- `let question = document.getElementById("Numbers2")` would return an object representing the second question element, and everything descending from it. I highlight what's modeled by **`question`** in **green.**
- `let answers = question.getElementsByTagName("answer")` would return an length-2 array of two objects—the 0th object models the second question's response to `"true"`, and the 1th object models the second question's response to `"false"`. The HTML elements modeled by these two objects are *italicized*.
- `let next = answers[0].getAttribute("nextQuestion")` would return the `"Finish"`, since that's the string value attached to `answers[0]`'s. `nextQuestion` attribute. The relevant `"Finish"` is underlined.
- `console.write(question.innerHTML + "<br/>")` would print `"True or false: Numbers can be negative."`

Reader code for the `TeachingMachine`

```
/**
 * This program executes a programmed instruction course
 * as discussed in Section 12.5 of the course reader.
 */
function TeachingMachine() {

   let askQuestion = function() {
      console.write(questionXML.innerHTML + "<br/>");
      console.requestInput("> ", checkAnswer);
   };

   let checkAnswer = function(response) {
      let answerXML = getAnswerXML(response.toLowerCase());
      if (answerXML === null) {
         console.log("I don't understand that answer.");
      } else {
         let questionID = answerXML.getAttribute("nextQuestion");
         if (questionID === "EXIT") return;
         questionXML = document.getElementById(questionID);
      }
      askQuestion();
   };
```

Reader code for the `TeachingMachine`

```
   let getAnswerXML = function(response) {
      let answers = questionXML.getElementsByTagName("answer");

      for (let i = 0; i < answers.length; i++) {
         let answerXML = answers[i];
         let possibility = answers[i].getAttribute("response");
         possibility = possibility.toLowerCase();
         if (possibility === response || possibility === "*") {
            return answerXML;
         }
      }

      return null;
   };

   let questionXML = document.getElementsByTagName("question")[0];
   askQuestion();
}
```

Redesigning `TeachingMachine`

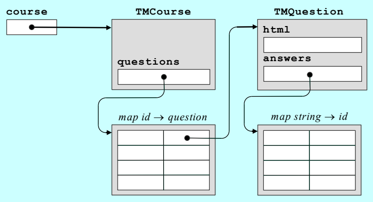
- As presented in the reader, the `TeachingMachine` is a lovely example that uses data—in particularly, data embedded as XML within the `index.html` file—to drive an executable.
- There are benefits, however, to ingesting all of the XML into JavaScript data structures so they are more quickly and easily manipulated.
- The teaching machine teaches a single course composed of questions, where each question is composed of a statement and a collection of correct and incorrect answers.
- The next several slides provide a defense for an object-oriented approach to building the `TeachingMachine` and speak to the design and implementation of several classes to model courses, questions, and answers.

Designing Data Structures

- When you design data structures for a large program, one of the first tasks you need to undertake is understanding how the underlying data structures fit together and how each level of the data hierarchy can best be represented.
- This process is similar to that of decomposing a large problem into a set of successively simpler sub-problems. In the data domain, the information your program needs to process must be decomposed into successively simpler data structures until everything can be represented using a built-in JavaScript value, such as a number or a string.
- The tools for data decomposition you have seen so far include
  - *Arrays,* which implement sequences of values
  - *Aggregates,* which represent collections of related values
  - *Maps,* which establish a relationship between keys and values

Choosing an Internal Representation

The first step in building an OO teaching machine is to design a set of classes that can represent the data and their relationships. All the relevant data should be accessible from a single structure that stores the information in a nested series of classes.



Converting XML to Internal Form

```
<div style="display: none">
 <question name="Q1">
  True or false: Numbers can
  have fractional parts.
  <answer response="true"
          nextQuestion="Q3" />
  <answer response="false"
          nextQuestion="Q2" />
 </question>
 <question name="Q2">
  That's incorrect.
  True or false: Numbers can
  be negative.
  <answer response="true"
          nextQuestion="Q3" />
  <answer response="false"
          nextQuestion="Q1" />
 </question>
 <question name="Q3">
  You seem to have mastered
  JavaScript. Start over?
  <answer response="yes"
          nextQuestion="Q1" />
  <answer response="no"
          nextQuestion="EXIT" />
 </question>
</div>
```

New code for the `TeachingMachine`

```
/*
 * File: TeachingMachine.js
 * ------------------------
 * This program serves as the entry point to a more
 * object-oriented realization of the teaching machine
 * discussed in the textbook, and will serve as a much
 * better template for your Adventure assignment than the
 * version in the course reader.
 */

function TeachingMachine() {
   let course = TMCourse();
   if (course === undefined) {
      console.log("No course is defined in the HTML file");
   } else {
      course.run();
   }
}
```

Code for the `TMCourse` Class

```
/*
 * File: TMCourse.js
 * -----------------
 * This class defines the data structure for a course used by
 * the TeachingMachine program.
 */

/*
 * Creates a TMCourse object by extracting data from
 * the index.html file.
 */
function TMCourse() {
   let questions = readQuestions();
   let currentQuestion = questions["START"];
   function askQuestion() { /* code on next slide */ }
   function checkAnswer() { /* code on next slide */ }
   let course = {};
   course.run = askQuestion;
   return course;
}
```

The `TMCourse.run` Method

```
   function askQuestion() {
      currentQuestion.printQuestionText();
      console.requestInput("> ", checkAnswer);
   };

   function checkAnswer(line) {
      let questionID = currentQuestion.getNextQuestion(line);
      if (questionID === "EXIT") return;
      if (questionID === undefined) {
         console.log("I don't understand that response.");
      } else {
         currentQuestion = questions[questionID];
      }
      askQuestion();
   };
```

Code for the `TMQuestion` Class

```
/**
 * Class: TMQuestion
 * -----------------
 * This class represents a question for the teaching machine.
 * Intuitively, a TMQuestion object encapsulates the HTML markup
 * and a map from each permitted answer to the next question id.
 *
 * The supplied parameters are private to the implementation
 * and are only accessed by invoked TMQuestion methods.
 */

function TMQuestion(html, answers) {
   let question = {};
   question.printQuestionText = function() {
      console.write(html + "<br/>");
   };
   question.getNextQuestion = function(answer) {
      let match = answers[answer.toLowerCase()];
      if (match === undefined) match = answers["*"];
      return match;
   };
   return question;
```

The `readQuestions` Function

```
/**
 * Creates the questions data structure by reading the elements
 * with the tag "question" from the XML for the course.  To ensure
 * that the course starts with the first question in the file, the
 * map stores a reference to that question under the key "START".
 */

function readQuestions() {
   let elements = document.getElementsByTagName("question");
   if (elements.length === 0) return undefined;
   let questions = {};
   for (let i = 0; i < elements.length; i++) {
      let questionXML = elements[i];
      let id = questionXML.getAttribute("id");
      let html = questionXML.innerHTML;
      let answers = readAnswers(questionXML);
      questions[id] = TMQuestion(html, answers);
      if (i === 0) questions["START"] = questions[id];
   }
   return questions;
}
```

The `readAnswers` Function

```
/**
 * Reads the data structure containing the answers, which maps
 * user responses to the DOM id of the next question.
 */

function readAnswers(questionXML) {
   let answers = { };
   let elements = questionXML.getElementsByTagName("answer");
   for (let i = 0; i < elements.length; i++) {
      let answerXML = elements[i];
      let response = answerXML.getAttribute("response");
      let nextQuestion = answerXML.getAttribute("nextQuestion");
      answers[response.toLowerCase()] = nextQuestion;
   }
   return answers;
}
```

The End