JDuck: Building a Software Engineering Tool as a CS2 Project

Michael W. Godfrey (Univ. of Waterloo)
Daniel J. Grossman (Cornell Univ.)

Outline

- CS211 @ Cornell
- Goals for the project
- Overview of JDuck
- Results, feedback, *Golden Ducks*
- Conclusions: JDuck as a learning experience

CS211 @ Cornell

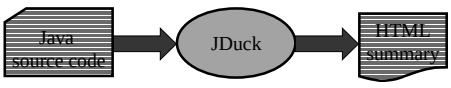
- One of Cornell's two CS2 courses:
  - CS majors encouraged to take CS212
- Incoming students:
  - mostly non-majors who are required to take it
  - varying backgrounds (C/C++, Java, Pascal, HS)
- Goals:
  - Reach out to everyone.
  - Teach them something they will remember.

Project Goals

- Re-enforce lecture material (OOP, ADTs)
- A real chunk of work:
  - work in groups,
  - staged development,
  - extend an existing infrastructure,
  - easy enough to be doable,
    hard enough to be interesting,
    flexible enough to be fun, and
    compelling enough to be memorable.

JDuck: A Simple Software Engineering Tool

- Java DocUmentor of Code, oK?
- Idea based on **`javadoc`** tool in Sun's JDK.
- It grinds up code:
  ⇒ generate an HTML summary for each Java source class.



JDuck: Output Specification

- Generate an HTML summary for each Java source class:
  - class name, its package, what it imports, implements, extends
  - what variables, methods, constructors it defines (precise syntax, visibility, static ?, final ?)
  - hyperlinks to other classes mentioned
  - inherited methods/attributes (extra credit)

JDuck: Output Specification

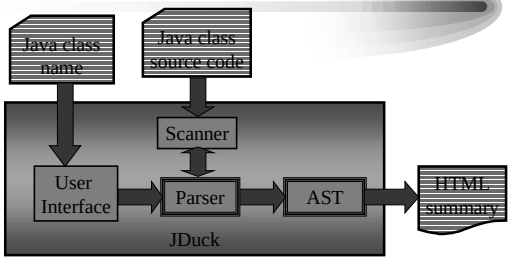
- Present features in this order:
  1. static variables
  2. instance variables
  3. constructors
  4. static methods
  5. instance methods
- "Visually pleasing" *(but don't go crazy)*

JDuck: Project Structure

- We gave them:
  - a scanner (JLex),
  - a simplified grammar for Java,
  - a predefined top-level user interface,
  - an output format specification,
  - tutorials on HTML, scanning/parsing, and
  - advice on how to proceed.

Then we turned them loose!

JDuck: Architecture of a Solution



Simplifying Assumptions

- Can safely ignore method bodies by counting curlies (*i.e.,* "{" and "}")
- Ignore comments (!) , arrays, initializing expressions, *etc.*
- Variables and methods declared separately
  - special comments mark beginning of each section: `//Variables` `//Methods`
- Code is assumed to be well formed.

Extra Credit Extensions

- Go up the inheritance hierarchy!
  - Parse parental information.
  - Indicate which features are inherited.
  - Watch out for private features in ancestors.
  - Look at parameters to see if this is overriding or overloading!
- Several groups tried this, but it was hard to get completely correct.

Simplified Java Grammar

```
[import LIBRARY;]*
[package PACKAGE_NAME;]
[public] [abstract] [final] class CLASS_NAME
[extends SUPERCLASS_NAME]
[implements INTF_NAME [, INTF_NAME]* ]
    {
        //Variables
        Variable_Decl*
        //Methods
        Method_Decl*
    }
```

Preparing the Students

- CS211 had five assgts plus the project.
  - Assgt #3: parse simple command language
- Tutorials:
  - use of scanner for simple examples
  - parsing
  - HTML and simple visual layout
  - discussion of JDuck software architecture

Testing

- We encouraged an open exchange of test cases by students … but little response.
- We warned the students that we would be thorough.
- Mass testing went surprisingly easily due to top-level "hook" that we required they implement.

Testing

- We developed a *secret* test suite to check as many cases as we could think of:
  - normal use/absence of all features
  - different legal orderings
  - no/one/many variables, no/one/many methods
  - empty/full method bodies, *etc.*

Evaluating the Solutions

- We defined a top-level UI they *had* to conform to.
  - GUI and non-GUI versions required.
- Students handed in diskette w. code, printouts of code and one test run.
- We compiled their solutions and ran them against our nasty set of tests.

Evaluating the Solutions

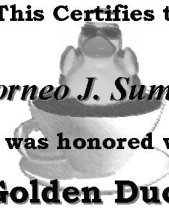
- Submissions were graded in bulk by undergrad consultants:
  - Test case failures $\Rightarrow$ diagnose in code.
  - Look out for bad style too.
- "Visual design" worth only 5%
  - Many entries were quite elaborate and creative!

The Golden Duck Award

- Five *Golden Ducks* chosen from 145 submissions (270 students).
- *Golden Duck* criteria:
  - Pass all correctness tests.
  - Good use of OO programming style and design.
  - Compelling visual appearance.

Cornell University CS211 Spring 1998

This Certifies that



Borneo J. Sumatra

was honored with

The Golden Duck Award

for Java Programming

Professor Michael Godfrey

Golden Ducks -- Creole Style

- A (new) example source file:
  - NewOrleans.java
- Some *Golden Duck* output:
  - Dave Rollenhagen
  - Charitha Tillerkeratne and Kfir Shay
  - George Chang

Conclusions: JDuck as a Learning Experience

Fundamental CS:

- Designed and used non-trivial OO data structures, trees, recursion.
- Exposure to some advanced CS topics:
  - scanning and parsing
  - simple design pattern (the *visitor* pattern)

Conclusions: JDuck as a Learning Experience

Technology:

- Exposure to a real software engineering tool.
- Basics of HTML and web design.

Conclusions: JDuck as a Learning Experience

Software Engineering Education:

- Built a “big” system.
- Worked in a team of two.
- Scale enforced some discipline.
- Staged development.
- Test cases design and (harsh) validation.

Student Feedback

- Some real surprises:
  - Task appeared daunting at first, but was tractable if they followed our advice.
  - Only a few disasters.
  - Likely, we could have made it harder!

Student Feedback

- Many said they enjoyed being led by the hand through the development of a big piece of software.
  - Too often, we give little advice on *how* to proceed.
- Web design was fun but time consuming.

JDuck: The Next Generation

- Worked well, students enjoyed it,
  I really oughtta try it again someday …
- Make it harder by adding new requirements.
  - Easy to find new requirements, tweak the old ones to make the project different.
- Release one nasty test suite ahead of time to encourage paranoia and test suite exchange.

JDuck: Renewable Resources

- Many, many thanks to the TAs:
  - Dan Grossman, Max Khavin, Kristen Summers, Linda Lee, Evan Gridley, Martin Handwerker.
- All resources (except the example solution) available on the JDuck homepage:

`http://plg.uwaterloo.ca/~migod/jduck/`