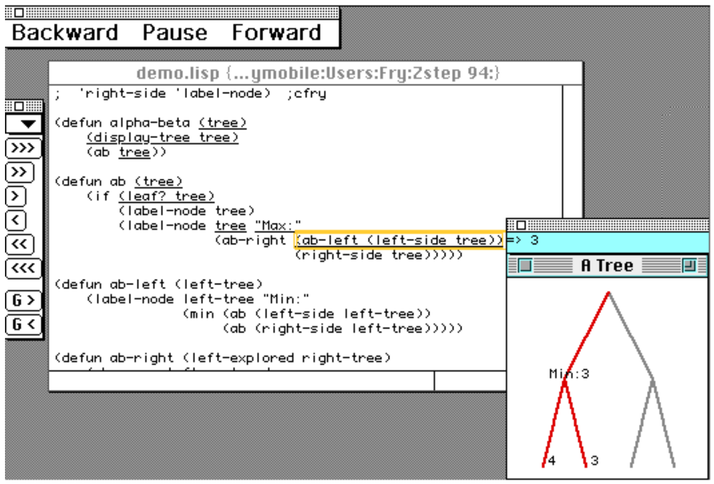
Programming as Driving: Unsafe at Any Speed?

Christopher Fry
Harlequin, Inc.
1 Cambridge Center
Cambridge, Mass. USA
cfry@harlequin.com

Henry Lieberman
Media Laboratory
Massachusetts Institute of Technology
Cambridge, Mass. USA
lieber@media.mit.edu



ABSTRACT

Programming is dangerous. As programmers, we are still driving the equivalent of a '57 Chevy: the chrome plated bumpers on our programming environment might look good while it's cruising down the road, but it's not very efficient with [mental] fuel, and it's all too likely to crumple in a crash. No seat belts, no anti-lock brakes, and the rear view mirror is obstructed by the fuzzy dice.

ZStep 94 is a reversible, WYSIWYG, animated, source code debugger that brings programming into the safety conscious '90s. It provides safety and efficiency options not found on the used car dealer's lot.

SAFETY EQUIPMENT

All programs are accident-prone. It took lawsuits and bad publicity from such debacles as the Corvair and Pinto to wake up the car industry to designing cars to expect the possibility of crashes and protect the driver. Most current programming environments are like the old Corvair; when an error occurs, the programmer is left staring at nothing but the flaming wreckage of an error message and perhaps a core dump. Every program crash totals the vehicle, and nothing can be learned from the experience except to buy a new car and start over.

A basic piece of safety equipment missing from most programming environments is the rear-view mirror. Since many traffic hazards sneak up on you from behind, the ability to see where you've just been is essential. ZStep 94 keeps a complete history of the computation, and can run the program both forward and backward. There is fine control over the level of detail, essential in those tight parallel parking situations. In conventional environments, even if some rear view is provided by stack inspection, the view is obscured by fuzzy dice and tree-shaped air fresheners such as the loss of intermediate values in the computation and the warning that objects in the mirror may be closer than they appear. ZStep 94 tries to assure that the view of past states resembles as closely as possible the original state.

ZStep 94 installs airbags for the programming environment. An error is "fail-soft", and doesn't throw the programmer out of the programming environment, or into a special breakpoint loop. The recorded history of the computation, including partial results, is still available for inspection, and the program can be run forward or backward from any point before the error occurred. Even if the car is damaged, you can still examine the undamaged portions or the site of impact. The driver can walk away from the scene of the accident without a scratch.

Further, the history keeping acts like an airplane's "black box", providing the information necessary for post-accident analysis and providing the basis for improving safety measures in the future. A video camera pointed out the side window recording the history is useful to understand those unfortunate side effects that you sometimes observe during the trip, just as the motorist who taped the Rodney King incident.

ZStep 94 keeps a first aid kit in the glove compartment for easy access to code repair tools. There is a convenient cigarette lighter interface to a data inspector.

NAVIGATION TOOLS

The goal of driving isn't just to make the car go; it's to get where you're going. ZStep 94 replaces the '57 Chevy door map pocket with a modern GPS satellite positioning system that displays a dynamic "you are here" cursor.

When a bug happens, the programmer is usually placed in the role of an ambulance driver racing to the scene of an accident. S/he's asleep at the wheel in the hospital garage when the car radio blurts out "Car 54, where are you? There's been a crash in the city. Victim last seen heading down Main (); Street". The poor driver of the '57 Chevy wagon ambulance has a map of the city, a full tank of Jolt, and flames painted on hood indicating a long history of being burned.

The first problem is to find the scene of the accident. In fact, that's *the* problem. The static representation of the code, the map, doesn't help a whole lot because the dynamic execution goes the wrong way down one way streets, finds blind alleys that are too small to see on the map and discovers traffic jams that can't be represented on static media. The task of the ambulance driver might not be much easier even if the driver happens to be the city planner, the same person who laid out the streets in the first place.

For programming, the navigation problem is understanding the relationship between the static code and the dynamic process of execution. ZStep 94 displays the static code in the user's editor buffer, exactly as it was typed, including ccomments and formatting. Expressions to be evaluated and the stepper's state are highlighted, and a floating window follows the focus of execution and displays the current value. This is like a car with transparent floorboards enabling you to see the source code painted on the street. Street signs at every corner display the value of the executed source code that you just drove over.

SPEED CONTROL

We replace the '57 Chevy 3-speed manual gearbox with a 4-speed manual *and* automatic transmission, in both forward and reverse, permitting a variety of execution speeds. The fine control helps you find that one empty parking space in the Haystacks-Calhoun parking garage, yet lets you zoom out so you don't miss the Forrest Ave. exit on the Trees Interstate Highway. Optional equipment includes cruise control for programmers too busy necking to click the mouse.

FUEL EFFICIENCY

The mental energy of the programmer is the fuel that runs programming environments. ZStep 94 features aerodynamic styling that conserves programmer energy.

Care is taken to make sure that relevant information appears on the screen when and where it is needed and in the appropriate static and dynamic context. ZStep 94 provides one-click access from expressions in the code to their values and graphical output, and one-click access from graphical objects to the code that drew them.

This reduces air drag caused by the programmer moving the mouse to another part of the screen, and air turbulence caused by the programmer turning his or her head to see information displayed in separate windows. This allows ZStep 94 to get more mileage* out a very expensive resource - hacker power.

For a more academic treatment of ZStep 94, read and attend [4]. For a less academic treatment, come to the demo!

REFERENCES

[1] Ron Baecker, Two Systems Which Produce Animated Representations of the Execution of Computer Programs, SigCSE Bulletin, February 1975.

[2] Marc Eisenstadt and Mike Brayshaw, The Transparent Prolog Machine: An Execution Model and Graphical Debugger for Logic Programming, Journal of Logic Programming 5(4), p.1-66, 1988.

[3] Henry Lieberman, Steps Toward Better Debugging Tools for Lisp, ACM Symposium on Lisp and Functional Programming, Austin, Texas, August 1984

[4] Henry Lieberman and Christopher Fry, Bridging the Gulf Between Code and Behavior in Programming, CHI '95.

[5] Stewart Watt, Froglet: A Source-Level Stepper for Lisp, Human Cognition Research Laboratory, Open University, Milton Keynes, England, 1994.

---

* Estimated Mileage: 33.3 expressions/hour for city hacking, 45 exp/hr on the Information Superhighway. However, your mileage may vary.