Requirements/Design Debugging
(Session Summary)

Peter Bates
University of Massachusetts

Session Chair: Robert Balzer

Participants:
Lori Clarke
Don Cohen
Jim Cunningham
David Snowden
D.G. Shapiro
Bernd Brugge
Claude Jard
Bill Swartout
Elliot Soloway

This session was essentially a continuation of the earlier session on Knowledge-Based systems. The discussion proceeded along two dimensions. One is the paradigm used to, in some sense, debug high-level specifications. This is further broken into the 'current' and the 'operational specifications' paradigms. The other dimension is one of techniques for implementing these views. This dimension also has only two aspects, current techniques and knowledge-based techniques.

The plan for the session was to introduce the technologies that may or may not exist to support these capabilities followed by a debate on these methods. There are four areas to be discussed in terms of technology--symbolic evaluation, fault analysis, behavior expectations and natural language explanations.

1.0 Symbolic Evaluation

The first speaker, Lori Clarke, posed two questions to be considered when looking at software analysis in terms of debugging. The first, "How can we validate our specifications?", assumes there exists some pre-implementation description that is higher level than the code being produced. The second was "How can we use previously obtained analysis information [from the validation process] in the debugging process?"

Don Cohen works on validating Gist specifications. He indicated that the real problem is just finding that there is a bug with the specification since once it is found it is obvious where it came from. The position to be taken is that just about anything that can be done to show the user another view of the specifications should help the user find bugs. The plan is to have something look at a Gist specification and explain to the user what the specification says.

The next speaker, Jim Cunningham presented the parts of a specification as seen by the Aver project. A specification has three parts, the components of state consisting of objects and their relations, the actions which change states, and provision for constraints which will indicate when there are problems in the system. Following from this view, the formal use of tools in constructive design has two sides, in validation of the design and verification of the design. In verification, the offending system state can be isolated and used to show the components of that offending state. On the validation side, with a suitably good theorem prover, problems with the specification should be findable.

2.0 Fault Analysis

Fault analysis is reasoning from a functional defect to a structural defect in a program. Issues involved in construction of intelligent fault analysis systems are related to: 1) program representation--there is a need for more than one way to represent programs at various levels of abstraction; 2) fault representation--likewise

there is a need for defining and representing what a fault is at each level of abstraction; 3) reasoning mechanism--having a way to reason about faults in a given program representation; and 4) knowledge acquisition--where does knowledge on constructing levels of abstraction come from.

David Snowden works with an expert system that attempts to find declaration and scope errors in Ada programs. The expert has a hierarchy of rules that hold knowledge about declaration and scope errors. In operation, the system selects an error message and attempts to determine what types of errors could give rise to that message. A set of instance representations of the error type are generated and compared to how well they explain the error given the current information.

D.G. Shapiro related that there are many things that every good programmer knows and it is desirable to use that knowledge to find what is wrong with programs. In his approach an alternative view of the program is generated and a collection of recognizers of plans for programming cliches attempt to figure out what is intended. Bug experts are run and produce a bug analysis about bugs related to that cliche. The overriding need is to be able to identify what the programmer's intentions really were.

3.0 Formal Descriptions Of Behavior Expectations.

Bernd Brugge described his work using predicate path expressions as a method of describing some behaviors expected of a system. The technique allows users to specify behaviors and indicate some actions to perform when the behaviors match or fail to match the system's activity.

Claude Jard uses an extended state-transition model for validation of specifications. His system will then compare a detailed specification with user entities to detect errors.

4.0 Natural Language Explanations

Bill Swartout gave an overview of the structure of their system for explaining Gist specifications. Basically it provides methods for describing particular types of Gist constructs. The methods attempt to highlight surprising behavior that may be discovered by a symbolic evaluator.

Elliot Soloway described Proust, a system that finds bugs in novice programs. The basic premise is that programmers use plans when they read and write programs. Proust has these plans and it proceeds to generate explanations based on the plans and their goals.

5.0 Final Discussion

The session chair Bob Balzer then proposed a set of so called 'religious questions' as a basis for a discussion.

1. What does the operational specifications paradigm say about the programming process?
2. Is debugging an implementation the same as debugging the specifications?
3. What is the nature of the debugging process if big programs start to be built by evolution?
4. Specifications are easier to debug than implementations.
5. Implementation is easier to debug than specifications.