VanLehn, K. (1982) Bugs are not enough: Empirical studies of bugs, impasses and repairs in procedural skills. Journal of Mathematical Behavior, 3(2) pp. 3-71.

BUGS ARE NOT ENOUGH: EMPIRICAL STUDIES OF BUGS, IMPASSES AND REPAIRS IN PROCEDURAL SKILLS

Kurt VanLehn

Xerox Palo Alto Research Center

ABSTRACT

Prior to the empirical studies reported here, it was felt that errors in procedural skills such as multi-digit subtraction could be accurately modelled with two mechanisms: bugs and slips. Slips are taken as "performance" phenomena that were expected to be highly unstable over time and only loosely related to the subtraction problems they occur in. Bugs are "competence" phenomena reflecting mistaken beliefs about the skill and as such, are expected to be consistent across a whole test and stable across tests given some days and even months apart. A computational descriptive framework, the "Buggy" paradigm, and a sophisticated diagnostic program, DEBUGGY, were developed wherein bugs and slips modelled performance at a very fine level of detail, describing the content of the wrong answers as well as the steps taken in producing them.

This report presents the results of several empirical studies. 925 students who were in the process of learning subtraction were tested using highly diagnostic tests developed by DEBUGGY. Some students were retested two days later to measure the short-term stability of bugs, and others were retested several months later to study long-term stability. All tests were analyzed by DEBUGGY and by several expert diagnosticians in order to assess DEBUGGY's diagnostic abilities.

It was found that DEBUGGY was as good or better than human diagnosticians at discovering bugs that explain a student's errors. How-

ever, a third of the students who committed errors could not be modelled with bugs and slips. Moreover, bugs were found in general to be unstable rather than stable, both in the short term and the long term. These findings challenged the belief that bugs and slips alone could account for procedural error data.

Repair Theory was originally developed as a generative theory of bugs, one that predicted which bugs would exist for a given procedural skill. However, it also predicted that certain kinds of non-bug, non-slip performance would exist, both in the static analyses and the stability data. These predictions were verified. It now appears that all but a small (but still significant) fraction of the phenomena can be precisely modelled using bugs, slips and the mechanisms of Repair Theory.

It has long been known that many of the errors that students make in exercising a procedural skill, such as ordinary place-value subtraction, are systematic in that the errors appear to stem from consistent application of a faulty method, algorithm or rule. These errors occur along with the familiar unsystematic or "careless" errors which occasionally occur in expert performance as well as the learner's behavior. The common opinion is that careless errors, or "slips" as we prefer to call them (c.f. Norman 1981), are performance phenomena, an inherent part of the "noise" of the human information processor. Systematic errors on the other hand are taken as stemming from mistaken or missing knowledge about the skill, the product of incomplete or misguided learning. By studying them, insight can be gained into the mysterious processes of learning and memory. It is also commonly believed that there are a relatively small number of systematic errors for any given skill, perhaps a dozen or a hundred, and that once a student has acquired one of these unfortunate habits, it will be held until it is remediated. The data reported here challenge some of these basic beliefs, while supporting others.

In the last several years, a large scale investigation of errors in procedural skills has been conducted, with a special emphasis on systematic errors. It began with the idea that systematic errors could be formally represented and precisely described as "bugs" in

a correct procedure for the skill. In brief, a bug is a slight modification or perturbation of a correct procedure. The bug-based notation is *complete* in the sense that it not only describes which problems the student gets wrong, but what each wrong answer is and the steps followed by the student in producing it. This fine grained description raises a number of questions, such as: How many different bugs are there? How are these bugs acquired by students? How long are they held? What makes them go away? A question of fundamental importance is whether there are any students whose errors can be described neither as bugs nor as slips. This question challenges the foundational belief that errors are either systematic (i.e. deterministic, procedural) or unsystematic (i.e. careless, unintended).

To answer such questions, a large number of student errors have been collected and analyzed in terms of bugs and slips. These data were analyzed with the aid of DEBUGGY (Burton, 1981), a computer program that can determine which bugs, if any, underlie a student's errors. Thus, in addition to reporting data that bears on potential psychological and pedagogical theories, this paper provides an assessment of a particular approach to computerized student diagnosis.

This report can be read at three levels of detail. The first level of detail weaves a summary of the results with an introduction to the bug formalism for describing errors and a description of a recent theory, Repair Theory (Brown & VanLehn, 1980), that aims to explain the acquisition of bugs. It is meant to be a glossary of the concepts used to analyze the data as well as a quick synopsis of the findings. The first level of detail is section 1. The second level of detail is contained in the following sections. The choice of subtraction as a task domain as well as the methods for gathering and analyzing the data are discussed. The empirical predictions of Repair Theory are checked, and an extensive discussion of the adequacy of DEBUGGY-based diagnosis is presented. The third level of detail is contained in a set of appendices, which present the data in tabular form and discuss some relatively minor points concerning its aggregation and tabulation.

1. AN INTRODUCTION TO THE CONCEPTS AND THE FINDINGS

Slips in arithmetic hardly need an introduction since as adults all our arithmetic errors are slips. We have all experienced the forgotten carry, the unnecessary borrow, and of course the ever present "facts errors" wherein one mis-remembers an elementary number combination. These apparently careless, unintentional errors also occur in the work of students learning arithmetic. Of 895 third, fourth and fifth grade students, 182 (20%) were analyzed by DEBUGGY as knowing the correct algorithm for subtraction but making one or more slips during testing.

When students were tested twice a few days apart, half the students who answered all problems correctly one day made slips the other day. This finding confirms the impression that slips are unintentional, careless mistakes in that a little extra care (or something!) apparently makes them disappear. So, slips explain a fair number of student's errors, and their unstable existence conforms with the intuitive expectation that they are due to "noise" in the processor rather than conceptual defects.

What's a bug?

The following problems display a systematic error:

306	80	183	702	3005	7002	34	251
-138	- 4	- 95	- 11	- 28	- 239	-14	- 47
78	76	88	591	1087	4873	24	244

One could vaguely describe these problems as coming from a student having trouble with borrowing, especially in the presence of zeros. More precisely, the student misses all the problems that require borrowing from zero. One could say that he has not mastered the subskill of borrowing across zero. This description of the systematic error is fine at one level: it is a testable prediction about what new problems the student will get wrong. It predicts for example that the student will miss 305-117 and will get

315-117 correct. Systematic errors described at this level are the data upon which several psychological and pedagogical theories have been built (e.g. Durnin & Scandura, 1977). It has become common to use testing programs based on this notion for placement, advancement and remediation in structured curricula, such as mathematics. Such testing programs are often labelled "domain referenced" or "criterion referenced."

Once we look beyond what *kinds* of exercises the student misses and look at the actual answers given, we find in many cases that these answers can be precisely *predicted* by computing the answers to the given problems using a procedure which is a small perturbation in the fine structure of the correct procedure. Such perturbations serve as a precise description of the errors. We call them "bugs."

The student whose work appears above has a bug called Borrow-Across-Zero. This bug modifies the correct subtraction procedure by deleting the step wherein the zero is changed to a nine during borrowing across zero (this bug and others like it are described more thoroughly in appendix 1). This modification creates a procedure for answering subtraction problems. As a hypothesis, it predicts not only which new problems the student will miss, but also what his answers will be. For example, it predicts that the student above would answer 305-117 = 98 and 315-117 = 198. Since the bug-based descriptions of systematic errors predict behavior at a finer level of detail than missing-subskill/domain referenced testing, they have the potential to form a better basis for cognitive theories of learning and errors, and perhaps a better basis for remediation or placement as well.

It is often the case that a student has more than one bug at the same time. Indeed, the example given above illustrates co-occurrence of bugs. The last two problems are answered incorrectly but the bug Borrow-Across-Zero does not predict their answers (it predicts the two problems would be answered correctly). A second bug, called Diff-N-N = N is present. When the student comes to subtract a column where the top and bottom digits are equal, instead of writing zero in the answer, he writes the digit that appears in the column.

It often takes a set of bugs to form an accurate description of a student's errors. Of the 417 students that DEBUGGY analyzed as having bugs, 150 (36%) had a multi-bug diagnosis. Most of these diagnoses consisted of two or three bugs, but there were several cases of four bugs co-occurring. So, DEBUGGY's ability to combine bugs to form an accurate diagnosis turned out to be very important.

A Brief Look at the Bug Data

Overall, 77 distinct bugs occurred (by "occurred," we mean that a student had the bug as his diagnosis, or if he was diagnosed as having a set of bugs, as part of his diagnosis). A few bugs occurred quite often. The most common bug by far was Smaller-From-Larger (this bug never borrows but instead simply takes the absolute difference in each column). It occurred 106 times alone, and 18 times as part of multi-bug diagnoses. From there the frequency fell off rapidly, with the next five most common bugs coming in at 67, 51, 40, 22 and 19 occurrences. About half the bugs (32) were quite rare, occurring only once or twice. This marked skew in the frequencies of occurrence explains the impression left by informal studies that there are only a dozen or so systematic errors. In fact, there are many more, but it took precision analysis of thousands of students to find them. This raises the question, how could so many bugs come to exist?

Repair Theory: Repairs at Impasses

Repair Theory is a *generative* theory in that it attempts to explain why we found the bugs that we did and not other ones, to explain how bugs are caused, and most importantly, to predict what bugs will exist for procedural skills we have not yet analyzed. There are several benefits of a generative theory. We could automatically generate a list of bugs for a new skill and add these bugs to DEBUGGY, creating a diagnostic system tailored to the new skill. We could attack the issue of remediation of bugs with more than just a knowledge of what bugs a student has since such a theory

would provide a plausible basis for understanding why the student had those bugs. Such an understanding could also help us design learning environments that might inhibit formation of those bugs in the first place. Finally, in terms of cognitive research, such a theory would provide insights into knowledge representations and cognitive mechanisms (e.g. skill acquisition) that defy direct observation.

Repair Theory is based on the insight that when a student gets stuck while executing his possibly incomplete subtraction procedure, he is unlikely to just quit as a computer does when it can't execute the next step in a procedure. Instead, the student will do a small amount of problem solving, just enough to get "unstuck" and complete the subtraction problem. These local problem solving strategies are called "repairs" despite the fact that they rarely succeed in rectifying the broken procedure. Repairs are quite simple tactics, such as skipping the operation that can't be performed or backing up to the last branch point in the procedure and taking a different path. They do not in general result in a correct solution to the subtraction problem, but instead result in a buggy solution. For example, suppose the student has never borrowed from zero. The first time he is asked to solve a borrow-from-zero problem, such as (a),

(a)	(b)	(c)
305	305	305
- 48	- 48	- 48
	267	167

he begins processing the units column by attempting to borrow from the tens column, and immediately reaches an impasse because zero can not be decremented. He's stuck so he does a repair. One repair is simply to skip the decrement operation. This leads ultimately to the solution shown in (b). If he uses this repair to the borrow-from-zero impasse throughout a whole subtraction test, he will be diagnosed as having the bug Stops-Borrow-At-Zero. Suppose he chooses a different repair, namely to relocate the decrement operation and do it instead on a nearby digit that is not zero, such as the nearest digit to the left in the top row, name-

ly the three. This repair results in the solution shown in (c); the three has been decremented twice, once for the (repaired) borrow originating in the units column, and once for the borrow originating in the (unchanged) tens column. If he always chooses this repair to the impasse, he will be diagnosed as having the bug Borrow-Across-Zero.

Bug Migration and Tinkering

Many bugs can be generated by this impasse-repair process (the exact number depends on how the sets of impasses and repairs are constrained—see section 6). However, we hardly expected to actually see this process in operation during the execution of one of the tests we gave. Instead, we expected that the impasse and repair had happened some time long ago and the resulting sequence of steps had become habitual, that is, a consistent, stable bug had been formed. Nonetheless, two predictions were advanced, namely that a few students would be found who would repair an impasse several different ways on a test (a phenomena we label "tinkering"), and secondly, that some students would switch from one repair to another between tests, a phenomenon labelled "bug migration" because it would show up as a consistent bug on the first test, and a consistent but different bug on the second test. The following three pseudo-tests illustrate these phenomena:

a.	102	9007	4015	702	2006	10012	8001
	39	6880	607	108	42	214	43
	73	2227	3408	604	2064	10898	8068
b.	102	9007	4015	702	2006	10012	8001
	39	6880	607	108	42	214	43
	3	1227	3408	504	1064	98	6068
c.	102	9007	4015	702	2006	10012	8001
	39	6880	607	108	42	214	43
	73	2227	3408	504	1064	10898	6068

Row (a) has been answered by a hypothetical student with the bug Stops-Borrow-At-Zero, row (b) by a student with the bug Borrow-Across-Zero, and row (c) by a tinkerer. All three hypothetical students can not borrow from zero. They differ only in how they repair the resulting impasses. The first repairs by skipping the decrement operation. The second repairs by decrementing the nearest non-zero digit to the left of the zero. The tinkerer reacts by sometimes doing one of these two repairs and sometimes the other. On problems 1, 2 and 6, the tinkerer skips the troublesome decrements, producing the same answers as the student in row (a) who has the bug Stops-Borrow-At-Zero. On problems 4, 5 and 7, the tinkerer refocuses leftward, leading to the same numbers as row (b). (Problem three does not involve borrowing from zero, so all three students answer it correctly.) The two Buggy students always repair the borrow-from-zero impasses consistently, while the tinkerer switches back and forth between two different repairs. In general, a tinkerer can switch among several repairs.

Although DEBUGGY is not able to tell when a student is tinkering, intensive hand analysis of 120 students revealed that 14 (12%) of the students were tinkering. So this prediction of Repair Theory was verified.

To observe bug migration, students are tested twice a short time apart. If a student answered as in (a) on the first test and (b) on the second test, then we would have a case of bug migration. The bug Stops-Borrow-At-Zero has "migrated" into the bug Borrow-Across-Zero. Only 67 students were tested in this two-test condition, and of these only 12 were diagnosed as having bugs on both tests. However, of these 12 students, two (17%) exhibited bug migration, verifying the predictions of the Repair Theory.

Repair Theory also predicts that students can tinker on one test and have a bug on the other. That is, a student can answer as in (a) on the first test and (c) on the second. He has the same impasse on both tests, but whereas he repairs consistently with a single repair on the first test, he uses two (or more) repairs on the second test. When the two-test data was examined by hand, four cases of this phenomenon were found.

A Summary of the Findings

A model of the student population has emerged from the data based on the notions of impasses, repairs, bugs and slips. Given just one test, the students who are making errors can be put into four categories in roughly the following proportions:

50% Knows the correct algorithm; errors due to slips alone.
30% Has a bug or a set of bugs (plus perhaps some slips as well).
10% Tinkering, using several repairs for one impasse (plus perhaps some bugs and slips)
10% Errors can not be analyzed.

These proportions vary with the grade level. The above proportions are for third graders tested late in the year. In general, the older the student population, the greater the proportion of students in the slips category and the smaller the proportion in the buggy category. In the early third grade, for example, students in the buggy category constitute over 50% of the sample.

The various kinds of errors are expected to have differing kinds of short-term stability. We expect slips for example to vary widely over two tests given a short time apart. There may be no slips on one test, and several on another. If there are slips on both tests, they are not expected to occur on the same problems. Impasses on the other hand are expected to remain in evidence across tests.

An impasse may show up as a bug on one test and on the next as a different but related bug, or as tinkering. What would be unexplained is a bug that was present on one test but absent on the other. These considerations prompt the following tabular summary of the percentage of students exhibiting the various kinds of stability:

4% No errors on either test
50% Stable correct procedure; changes due to slips alone
12% Stable bugs; changes due to slips alone
12% Stable impasses; changes due to repairs (often along with slips and stable bugs)
12% Appearing and disappearing bugs and/or impasses (with slips and stable bugs)
10% Errors on one or both tests can't be analyzed

The stability patterns of the students in the first four categories

(78%) conform to expectations, while the behavior of the students in the remaining two categories (22%) remain unexplained.

In overview, these data show that the older view of errors are due to either bugs (deterministic, systematic errors) or slips (careless mistakes) is incomplete. The impasses/repair notions contribute substantially to our ability to understand error-filled tests (in addition to their role as an explanation of the acquisition of bugs).

However, a significant proportion of the tests (10% of the static, one-test data, and 22% of the stability data) can not yet be analyzed even with these advances. Some of these students are in the undiagnosed category because the tests were simply not long enough to give the analysts a large enough sample of their behavior to disambiguate the various possible explanations of the errors. In other cases, species of behavior that have not yet been formalized were apparent. Some students appeared to "game" the test by struggling through the first part of it, then giving up and using some easily executed bug such as Smaller-From-Larger on the rest. Other sources of errors were rather uninteresting—there seemed to be several cases of cheating by looking at a neighbor's paper; in one case, a skipping ballpoint pen apparently caused a student to lose track of his procedure in the middle of several problems. In short, there will undoubtedly be some errors that have rather uninteresting causes and hence can properly be left unanalyzed in a formal descriptive study of errors. Our belief is that we have not quite reached that level of understanding yet. We guess that there remain some undiscovered, interesting mechanisms that will further our understanding of errors as much as the impasse/repair process did.

2. BACKGROUND AND MOTIVATIONS

This section and the following ones provide a more detailed description of the findings and how they were obtained. Special attention is given to evaluating the DEBUGGY's diagnostic ability and discussing how it could be used in practical educational set-

tings. To set the stage, a discussion of the history and motivations of the research is presented.

Many studies of systematic errors in arithmetic preceded the Buggy studies (Buswell, 1926; Brueckner, 1930; Brownell, 1941; Roberts, 1968; Lankford, 1972; Cox, 1975; Ashlock, 1976). In all these studies, systematic errors were thought of as incorrect or faulty algorithms with the same inputs as the correct arithmetic algorithm. In particular, systematic performance was assumed not to depend on the position of the test item on the page, the nature of the preceding item, fatigue, or anything else. This assumption is shared by the Buggy studies. To do otherwise would require orders of magnitude more data per subject so that the influence of these context variables could be studied. Since we share the belief of our predecessors that the influence is negligible, the context-free assumption has been built into the bug notation.

The Buggy studies differ from their predecessors in that a precise, formal notation for systematic errors is used. All the early studies relied upon informal, English descriptions of the observed systematic errors. However, even the most precise natural language descriptions are often flawed. For example, Cox used the description "Borrowed from the tens column when it was unnecessary" (Cox, 1975, pg. 155) to notate the following behavior (ibid., pg. 152):

$$\begin{array}{r} 37 \\ -\ 4 \\ \hline 23 \end{array} \qquad \begin{array}{r} 43 \\ -\ 1 \\ \hline 32 \end{array} \qquad \begin{array}{r} 85 \\ -\ 3 \\ \hline 72 \end{array}$$

From these problems, it is clear that the tens digit in the answer is off by one, but it is not clear that extra borrowing is the culprit. Rather, it could be that the student thinks that a subtraction is necessary in each column, so a one is subtracted in columns that have a blank in the bottom (this is the bug Sub-One-Over-Blank; see appendix 1). Cox's description is adding some assumptions to the naked observation of the behavior. Even if scratch marks are present and it is clear that the top row's tens digit has been

decremented, it is not clear whether the student decrements every column except the units column, or only those that are over blanks, or only the leftmost column. The natural language description is seriously incomplete. On the other hand, the syntax of the bug notation is such that a bug could not be written without taking a stand on when the extra borrows occur. This insistence on *precision* and *completeness* comes quite naturally with a formal notation, and is a distinguishing characteristic of the Buggy studies.

The idea that systematic errors can be represented as sets of bugs became the heart of a computer system named BUGGY. BUGGY had many facets. It could be used as a game to introduce student teachers to the idea of bugs and to develop their skill in discovering systematic errors in their students' work. BUGGY could also be used to analyze written tests worked by students in order to diagnose which bugs (if any) the students had. It was used to analyze addition and subtraction tests from over a thousand students. This early research is reported in (Brown & Burton, 1978).

Why Subtraction?

Based on our early experience with BUGGY, a strategic decision was made to investigate one procedural skill thoroughly rather than to cast about for examples of systematic errors in many domains at once. The procedural skill chosen for investigation was ordinary multi-digit subtraction. Its main advantage, from a psychological point of view, is that is a virtually meaningless procedure. Most elementary school students have only a dim conception of the underlying semantics of subtraction, which are rooted in the base-ten representation of numbers. When compared to the procedures they use to operate vending machines or play games, subtraction is as dry, formal and disconnected from everyday interests as the nonsense syllables used in early psychological investigations were different from real words. This isolation is the bane of teachers but a boon to the psychologist. It allows one to study a skill formally without bringing in a whole world's worth of associations.

The Goals of the Studies

Since BUGGY, the research developed in several directions. One direction was the development of Repair Theory, which was described above. In another direction, the technology for diagnosis was improved and extended by Richard Burton to become the DEBUGGY system. DEBUGGY is able to produce much more elaborate diagnoses than BUGGY. In addition, it can analyze a set of test items to measure its diagnositicity, in the sense discussed below. Burton also developed an interactive version of DEBUGGY (called IDEBUGGY), where the test items are generated by the system on the basis of the student's previous answers, thus allowing IDEBUGGY to converge on a diagnosis faster and with greater certainty. This line of research is reported in (Burton, 1981).

In support of the theoretical and technological lines of research, extensive empirical research has been necessary. This research involved collection and detailed analysis of a large number of systematic errors. This empirical investigation and its results are the topic of this paper.

The two major goals of the empirical studies were:

to pilot-test DEBUGGY: To evaluate DEBUGGY's diagnostic capabilities by using it on the kind of data that it would encounter if it were deployed as an diagnostic adjunct to the curriculum.

to test Repair Theory: To observe so many bugs that the database of bugs can be taken as an approximation of all the subtraction bugs that can occur. Repair Theory should be able to generate this set of bugs. Also, the studies were intended to check for the existence of tinkering and bug migration.

These are not the only uses that the data can be put to. Indeed, we have been asked so often for these data that it seems worthwhile to present it in as complete and neutral a fashion as possible. Thus, for example, data on the frequency of occurrence of bugs will be presented; although very little use can be made of this data at the present time vis a vis the above goals, the frequency data is

important in design of remediation tools for education. Rather than present the data statistically or in some other summarized form tailored to meet our goals, it is presented for the most part in tabular appendices. This allows other investigators to analyze the data as they see fit. The text is devoted to presenting the concepts involved in bug-level diagnosis, the methods used to collect and analyze the data, and commentary on the tables.

3. SUBJECTS AND METHODS

Two important development cycles began back in the BUGGY days and continued throughout the studies reported here. One was to extend DEBUGGY's diagnostic ability by augmenting its database of bugs, and the other was to improve the set of test items used to elicit errors. Before discussing the studies per se, it is best to describe these development cycles.

DEBUGGY cannot invent new bugs. Its inventiveness is limited to creating new *sets* of bugs from known bugs. (Creating a new set of bugs may sound trivial, but it is actually quite difficult in general since bugs can interact with each other in complex ways.) Discovering new bugs is very important for testing generative theories since it is only by having as complete a database of bugs as possible that the generative sufficiency of the theory can be ascertained. Since DEBUGGY does not invent new bugs, the method used to discover new bugs is use DEBUGGY as a filter to remove students whose behavior is adequately characterized by some set of existing bugs, leaving the human diagnosticians to concentrate on discovering any systematicity that lurks in what DEBUGGY considers unsystematic behavior. When even the barest hint of a new bug is uncovered by the experts, it is formalized and incorporated in DEBUGGY's database. That way, DEBUGGY will discover any subsequent occurrences of the bug, even when it occurs with other bugs, and even when it interacts in non-linear, complex ways with those bugs. So the first cycle consists of computer analysis, human analysis, and augmentation of the bug database.

Test Diagnosticity

The second cycle involves development of highly diagnostic tests. The set of problems given on a paper and pencil test is probably the most important determinant our ability to do diagnosis. One of the facets of the DEBUGGY system is the ability to measure the diagnosticity of a test. Given any subtraction test, it can calculate exactly how many problems a bug (or set of bugs) will miss on a test and exactly which pairs of bugs (or pairs of sets of bugs) can not be differentiated because they get exactly the same answers on the whole test. It can do this even when the bugs in a set interact in complex ways, such as producing a correct answer to a problem that each bug in the set would miss had it alone been applied to the procedure. Naturally, one wishes each potential diagnosis to miss at least one problem on the test so that it can be discovered. However, becaue students often make slips, one wants each bug (or set of bugs) to miss several problems on the test so that it can be discovered even if some of the problems it misses are not matched exactly by the student's answer due to his careless mistakes. Similarly, one wants redundancy in the problems that allow two different bugs (or set of bugs) to be differentiated from each other.

DEBUGGY's measurement of test diagnosticity is just as dependent on the bug database as its analysis of student errors. If a test passes the diagnosticity test, this only guarantees that any diagnosis that can be constructed from *known* bugs will be distinguishable. It makes no such guarantee about new bugs. Hence, the second development cycle has been to upgrade the paper and pencil tests as new bugs are discovered.

Given this mutual dependence of the data and techniques used to acquire it, it is worthwhile examining the cycles from the very beginning.

The Nicaraguan and Wellesley Studies

The original database of bugs was developed by Richard Burton, Kathy Larkin and John Seely Brown from a collection of 1325

Nicaraguan students' test results (this is the data reported in Brown & Burton, 1978). Two problems hampered them. One was that they did not have the actual test papers, but only each student's answers to the tests. Although DEBUGGY makes no use of the scratch marks that students use, human diagnosticians seem highly dependent on them. The second problem faced by the investigators was that the diagnosticities of the tests were not high. Despite these handicaps, Burton, Larkin and Brown were able to identify 43 bugs while maintaining confidence that these bugs were not the product of pure speculation. However, the most important effect of these bugs was to start the cycles rolling. Now better diagnostic tests could be developed, and a tighter filter on new bugs could be used.

The Nicaraguan students were fourth, fifth and sixth graders who had been taught subtraction by radio (Searle, Friend, & Suppes, 1976). The wide variety of bugs discovered in their work made us wonder what kinds of bugs we would find in American students who had received normal classroom instruction. A study was conducted with 288 students from Wellesley, Massachusetts (Haviland, 1979). Although the actual test papers were available, the lack of test diagnosticity continued to plague the investigators. A second problem arose in that very few students made any errors at all, probably because the students were sixth graders from an upper class community that could be expected to have a good school system. Consequently, only a few new bugs were discovered.

Capitalizing on our past experience, two extensive studies were planned and executed by Jamesine Friend and the present author with the assistance of Richard Burton, John Seely Brown, and Elizabeth Berg. Highly diagnostic tests developed with the aid of DEBUGGY were used. The test papers were available to the human diagnosticians. The students were mostly third and fourth graders from communities of average social and economic status, with school systems of average competency.

Several tests were developed with DEBUGGY's help for these studies. The first two tests were for interim use. They were replaced when enough new bugs had been discovered to make their

diagnosticity unacceptable. The tests that we ended up with had twenty items, and most items had three or four columns. Here is one of them:

647	885	83	8305	50	562	742	106	716	1564	6591
45	205	44	3	23	3	136	70	598	887	2697

311	1813	102	9007	4015	702	2006	10012	8001
214	215	39	6880	607	108	42	214	43

Every bug misses at least two problems, and almost all missed three or more. This redundancy allowed DEBUGGY to detect most bugs even in the presence of a large number of slips. The tests were also extremely diagnostic in that every possible diagnosis was distinguishable from the others by at least one problem. (For purposes of defining "all possible diagnoses," DEBUGGY was restricted to including at most two bugs in a diagnosis.)

The first study, called the Southbay study, was designed to accumulate a large quantity of new bugs and to pilot test DEBUGGY. Each student's test was carefully examined by at least one and often three human diagnosticians to corroborate or revise DEBUGGY's diagnosis. Also, a proportion of the students were retested several months later in order to study the long term stability of bugs. The second study, called the Short-term study, featured testing students twice, a short time apart (e.g. Monday and Wednesday). With this design, we could investigate the short term stability of bugs.

The Southbay Study: Subjects and Test Administration

During the 1979–1980 school year, 849 students were tested during the Southbay study. Two school districts in the southern San Francisco bay area agreed to participate in the study. Both school districts were a fairly heterogeneous composition of social classes. The majority of the children came from white, English speaking families, although there were some Japanese, Vietnamese, Korean and Filipino children. There were a number of children with Spanish surnames. Standardized test scores from the two dis-

tricts show that one is slightly above the national norm and the other is at about the 70th percentile.

Each district office sent notices of the study to their elementary school principals, asking for teachers to volunteer for the program. 33 teachers cooperated in the study. Although the teachers were self-selected, we have no reason to believe that their students were unrepresentative of their grade level's population.

Most of the classrooms had a single grade, except the Special Education classrooms, which included students in fourth, fifth and sixth grades. There were two combination classrooms, one with third and fourth grades, the other with fifth and sixth grades. The number of classrooms by grade level is:

Grade	*Number of classrooms*
Third	15
Fourth	11
Fifth	4
Special Education	3

Testing of the classrooms was spread over several months in order to ease the load on the diagnosticians. A few days before the tests were to be administered in a school, Friend went to the school to conduct a brief teacher training session to acquaint the teachers with the nature of the study, the tests, and the test administration procedures. The teachers were asked not to impose a time limit on the children, but to allow them as much time as they needed. (Teachers reported to us that the test typically took 15 minutes or less to complete, but there were some children who finished it in 5 minutes and others who took a half hour.) The teachers were also asked to instruct the students to respond to every item even if they were not sure they could do it.

Each classroom's file of student tests was analyzed by DEBUGGY. DEBUGGY's analysis was then checked by hand, always by at least one person (Friend) and frequently by another (Burton or VanLehn). The objective of the hand analysis was to discover new bugs and to check that DEBUGGY's diagnoses were reasonable. DEBUGGY's analyses, with any corrections that the diagnosticians felt were necessary, were compiled into a report for the teacher.

This report, together with a teacher's guide that explained how to use it, was mailed to the teacher.

The teacher's guide (Friend & Burton, 1980) contains a few pages for each bug, describing the bug, giving examples of the errors it produces, and sometimes making suggestions for remediation. A page of DEBUGGY-generated exercises is included that is suitable for remediating the bug in that the exercises cause the bug to exhibit symptoms.

The diagnostic reports and the guide were seen merely as the incentive for teachers to participate in the study, and not as a serious attempt at deploying bug-level diagnoses in the classroom. However, informal follow-up interviews with some of the teachers exposed some interesting problems that deploying this kind of detailed diagnostic information in the curriculum appears to face. These problems will be discussed in a later section of the paper.

To study long-term stability of bugs, some of the students in the Southbay study were reported to the teachers as needing retesting. Whether they actually were retested was up to the teacher, and ultimately only 154 students were retested. A student was recommended for retesting if either (1) he could not be diagnosed, or (2) his diagnosis included one of the less common bugs. In retrospect, it would perhaps have been better to ask the teachers at the outset to retest their whole class at a later date. However, it was felt at the time that they would not be receptive to this since recent concern with educational quality has mandated a very large number of tests, and it was felt that teachers would not want to include yet another test in their crowded schedules without a very compelling reason. However, 13 of the teachers chose to do the retesting recommended to them, and one of them asked to be allowed to retest her whole class since she found that easier than retesting only specific individuals in the class. It appears that we could have asked teachers to retest their whole class and gotten equally high or higher participation.

The Short-Term Study: Subjects and Test Administration

The second study to be reported here was designed to test the short term stability of bugs. Tests were administered two days

apart, generally on a Monday and the following Wednesday. Teacher's were asked not to give any instruction in subtraction between tests.

To control for the possibility that students would remember the previous test's answers rather than recalculating, four testing conditions were used. In condition 1, students received exactly the same test form both days. In condition 2, students received the same problems, but in a different order. In condition 3, the order of problems was the same but each problem was changed slightly. In condition 4, both the order and content of problems were changed, In short, the conditions differed slightly in the content and/or order of the items. However, it turned out the test scores did not improve significantly in any of the four conditions ($p>.10$ in all four cases, using the Mann-Whitney U test).

The subject acquisition, test administration and analysis procedures were as described for the Southbay study. Only three third grade classes elected to participate in the study since the school year was almost over. A total of 67 students completed both tests.

Data Analysis

Bugs are modifications of some correct procedure for the skill. To represent a systematic error, it is necessary to state which correct procedure is being modified by the given set of bugs, in the case when more than one correct procedure is possible. There are several different subtraction procedures taught in different parts of the world. Moreover, even in the United States, several variations of the "standard" procedure are taught, differing particularly with regard to the use of scratch marks. However, since our largest sample populations were drawn from school systems that taught similar subtraction procedures, we have been able to represent the subtraction data using just one correct procedure. Nonetheless, the database and the diagnostic programs are designed to handle multiple alternative correct procedures.

DEBUGGY analyzes a test in three stages. First, it grades the test. The students who make no errors on the test are placed in the *Correct* category. Next, the set of bugs that fits the errors the

best is found. Lastly, DEBUGGY decides whether the fit is good enough to put the student in the *Buggy* category. If not, then it decides whether there are few enough errors that the student can be put in the *Slips* category. If there are too many errors, then the student is placed in the *Undiagnosed* category.

The criteria used to assign subjects to the Buggy, Slips and Undiagnosed categories are complex and ad hoc. They were designed to mimic the intuitions of human diagnosticians. The following is a rough characterization of it (see Brown & Burton, 1978, for a complete treatment).

Rules for assigning diagnostic category

1. If the diagnosis predicts all the student's answers, both correct and incorrect, then he is assigned to the *Buggy* category.
2. Also, a student is assigned to the *Buggy* category if (i) the diagnosis makes more true predictions than false predictions and (ii) makes "enough" true predictions about wrong answers. The latter criterion is meant to prevent a student from being diagnosed as having bugs when in fact his wrong answers are equally well predicted by the hypothesis that he is performing the correct algorithm with an overlay of slips. Since the Buggy and Slips hypotheses agree whenever the bugs predict a correct answer, it is the problems where the bugs predicts wrong answers that split these two hypotheses. This criterion of "enough" true predictions of wrong answers is implemented by the following three conditions:
   a. Of the answers predicted to be wrong by the bugs, 75% or more are indeed the answers given by the student (i.e. are true predictions).
   b. Of the answers predicted to be wrong by the bugs, 50% or more are true predictions, and there is only one bug in the set of bugs, and more than half the wrong answers given by the student are predicted by the bug.
   c. Of the answers predicted to be wrong by the bugs, there are more true predictions than there are false predictions of all kinds, and there are at least two true predictions.

3. If a student is not assigned to the Buggy category by rules 1 or 2, and he has gotten 90% of the problems correct, then he is assigned to the *Slips* category.
4. Otherwise, he is assigned to the *Unsystematic* category.

The actual algorithm used by DEBUGGY is more complex than this. DEBUGGY has an ability to model certain kinds of slips which it uses to decide which of two competing sets of bugs is the best fitting diagnosis (Burton, 1981). Slips are used to temper the decisions of rule 2, although they do not in fact play a role in rule 3 despite the fact that rule 3 decides between the Slips category and the Undiagnosed category. The emphasis in the DEBUGGY research has been on modeling bugs, not slips.

As mentioned above, the database of bugs grew during the Southbay and Short-term studies. Since the contents of the database effects DEBUGGY's diagnosis, all the tests from both studies were reanalyzed by DEBUGGY at the conclusions of these studies, after the newly discovered bugs had been installed in the database. The data from the reanalysis are the ones reported in this paper.

4. EVALUATING DEBUGGY'S DIAGNOSTIC EXPERTISE

One of the main goals was to determine whether DEBUGGY could be relied upon to diagnose bugs as accurately as expert human diagnosticians. There were two reasons to believe that it might not be reliable. First, DEBUGGY was not given the scratch marks that some students use to help them do borrowing. Our experience has been that the scratch marks are invaluable to human diagnosticians. It was not at all clear that DEBUGGY could succeed given just the answers. The second area of uncertainty in DEBUGGY's design was the heuristics used to split Buggy students from Undiagnosed students, and to determine which of two diagnoses is a better predictor of the students answers. These can be difficult decisions for human diagnosticians, let alone DEBUGGY.

DEBUGGY Versus the Experts

Table 4.1 of appendix 4 (as shown below) compares DEBUGGY's categorization of students with the human diagnosticians' using the Southbay data. This is a three-by-three table, by diagnostic category. If there were perfect agreement, all the off-diagonal entries would be zero. Although they are not zero, in most cases they are fairly small, with two exceptions. The experts found Buggy diagnoses for 30 (13%) of the students that DEBUGGY consided Undiagnosed. These cases are due to the fact that the human diagnosticians could see the scratch marks, and hence could be more sure of the systematicity of the errors which DEBUGGY found but rejected as being not quite systematic. The other large off-diagonal entry represents 59 cases of the expert deciding that there was not enough evidence to rate a diagnoses as Buggy even though DEBUGGY thought so (these 59 cases represent 20% of DEBUGGY's Buggy diagnoses). Again, we believe these represent inherent differences between DEBUGGY and the human diagnosticians' judgments caused by the latter's access to the scratch marks.

	Experts' Diagnosis			
DEBUGGY's Diagnosis	Slips	Buggy	Undiagnosed	totals
Slips	148 (95%)	4 (3%)	3 (2%)	155 (100%)
Buggy	3 (1%)	233 (79%)	59 (20%)	295 (100%)
Undiagnosed	18 (8%)	30 (13%)	188 (80%)	236 (100%)
totals	169	267	250	686

However, it appears that DEBUGGY's classification algorithm is a good compromise in that about the same percentage of students are mis-classified into the Buggy and Undiagnosed categories. That is, it is biased neither towards systematicity nor against it. This leads to the conclusion that a nearly optimal setting of the parameters of the classification algorithm has been reached. Improved classification would require giving DEBUGGY access to the scratch marks and/or a more complete model of slips.

Another way that DEBUGGY could be inaccurate is by choosing the wrong diagnosis for a Buggy student when several diagnoses were in close competition. Table 4.2 of appendix 4 shows how the diagnoses given by DEBUGGY were corrected by the human diagnosticians. In almost every case (220 out of 233, or 94%), there was substantial agreement between the experts' diagnoses and DEBUGGY's, and in many cases (193, or 83%) their diagnoses were identical. This shows that DEBUGGY was excellent at choosing among competing hypotheses. In short, DEBUGGY and the experts agree on *what* a systematic error consists of, but sometimes disagree on *how systematic* the error is.

DEBUGGY versus the Experts with Matched-Item Tests

In an effort to probe DEBUGGY's expertise more deeply, we again compared it to expert judgments but handicapped it. Using the Short-term study, the experts were allowed to use both tests of a student in performing their diagnosis while DEBUGGY analyzed each test individually. Since the test forms were matched item-by-item, the experts could see the same item answered twice, and thus more easily come to an intuitive assessment of whether an error was due to a slip or a bug. In order to have a basis for comparison, two experts analyzed the tests independently. The differences between their diagnoses could be used as a baseline for the differences between DEBUGGY's diagnoses and expert ones. The results are summarized in appendix 4.

Roughly speaking, *the experts agreed more with DEBUGGY than they did with each other.* The only substantive difference was that the experts put more students in the Slips category than DEBUGGY, and fewer in the Undiagnosed. This can be attributed to their access to matched test items, which presumably allowed them to more confidently assert that non-buggy errors came from slips rather than from some unknown cause.

To sum up, it appears that DEBUGGY using just the answers is as good as the expert diagnosticians using the answers plus scratch marks, plus matched-item tests. The decision to allow it no access to the scratch marks appears to have been, on balance, a

good one. It makes the system practical because much less data (namely the answers alone) needs to be entered off the test sheets. The lack of this information does not appear to hurt its ability to formulate a bug-level diagnosis at all, although it does appear to hurt its ability to assess whether a diagnosis deserves the Buggy classification (the difference in judgment could also result from DEBUGGY's incomplete treatment of slips).

5. THE UNDIAGNOSED CATEGORY

When all the tests in both the Southbay and Short-term studies are considered, a large percentage were assigned to the Undiagnosed category. Similar figures occurred in the Nicaraguan study, as shown below.

Category	*Nicaraguan*	*Southbay & Short-term*
Correct	37	112
Buggy	505 (39%)	417 (40%)
Slips	116 (9%)	223 (22%)
Undiagnosed	667 (52%)	386 (37%)
totals	1325	1138

The figures in parentheses are the percentages of the students who made errors that were assigned to each category. In both studies, a substantial proportion of the population could not be diagnosed. What is causing the errors in these students' performances?

The "Math Disabilities" Hypothesis

One conjecture is that the Undiagnosed students are not well practiced in subtraction, math phobic, or "stupid." If test scores are taken as a measure of such math disabilities, then there is no evidence for this explanation. The scores of the Buggy and Undiagnosed students are almost identical. Figure 1 is a graph of the distributions of test scores. One line shows the distribution for students in the Buggy category and the other shows students in

the Undiagnosed category. These distributions are very similar, except in two places. One is a large peak in the Buggy distribution at a score of three answers correct. This is due to the fact that the test had just three problems that did not require borrowing (only form 2 is covered by the graph, since it was the most commonly

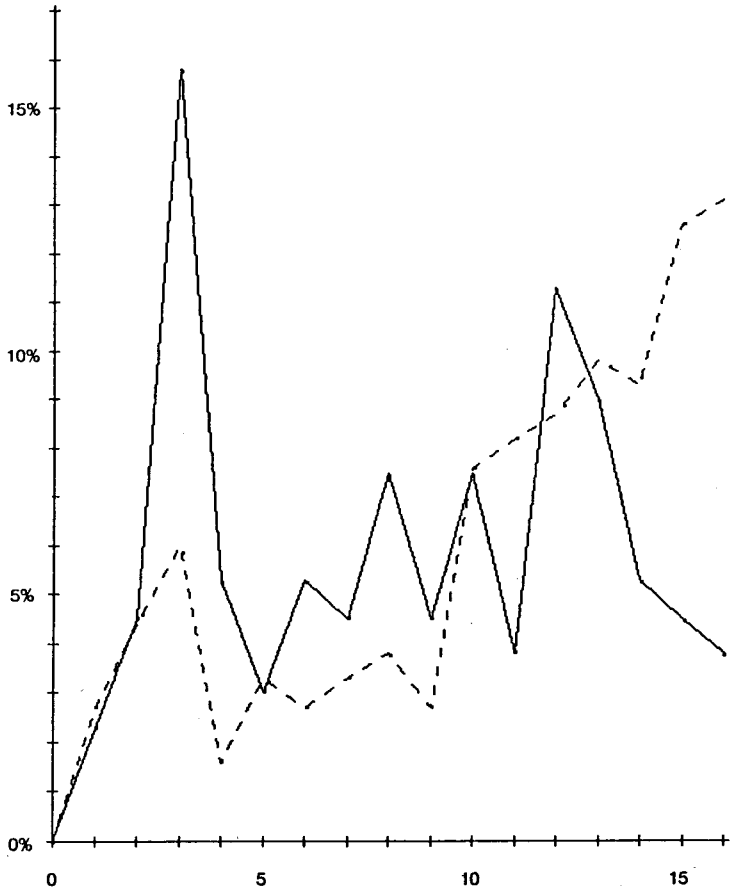

Figure 1. The distribution of raw test scores is plotted for the Buggy category (solid line) and the Undiagnosed category (dashed line). The x-axis is the number of problems answered correctly. The y-axis is the percentage of students in the category who obtained that score. The test had 19 problems. Students who missed only one or two problems are not shown.

used form). This peak is almost entirely due to the bug Smaller-From-Larger, of which there were 106 occurrences. This bug misses all but three problems on the test.

The other peak is in the Undiagnosed distribution, at the 16-answer correct mark. We think this peak is due to students making careless mistakes, i.e. slips. This peak is the leading edge of a much larger slips peak that would continue upward from 16 answers correct to 18 answers correct (the test has 19 problems), except that DEBUGGY's classification algorithm places undiagnosable students who get three or more problems wrong in the Undiagnosed category rather than the Slips one. This peak probably represents students who should have been put in the Slips category (more on this conjecture in a moment). These two peaks aside, the distributions are very similar. When the points corresponding to these two peaks (namely the points at 3, 4, 15 and 16) are left out, the correlation between the two distributions is .57 ($p<.05$).

Reanalysis With Matched-Item Tests

This similarity in the test score distributions sparked a more intensive hand analysis of the tests of the students in the Undiagnosed category. We already knew that it was difficult to do better than DEBUGGY given just one test (cf. the preceding section). However, it was discovered during the Short-term study that much of the uncertainties in hand diagnosis were removed by having two executions of the same test available, even though the tests were taken several days apart. Since each problem was answered twice, it was easier to separate slips from repeated, conceptually based errors. By using *pairs* of tests, informative hand diagnosis of the Undiagnosed class proved possible.

Forty-four of the students in the Short term study were Undiagnosed on one or the other of these tests. These were analyzed by the author with an eye to finding what proportion of them could be accounted for with slips and tinkering.

The major source of unsystematic performance appears to be facts errors and other slips (Norman, 1981). Slips are unintended actions. The person had the *competence* to choose and execute a

certain action but during *performance,* he did not. In addition to errors in the basic subtraction facts, one often sees cases where

- a decrement is forgotten,
- a digit is incremented instead of decremented,
- the last column of a problem is not processed,
- an extra, inappropriate borrow is performed,
- a digit is decremented by two when the bottom digit in its column is a two, and many others.

Forty-five percent of the students who were placed in the Undiagnosed category on the basis of a single test were analyzed as performing the correct algorithm with several slips. (These performances are particularly easy to pick out since slips do not in general appear on the same problems on both tests. That is, if errors on the first test are not matched by errors on the same problems on the second test, then it is likely that the student's errors are due to slips. Perhaps this observation could be used in constructing aids for this style of two-test diagnosis, or for single-test diagnosis using redundant items.)

Slips occur on top of bugs as well as the correct procedure. Here, their occurrence complicates and sometimes prevents DEBUGGY from reaching a diagnosis. That is, the underlying competence is well modelled by bugs, but slips create enough noise that diagnosis is blocked. Our tests were designed so that almost all bugs get at least three problems wrong, but there are many that miss six or fewer problems. When slips modifies a significant percentage of the answers that the bug will cause the student to get wrong, which for some bugs means the student need slip on only say, three of the six problems it would miss, then there will probably not be enough evidence (either for DEBUGGY or a human expert) to diagnoses the performances as Buggy. However, the two-test samples allows slips to be uncovered even when they are on top of bugs. It was found that 14% of the Undiagnosed students had bugs that were covered by slips.

A third conjecture is that much unsystematic error is due to tinkering. 18% of the Undiagnosed students in the two-test sample were found to be tinkering.

The remaining 23% of the two-test sample exhibited a pattern of errors that was too stable to be analyzed as due to slips, but could not be accounted for as tinkering or slip-ridden bugs. The expert simply did not have enough data on each student's performance to reach a definitive diagnosis, even with the two tests. The source of the errors in their performance remains a mystery.

The following table summarizes the findings given above:

45%	Slips
14%	Bugs
18%	Tinkering
23%	Unknown
100%	total

The main reason that so many students could not be diagnosed by DEBUGGY is that they were making too many slips. However, there was also some tinkering present among the Undiagnosed students, as well as a non-trivial amount of truly puzzling behavior. For DEBUGGY to do better, it probably would have to have much more redundant test items so that it could locate slips in the way that the expert did, namely, by comparing a student's performance on identical or nearly identical problems.

6. REPAIR THEORY'S PREDICTIONS

The main claim of Repair Theory is that it can generate all the observed bugs. In fact, it can not do this, but for a good reason. Repair Theory postulates a particular process, namely of reaching an impasse then patching it with some highly local problem solving. However, this process is in effect parameterized by the set of impasses and repairs it is equipped with. The more impasses and repairs, the more bugs it can generate. If enough impasses and repairs were used, all the bugs could be generated. Indeed, we have collected and precisely described an ad hoc but empirically sufficient set of impasses and repairs. Some of these are mentioned in the appendix 5, and others are described in the "periodic table" section of (Brown & VanLehn, 1980).

However, we would rather have *principled* sets of impasses and repairs, ones that are generated from some constraints imposed by the task domain or the learning sequence the student is in the midst of. This would, for example, allow one to predict the bugs that would occur in a new procedural skill before any data were collected. One highly principled technique for generation of impasses is used in (Brown & VanLehn, 1980). But it generates only a few of the needed impasses, and consequently only 18 of the 77 observed bugs (the observed bugs are listed in appendix 3 of this paper). Other principles for generating the sets of impasses and repairs are being investigated that will hopefully converge on generating the ad hoc, observation-based sets used in this paper.

Using Co-Occurrence Frequencies as Evidence

In section 3.6 of Brown & VanLehn (1980), the authors consider a new technique for generating impasses based on overgeneralization. The argument they present is interesting for the way it uses the data. It makes a prediction about the frequency of bugs co-occurring with other bugs. That prediction happened to be correct in the Southbay data, and the argument and its prediction is worth recapitulating here for the way it uses the frequency data.

There are several bugs that could be generated by various repairs if only an impasse occurred whenever the student borrowed *into* a zero. That is, when the student comes to process a column with zero on top and a non-zero number on the bottom, instead of borrowing he does some repair. The bugs Diff-0-N = N and Diff-0-N = 0 are two such bugs. (See appendix 1 for a description of these bugs.) Brown and VanLehn suggest that these bugs occur because the student does not know how to borrow *from* zero. When such a student is asked to do a borrow from zero, he violates the precondition that a zero can not be decremented. He then overgeneralizes the newly discovered prohibition "you can't borrow from zero" to include "you can't borrow *into* zero." So in addition to hitting an impasse whenever he borrows from zero, the student's overgeneralized precondition blocks borrowing in 0-N columns, causing an impasse which leads to the two 0-N bugs. Crucially, this explana-

tion predicts that when the two 0-N bugs occur in compound diagnoses, another bug in the diagnosis will involve an inability to do borrowing across zero (i.e. recursive borrowing). Close examination of appendix 2 verifies this prediction. The results are summarized in the following two tables. The first counts the number of times the two Diff-0-N bugs occurred with recursive borrow bugs, and the second counts the number of times they occurred with other kinds of bugs. As predicted, there are more occurrences with recursive borrow bugs than with other bugs.

Occurrences	*with Recursive Borrow Bug (plus others, in some cases)*
15	Borrow-Across-Zero
17	Stops-Borrow-At-Zero
3	Smaller-From-Larger-Instead-of-Borrow-From-Zero
1	Borrow-From-Bottom-Instead-of-Zero
36	total

Occurrences	*with Non-Recursive Borrow Bug (plus others, in some cases)*
7	Borrow-No-Decrement
4	Smaller-From-Larger
1	Diff-N-0=0
1	Blank-Instead of-Borrow
1	Borrow-From-Zero
14	total

This kind of argumentation using the frequency of co-occurrence is interesting, and one we would like to be able to do more of. Unfortunately, the sample is too small to see such effects except with extremely common bugs such as Diff-0-N=N. Most bugs occur less than a half dozen times, a quantity so small that differential co-occurrence frequencies are often not statistically meaningful.

Reifying The Impasse-Repair Process

Repair Theory was developed well before the Short-term study was conducted. At that time, it was unknown whether the impasse-repair process was actually a process the students went through during the course of solving some arithmetic problem, or whether it could not be reified and had to remain an essentially mathematical model, that is, a formal specification of a set of

bugs, implemented as a function whose internal chronology had nothing to do with students' real, temporal behaviors on arithmetic tests. A few students had been found who appeared to be tinkering, but the single test data was not redundant enough to eliminate the possibility that what appeared to be shifting among repairs was in fact just slips. The Short-term tests were critical in finding out whether the impasse-repair process, or rather its chronology, could be reified. Showing that the sequence of states traversed by the model while generating a bug could in some fashion be observed in the real, temporal behavior of a student who developed that bug would justifying claiming "psychological reality" at a much finer grain size, namely the grain size of the process itself rather than just its output.

The data from the Short-term study was presented appendix 5. It is analyzed two ways, by DEBUGGY, which can not analyze the data in terms of tinkering, impasses and repairs, and by an expert who can. When the data are analyzed by DEBUGGY, there is a moderate amount of instability:

Stable procedure; changes due to slips alone		
21	(31%)	Stable correct algorithm
2	(3%)	Stable bugs
23	(24%)	subtotal
Stable impasses (plus perhaps some stable bugs); changes due to differing repairs		
2	(3%)	Bug migration
Unstable procedure; changes unexplained		
11	(16%)	Correct procedure changes to Undiagnosed
19	(28%)	Bugs appear or disappear (plus perhaps some stable bugs)
30	(45%)	subtotal
Unanalyzable		
12	(18%)	Undiagnosed both tests
67	(100%)	total

Although two cases of bug migration were found, these were embedded in so much unexplained shifting among procedures that it is uncertain whether the bug migration cases were caused by the mechanisms that Repair Theory postulated, or whether they were due to whatever it was that was causing so many other bugs to

appear and disappear. To resolve this uncertainty, the 67 students were thoroughly analyzed by hand. The results came out somewhat differently:

Stable procedure; changes due to slips alone		
36	(54%)	Stable correct algorithm
3	(4%)	Stable bugs
39	(58%)	subtotal
Stable impasses (plus perhaps some stable bugs); changes due to differing repairs		
0	(0%)	Bug migration
6	(9%)	Tinkering on one test, bug on the other
6	(9%)	Tinkering on both tests
12	(18%)	subtotal
Unstable procedure; changes unexplained		
1	(1%)	Correct procedure changes to Undiagnosed
5	(7%)	Bugs appear or disappear (plus perhaps some stable bugs)
7	(10%)	Impasses appear or disappear (plus perhaps some stable bugs)
13	(19%)	subtotal
Unanalyzable		
4	(6%)	Undiagnosed both tests
67	(100%)	total

There is still a significant amount of unexplained shifting among procedures, but it is comparable in size to the amount of shifting explained by Repair Theory. A large amount of tinkering was found. Hence, it can be concluded that reifying the impasses-repair process is justified.

7. PROSPECTS FOR DEPLOYING BUG-LEVEL DIAGNOSIS

As a user, the author found Burton's DEBUGGY program truly amazing. It performs much better than I can given the same input; in fact, none of the diagnosticians on the project feel they can match DEBUGGY's ability to do diagnosis when given only the answers and not the scratch marks. Despite the fact that it is a very large, very complicated program that uses Artificial Intelligence

technology, it is extremely reliable. Typically, it is left running unattended overnight.

DEBUGGY was built as a research vehicle, rather than a practical way to deploy diagnostic technology in the educational system. However, it has succeeded so well that it is worth examining the possibility of making it a practical service to teachers.

Diagnosing the Undiagnosed

One feature is crucially important to add to DEBUGGY. In a third of the cases, namely the Undiagnosed students, DEBUGGY has *nothing informative to tell the teacher about the student.* It is only when the student is Buggy that the teacher receives information about the student that can be used for placement or remediation. In the near term, bug-level diagnosis should be combined with standard criterion or subskill-based diagnosis (e.g. Friend, 1981) for the Undiagnosed students.

Unfortunately, the tests used in this study are totally inappropriate for subskill-based diagnosis. Subskill-based tests require test items which isolate each subskill from the others. Thus, if the ability to perform borrows in adjacent columns is considered a subskill, a number of items must require that subskill but not others. This allows the missing subskill to be identified by the binary pattern of right and wrong answers. Since almost every test item on our tests requires using almost every subtraction subskill to solve it (which is exactly what one wants in an optimal test for detecting bugs), the pattern of right and wrong answers reveals next to nothing about the presence and absence of subskills. Burton (1981) suggests it would be possible to construct a test that both isolated subskills while remaining highly diagnostic for bugs. How many test items such a test would have to have remains to be seen.

Moving to New Procedural Skills: The "New Bugs" Problem

The central limitation on DEBUGGY is its inability to invent new bugs. Although it can invent new systematic errors, so to speak, by

forming never-before-seen sets of bugs, some of which have complex non-linear interactions, this is not the same as inventing a new bug. DEBUGGY helps researchers find new bugs by filtering out students with known bugs, thus allowing human diagnosticians to focus on the apparently unsystematic students.

However, this is not a strong filter since a third of the students are classified as Undiagnosed. This leaves the human diagnosticians with a large number of test papers to analyze for new bugs. Consequently, it requires a great deal of effort to build a database of bugs the size of the one the DEBUGGY now has. Lumping together the work of the six diagnosticians that have worked on the project over the last several years, we estimate that *four or five thousand hours* were spent doing hand diagnosis. Have we found all the bugs, or are another several person-years necessary to guarantee accurate diagnosis as well as insure that the database of bugs is complete enough to adequately test generative theories of bugs?

The following table is a rough summary of the rate at which new bugs were discovered. It should be taken as an estimate only since we did not record the dates that bugs were entered in the database.

	added	total
Nicaraguan Study (December 1977)	45	45
Wellesley Study (October 1979)	15	60
First data from Southbay Study (November 1979)	30	90
Further data from Southbay Study (April 1980)	15	105
Last data from Southbay Study (June 1980)	10	115

The original Nicaraguan study initialized the database with 45 bugs. The Wellesley study a year later added only 15 bugs because the test was not very diagnostic and the students were too competent. The Southbay study was extremely productive, adding 55 new bugs to the database. Most of these came at the beginning of the study. One could conclude from the fact that the rate of new bugs being entered decreases that the database was converging. However, it appears to be converging rather slowly. In short, we probably have not found all the bugs, and it could take substantial effort to do so.

This means that additional methods should perhaps be sought to reject generative theories of bugs when they appear to be generating "too many" bugs that don't occur in the database. That is, since the database can not be assumed to be complete, the fact that a particular theory generates a bug or even many bugs that are not in the database is not sufficient cause to reject that theory. A method based on absurd or "star" bugs is used to reject theories that overgenerate in (Brown & VanLehn, 1980). Its merits and shortcomings are discussed there.

Although having a nearly complete database of bugs is very important for testing generative theories of bugs, it may be that it is not very important for diagnostic applications in education if the newly discovered bugs turn out to be very rare. To show whether the new bugs were indeed rare, the bugs in the table of bug occurrence frequencies in appendix 3 have been annotated to indicate whether they were in the database at the beginning of the Southbay study. Although most of the new bugs occurred only once or twice, some were not at all rare. One of the very last bugs to be entered, 0-N = N-Except-After-Borrow, turned out to be the sixth most frequently occurring bug upon reanalysis of the Southbay data. So, the new bugs were not uniformly rare. To get a diagnostic instrument of even medium accuracy, discovery of new bugs must be pursued vigorously through several iterations of the test development cycle.

One possible way to reduce the need for bug discovery is to pre-load the database by giving the imagination free rein to invent as many conceivable bugs as possible. This was not done in our studies because increasing the size of the database increases the number of hypotheses considered by DEBUGGY during analysis; on the computers used originally, this caused DEBUGGY to run out of memory. We now run DEBUGGY on computers with much larger address spaces, so loading the database with speculative bugs is feasible.

However, our experience during the Southbay studies indicates that the imagination, even of experienced diagnosticians, is not powerful enough to invent some of the bugs that children invent. For example, take the bug Decrement-Multiple-Zeros-By-Number-

To-Right. When borrowing across several zeros, this bug changes the rightmost one to 9, then next one to 8, then next to 7, and so on. It will answer problems in this fashion:

$$\begin{array}{r} 30005 \\ -\quad 7 \\ \hline 27898 \end{array}$$

This bug seemed incredible to us when we first noticed it. It was certainly not a bug that we would have thought of before we saw it. But having seen this bug, it seemed plausible that its symmetric partner, the bug Decrement-Multiple-Zeros-By-Number-To-Left, might exist. Both bugs were added to the database. DEBUGGY found four cases of one and three cases of the other. This anecdote illustrates how data and the imagination both are necessary to fill out the database.

Repair Theory offers a different approach to pre-loading a database with bugs. Essentially, it provides a way of transferring the imagination and experience expended on subtraction to other domains. The two bugs mentioned just above, for example, could be generated by Repair Theory when it is equipped with certain repairs. When the base skill of the theory is changed from subtraction, to for example, multiplication, the analogs of these two bugs would be generated. In this fashion, the database could be preloaded by Repair Theory. Those bugs in combination with those suggested by the imagination of experts might provide a fairly good initial set. What proportion of the ultimate set of observed bugs these would turn out to be is at this point a matter of conjecture.

Micro-Computer DEBUGGY

It has often been suggested that a quick-and-dirty version of DEBUGGY be built for a micro-computer so that it could be used by the teacher in the classroom or while grading papers at home. By examining the data, we can guess at how much diagnostic ability one would lose in such an implementation.

Appendix 2 shows that only a few sets of bugs occurred more than once. Of the 157 distinct sets of bugs, only 50 occurred more than once. The other 107 diagnoses occurred exactly once. One might say that there is a long "tail" to the frequency distribution. Of course, the bulk of the students fell under the non-tail part of the distribution; the 50 multiply-occurring sets of bugs accounted for 310 (74%) of the 417 Buggy students.

One inference that can be drawn is that a diagnostic instrument that can only diagnose high and medium frequency systematic errors will fail to diagnose roughly a quarter of the students that DEBUGGY can diagnose. That is, if it is unable to diagnose the students in the long tail, then it will miss 107 (or 26%) of the 417 students that DEBUGGY diagnosed. Moreover, it is these rare bugs that most need to be diagnosed since these are the ones teachers can't detect thereby causing the student to be perceived as "random," which in turn could perhaps lead to the student's developing a permanent fear of mathematics.

This lengthy tail was also found in the Nicaraguan data. However, the fall-off in frequency was not as rapid as it is in the Southbay data, and the tail was not as long. We believe the explanation for this lies in the higher diagnosticity of the tests used in the Southbay study, and in improvements to DEBUGGY and its database. Because the Nicaraguan tests were not as diagnostic, students that would receive distinct diagnoses using the Southbay tests actually received the same diagnosis. Similarly, DEBUGGY's increased prowess at discriminating bugs (due mostly to the larger variety of bugs in its database, but also to its ability to use up to four bugs in a diagnosis instead of only two) tended to split students that it once would have given the same diagnosis to into separate, although perhaps overlapping, diagnoses. In short, when more accurate diagnosis is performed, one finds that there are more distinct diagnoses. Consequently, the frequency of any given set of bugs tends to decline towards one.

This means that the worse the diagnostic tests and analyzer are, the shorter the tail on the distribution and the higher the frequency of individual diagnoses. Hence, a quick-and-dirty high-frequency-only diagnostic program would justify itself falsely. Suppose a

diagnostic instrument can only detect Smaller-From-Larger and a few of the other high-frequency bugs. Then, students who in fact have one of the less common bugs that is similar to Smaller-From-Larger, such as Smaller-From-Larger-Insteadof-Borrow-From-Zero or Smaller-From-Larger-When-Borrowed or Borrow-Once-Then-Smaller-From-Larger, would be diagnosed as having Smaller-From-Larger. The instrument's assumption that most students have one of the high frequency diagnoses would be unfairly vindicated.

More implications for a quick-and-dirty diagnostic instrument follow from the large variety of bugs that occur in the high-mid frequency diagnoses in appendix 2. Considering just the 50 diagnoses (i.e. sets of bugs) that occur more than once, one finds 32 different bugs. These figures become meaningful when compared to the 77 bugs that occurred overall. A large fraction of the bugs (32 of 77, or 42%) is necessary just to get the high frequency sets of bugs. This has implications for the design of a quick-and-dirty diagnostic instrument. It says that such a program would not benefit from including an ability to do bug compounding, that is to cope with the non-linear interactions involved in forming sets of bugs dynamically. It would still need to store 32 bugs just to get the high frequency diagnoses; it would be cheaper in terms of space and time just to store the 50 high frequency *sets* of bugs. Of course, this is incredibly risky in that slight perturbations in the data presented here would boost some diagnoses over the single-occurrence line, and others under it.

Using DEBUGGY's Diagnoses

We do not know whether providing bug-level diagnoses to the teachers will help them educate their students. That effect depends crucially on what they do with the information. However, simply handing them the information does not appear to be effective. The teachers who participated in our studies were given a teacher's manual (Friend & Burton, 1981) that described each bug in detail along with, for some bugs, common sense suggestions for remediation. A sheet of exercises, suitable for photocopying and

designed to exhibit the bug, was included in the manual for each bug. In informal meetings with the teachers after they had had their class's diagnostic report and the manual for some time, we found that in general they did not understand the concept of a bug, and consequently did not use the information we gave them as well as they could have, if at all.

This is consistent with our experience in using the Buggy game, a micro-computer game designed to teach the concept of a bug to student teachers (Brown & Burton, 1978). Players often commented after playing the game that it they learned a great deal from it, that it changed the way they thought of students' errors (Brown & Burton, 1978, appendix 1). In short, the bug concept is sufficiently foreign to teachers that a certain amount of teacher training, perhaps as little as a couple of hours playing the Buggy game, is necessary before they can begin to use the diagnostic information that DEBUGGY provides.

Longitudinal Effects

Before one can assess the effects of bug-based remediation, it is necessary to know what happens naturally to bugs. In particular, do they persist throughout school, or do current educational practices eventually remediate them?

A relationship was found between bugs and grade level. The following table shows, for each grade, how many subjects were assigned to each diagnostic category. (The students in remedial and special education classes, of which there were 30, have not been included in this table. Eight sixth graders are included in the fifth grade since that was the grade level at which they were receiving instruction.)

Grade	Correct	Slips	Buggy	Undiagnosed	totals
Third	32 (6%)	64 (13%)	237 (49%)	148 (30%)	481 (100%)
Fourth	48 (15%)	77 (24%)	87 (27%)	104 (32%)	316 (100%)
Fifth	19 (19%)	41 (41%)	13 (13%)	25 (25%)	98 (100%)
totals	99 (11%)	182 (20%)	337 (38%)	277 (31%)	895 (100%)

As one would expect, the proportion of the sample assigned to the Correct and Slips categories rises as the grade level increased from three to five. However, it was somewhat surprising to find that the proportion assigned to the Undiagnosed category remained relatively constant across grade levels. It appears that the systematic errors of the younger students apparently become the systematic correct algorithm of the older student, while current teaching practices seem to leave the Undiagnosed errors unremediated. Should this trend in fact underlie the variation of this table, it would imply that remediation should be focused on the Undiagnosed student rather than the Buggy one. However, the actual situation is much more complicated than that, as the long term stability data shows.

A long-term stability study was included as part of the Southbay study. By long-term, we mean that the tests were given at least two months apart, but still within the same school year. Appendix 6 presents the long-term data. The basic finding is that about half the Buggy students became Undiagnosed and about half the Undiagnosed students become Buggy. This casts doubt on the hypothesis that current teaching practices are succeeding in correcting systematic errors but not in correcting unsystematic errors. However, the instability in the short term makes these category figures somewhat suspect. It could be that these "long term instability" figures are so buried in "short term instability" that they do not in fact reveal the underlying trends. So, we will have to leave the question of the efficacy of remediation on systematic error until a better understanding of short-term stability is developed.

Although the changes in diagnostic category are a mystery, the changes in the bugs of the 34 students who were Buggy on both tests follow the patterns that one would expect intuitively. There were 17 students who had the same or overlapping bug diagnoses on both tests. Appendix 6 shows their diagnoses. The bugs in common on both tests show long-term stability. Not surprisingly, the most frequently stable bugs are Smaller-From-Larger, Stops-Borrow-At-Zero and Borrow-Across-Zero, which are exactly the three most common bugs.

There were 17 students who had non-overlapping diagnoses on the two tests. In most cases, the changes showed that the student was learning more about subtraction. For example, the students who had Smaller-From-Larger on the first test had bugs involving borrowing on the second test. Evidently, they had learned something about borrowing between the two tests, but not enough to do it perfectly. Another classic case is a student who moved from Borrow-From-Zero to Stops-Borrow-At-Multiple-Zero. This student evidently learned about borrowing across one zero beween tests, but had not yet learned borrowing across several zeros—a skill which is usually taught after borrowing across one zero. Of the 17 students, there were only four who did not show this pattern.

8. CONCLUSIONS

Prior to the studies reported here, it was felt that all subtraction errors could be modelled with two mechanisms: bugs and slips. Slips were a "performance" phenomena that were expected to be highly unstable over time and only loosely related to the subtraction problems they occur in. Bugs were a "competence" phenomena reflecting mistaken beliefs about the skill and as such, bugs were expected to be consistent across a whole test and stable across tests given some days apart.

There were two areas of uncertainty marring this early view of a world composed of bugs and slips. First, half the students that we had analyzed at that point (i.e. in the Nicaraguan study) were not consistently following bugs during the test. We attributed this to the shortness of the test and its lack of diagnosticity; if only we had more data on each of those students, the bugs they had could be diagnosed. But there remained the possibility that these students were not really buggy, and that a third mechanism would have to be added to bugs and slips to model their behavior.

The second uncertainty was raised by teachers who reported that they had seen bugs (or what appeared to them to be bugs) come and go over very short periods of time. It could be that bugs

are not stable like bugs in computer programs in that they required active participation of the teacher or some other aspect of the learning environment to make them change. If they were not stable, then a new mechanism would be needed to model how bugs changed so quickly with no outside intervention. In short, the parsimony of the bug-slips view of the world was threatened.

The Southbay study was conducted with excellent tests, an improved DEBUGGY, and a dedicated staff of experienced diagnosticians. This diagnostic power seemed sufficient to determine whether bugs and slips alone would be enough to model any individual's performance on a test, or whether a significant proportion of the population could not be analyzed in these terms. The latter hypothesis was correct: 34% of the population could not be diagnosed in terms of bugs and slips.

The Short-term study, although conducted with only a small number of students, sufficed to test whether bugs were in general stable. They are not. Of the students who had bugs on the first test, only 12% had the same bugs on the second test, and some had no bugs at all. There are definitely major short term instabilities in bugs.

Data was also collected to compare the long-term stability of bugs by testing several months apart. Roughly speaking, the long-term stability data is very similar to the short-term stability data. This implies that the bug instabilities could be a result of testing itself rather than the time between tests. That is, short-term instability can not be attributed to students remembering the test from the preceding day and actively trying to do the present test differently, nor can long term instability be attributed to remediation or spontaneous remission of the bugs. Instead, it appears that instability results from students adopting a different "mental set" for the duration of each test.

This challenges us to change our image of a bug as something that necessarily exists over time as part of the child's long term beliefs about subtraction. Instead, many students' bugs appear to be manufactured as the test begins and held consistently for the duration of the test, only to be dismissed and evaporate from memory as soon as the test is over and they have served their pur-

pose, namely to get the student through the test. That is, bugs appear to be manifestations of rational, albeit incorrect, problem solving strategies for working the test. As such, there is no reason for a student to retain the strategy after the test is over; it or another like it can be generated again next time. (Indeed, practice may have exactly the wrong effect here. When the student has just invented a bug, practice may solidify the bug in memory thus making remediation more difficult.)

This view is formalized by Repair Theory. It describes the kind of local problems and their solutions that lead students to perform as if they had bugs. It has proved successful in explaining a certain percentage of the behavior that DEBUGGY could not diagnose; these students appear to have been tinkering with various problem solving strategies in the midst of the test, and thus did not exhibit consistent enough behavior for DEBUGGY to characterize it with bugs. Repair Theory was successful in explaining the apparent instabilities of bugs. Students who applied one problem solving strategy consistently on the first test, and thus were diagnosed as having bugs, often applied a different problem solving strategy to solve the same local problems on the second test, leading to a different bug-level diagnosis on that test. Often, they chose to use a variety of problem solving strategies on the second test, which explains how they can appear to have no bugs at all on that test.

Despite the success of Repair Theory, there are still a substantial number of students whose errors can not be explained, and an even larger number whose changing pattern of errors over two tests cannot be explained. These are fit targets for further theoretical and empirical work.

Yet even at the bug level, the data present several challenges. The goal of explaining how such a wide variety of bugs could exist is being tackled by ourselves and others (Brown & VanLehn, 1980; Young & O'Shea, forthcoming). Such an effort could lead to new, deeper theories of skill acquisition. Others are investigating more immediate applications of the bug-level diagnosis. Bunderson (1980) is doing controlled experiments to determine whether remediation can be improved if the tutors are given the student's diagnosis by DEBUGGY. Resnick has found some methods of

teaching that appear to successfully remediate bugs, and is looking into methods of teaching that will prevent bugs from being developed in the first place (Resnick, 1981). Some preliminary research (VanLehn & Brown, 1978) has been directed toward understanding theoretically how a procedure could be given meaning for the student, thereby blocking bug formation. Yet another challenge is to push the Buggy paradigm beyond place-value arithmetic into other kinds of problem solving and procedural skills. A simplified bug-like notation has been found sufficient to represent systematic errors in signed-number arithmetic, and a diagnostic system for it has been developed (Tatsuoka, Birenbaum, Tatsuoka & Baillie, 1980). Evidence of bugs have been found in the work of students doing high school algebra (Carry, Lewis & Bernard, 1979; Matz, 1980). The concept of a bug in procedural skills seems to have wide applicability as a basis for developing new psychological and pedagogical theories.

Of course, the real star of this investigation is DEBUGGY. It proved to have a seemingly superhuman ability to perform diagnosis. Occasionally the experts differed with its opinions, but no more so than they differed among themselves. As a research tool, it was superb. However, there are problems with employing it in the educational system. One is that teachers are ill equipped to use its diagnoses. Not only is the concept of a bug foreign to them, but there currently exist no remedial aids or programs that employ bug-level diagnostic information. A second major problem is that it has taken a great deal of effort to accumulate the extensive database of subtraction bugs we now have. To duplicate this effort for each new procedural skill is a daunting task. Repair Theory could be of help here in that it can in principle generate bugs for a new procedural skill with only a few changes. Testing this ability is just one of the exciting directions further research can pursue.

REFERENCES

Ashlock, R.B. *Error patterns in computation.* Columbus, Ohio: Bell and Howell, 1976.

Brown, J.S. & Burton, R.B. Diagnostic models for procedural bugs in basic

mathematical skills. *Cognitive Science,* 1978, *2,* 155-192.
Brown, J.S. & VanLehn, K. Repair Theory: A generative theory of bugs in procedural skills. *Cognitive Science,* 1980, *4,* 379-426.
Brownell, W.A. The evaluation of learning in arithmetic. In *Arithmetic in General Education.* 16th Yearbook of the National Council of Teachers of Mathematics. Washington, D.C.: N.C.T.M., 1941.
Brueckner, L.J. *Diagnostic and remedial teaching in arithmetic.* Philadelphia, Pa.: John C. Winston Co., 1930.
Bunderson, V. Personal communication, 1980.
Burton, R.B. DEBUGGY: Diagnosis of errors in basic mathematical skills. In D.H. Sleeman & J.S. Brown (Eds.) *Intelligent Tutoring Systems.* London: Academic Press, 1981.
Buswell, G.T. *Diagnostic studies in arithmetic.* Chicago: University of Chicago Press, 1926.
Carry, L.R., Lewis, C. & Bernard, J.E. Psychology of equation solving: An information processing study. Dept. of Curriculum & Instruction, University of Texas at Austin, Austin, Texas: 1978.
Cox, L.S. Diagnosing and remediating systematic errors in addition and subtraction computations. *The Arithmetic Teacher,* February 1975.
Durnin, J.H. & Scandura, J.M. Algorithmic approach to assessing behavior potential: comparison with item forms. In J.M. Scandura (Ed.) *Problem Solving: a structural/process approach with instructional implications.* New York, N.Y.: Academic Press, 1977.
Friend, J. Domain referenced adaptive testing. Palo Alto, C.: Xerox Palo Alto Research Center technical report CIS-10, 1981.
Friend, J. & Burton, R. A teacher's manual of subtraction bugs. Palo Alto, Ca.: Xerox Palo Alto Research Center working paper, 1981.
Lankford, F.G. Some computational strategies of seventh grade pupils. ERIC reports. School of Education, Virginia University, 1972.
Haviland, S.E. Buggy's analysis of the TORQUE Wellesley Data. Palo Alto, Ca.: Xerox Palo Alto Research Center working paper, 1979.
Norman, D.A. Slips of the Mind and an Outline for a Theory of Action (CHIP report No. 88). La Jolla: University of California, Center for Human Information Processing, 1979.
Norman, D.A. Categorization of Action Slips. *Psychological Review,* 1981, *88,* 1-15.
Matz, M. Towards a process model of high school algebra errors. In D.H. Sleeman & J.S. Brown (Eds.) *Intelligent Tutoring Systems.* London: Academic Press, 1981.
Resnick, L. Syntax and semantics in learning to subtract. In *Addition and Subtraction: a developmental perspective.* T. Carpenter, J. Moser, and T. Romberg (Eds.) Hillsdale, N.J.: Lawrence Erlbaum Associates, 1981.
Roberts, G.H. The failure strategies of third grade arithmetic pupils. *The Arithmetic Teacher.* May, 1968.
Searle, B., Friend, J., & Suppes, P. *The radio mathematics project: Nicaragua*

1974-1975. Institute for Mathematical Studies in the Social Sciences, Stanford University, 1976.

Tatsuoka, K.K., Birenbaum, M., Tatsuoka, M.M., & Baillie, R. Psychometric approach to error analysis on response patterns of achievement tests. Report 80-3 of the Computer-based Education Research Laboratory, University of Illinois, Urbana, Ill.: 1980.

VanLehn, K. & Brown, J.S. Planning Nets: a representation for formalizing analogies and semantic models of procedural skills. In R.E. Snow, P.A. Frederico, & W.E. Montague (Eds.) *Aptitude learning and instruction: Cognitive process analyses.* Hillsdale, N.J.: Lawrence Erlbaum Associates, 1978.

Young, R.M. and O'Shea, T. Errors in Children's Subtraction. Submitted for publication, forthcoming.

Appendix 1
The Bug Glossary:
A description of each bug

0-N=0/AFTER/BORROW
When a column has a 1 that was changed to a 0 by a previous borrow, the student writes 0 as the answer to that column. (914 - 486 = 508)

0-N=0/EXCEPT/AFTER/BORROW
Thinks 0-N is 0 except when the column has been borrowed from. (906 - 484 = 502)

0-N=N/AFTER/BORROW
When a column has a 1 that was changed to a 0 by a previous borrow, the student writes the bottom digit as the answer to that column. (512 - 136 = 436)

0-N=N/EXCEPT/AFTER/BORROW
Thinks 0-N is N except when the column has been borrowed from. (906 - 484 = 582)

1-1=0/AFTER/BORROW
If a column starts with 1 in both top and bottom and is borrowed from, the student writes 0 as the answer to that column. (812 - 518 = 304)

1-1=1/AFTER/BORROW
If a column starts with 1 in both top and bottom and is borrowed from, the student writes 1 as the answer to that column. (812 - 518 = 314)

ADD/BORROW/CARRY/SUB
The student adds instead of subtracting but he subtracts the carried digit instead of adding it. (54 - 38 = 72)

ADD/BORROW/DECREMENT
Instead of decrementing the student adds 1, carrying to the next column if necessary.

```
  8 6 3        8 9 3
- 1 3 4      - 1 0 4
-------      -------
  7 4 9        8 0 9
```

ADD/BORROW/DECREMENT/WITHOUT/CARRY
Instead of decrementing the student adds 1. If this addition results in 10 the student does not carry but simply writes both digits in the same space.

```
  8 6 3        8 9 3
- 1 3 4      - 1 0 4
-------      -------
  7 4 9        710 9
```

ADD/INSTEADOF/SUB
The student adds instead of subtracting. (32 - 15 = 47)

ADD/LR/DECREMENT/ANSWER/CARRY/TO/RIGHT Adds columns from left to right instead of subtracting. Before writing the column's answer, it is decremented and truncated to the units digit. A one is added into the next column to the right. (411 - 215 = 527)

ADD/NOCARRY/INSTEADOF/SUB
The student adds instead of subtracting. If carrying is required he does not add the carried digit. (47 - 25 = 62)

ALWAYS/BORROW
The student borrows in every column regardless of whether it is necessary. (488 - 229 = 1159)

ALWAYS/BORROW/LEFT
The student borrows from the leftmost digit instead of borrowing from the digit immediately to the left. (733 - 216 = 427)

BLANK/INSTEADOF/BORROW
When a borrow is needed the student simply the skips the column and goes on to the next (425 - 283 = 2 2)

BORROW/ACROSS/TOP/SMALLER/DECREMENTING/TO

When decrementing a column in which the top is smaller than the bottom, the student adds 10 to the top digit, decrements the column being borrowed into and borrows from the next column to the left. Also the student skips any column which has a 0 over a 0 or a blank in the borrowing process.

```
  1 8 3        5 1 3
- 9 5        - 2 6 8
-------      -------
  9 7          2 5 4
```

BORROW/ACROSS/ZERO

When borrowing across a 0, the student skips over the 0 to borrow from the next column. If this causes him to have to borrow twice he decrements the same number both times.

```
  9 0 4        9 0 4
-     7      - 2 3 7
-------      -------
  8 0 7        5 7 7
```

BORROW/ACROSS/ZERO/OVER/BLANK

When borrowing across a 0 over a blank, the student skips to the next column to decrement. (402 - 6 = 306)

BORROW/ACROSS/ZERO/OVER/ZERO

Instead of borrowing across a 0 that is over a 0, the student does not change the 0 but decrements the next column to the left instead. (802 -304 = 308)

BORROW/ADD/DECREMENT/INSTEADOF/ZERO

Instead of borrowing across a 0, the student changes the 0 to 1 and doesn't decrement any column to the left. (307 - 108 = 219)

BORROW/ADD/IS/TEN

The student changes the number that causes the borrow into 10 instead of adding 10 to it. (83 - 29 = 51)

BORROW/DECREMENTING/TO/BY/EXTRAS

When there is a borrow across 0's, the student does not add 10 to the column he is doing but instead adds 10 minus the number of 0's borrowed across.

```
  3 0 8          3 0 0 8
- 1 3 9        - 1 3 5 9
-------        ---------
  1 6 8          1 6 4 7
```

BORROW/DIFF/0-N=N&SMALL-LARGE=0

The student doesn't borrow. For columns of the form 0 - N he writes N as the answer. Otherwise he writes 0. (304 - 179 = 270)

BORROW/DON'T/DECREMENT/TOP/SMALLER

The student will not decrement a column if the top number is smaller than the bottom number.

```
  7 3 2        7 3 2
- 4 8 4      - 4 3 4
-------      -------
  2 5 8        2 9 8
  Wrong        Correct
```

BORROW/DON'T/DECREMENT/UNLESS/BOTTOM/SMALLER

The student will not decrement a column unless the bottom number is smaller than the top number.

```
  7 3 2        7 3 2
- 4 8 4      - 4 3 4
-------      -------
  2 5 8        3 0 8
```

BORROW/FROM/ALL/ZERO

Instead of borrowing across 0's, the student changes all the 0's to 9's but does not continue borrowing from the column to the left. (3006 - 1807 = 2199)

BORROW/FROM/BOTTOM

The student borrows from the bottom row instead of the top one.

```
  8 7          8 2 7
- 2 8        - 2 0 8
-------      -------
  7 9          8 3 9
```

BORROW/FROM/BOTTOM/INSTEADOF/ZERO

When borrowing from a column of the form 0 - N, the student decrements the bottom number instead of the 0.

```
  6 0 8          1 0 8
- 2 4 9        -   4 9
-------        -------
  3 7 9          7 9
```

BORROW/FROM/LARGER
When borrowing, the student decrements the larger digit in the column regardless of whether it is on the top or the bottom. (872 - 294 = 598)

BORROW/FROM/ONE/IS/NINE
When borrowing from a 1, the student treats the 1 as if it were 10, decrementing it to a 9. (316 - 139 = 267)

BORROW/FROM/ONE/IS/TEN
When borrowing from a 1, the student changes the 1 to 10 instead of to 0. (414 - 277 = 237)

BORROW/FROM/ZERO
Instead of borrowing across a 0, the student changes the 0 to 9 but does not continue borrowing from the column to the left.

```
  3 0 6      3 0 0 6      1 0 3
- 1 8 7    - 1 8 0 7    -   4 5
---------  -----------  -------
  2 1 9      1 2 9 9      1 5 8
```

BORROW/FROM/ZERO&LEFT/OK
Instead of borrowing across a 0, the student changes the 0 to 9 but does not continue borrowing from the column ot the left. However if the digit to the left of the 0 is over a blank then the student does the correct thing.

```
  3 0 6      3 0 0 6      1 0 3      2 0 3
- 1 8 7    - 1 8 0 7    -   4 5    -   4 5
---------  -----------  -------    -------
  2 1 9      1 2 9 9        5 8        1 5 8
  Wrong      Wrong        Correct    Correct
```

BORROW/FROM/ZERO/IS/TEN
When borrowing across 0, the student changes the 0 to 10 and does not decrement any digit to the left. (604 - 235 = 479)

BORROW/IGNORE/ZERO/OVER/BLANK
When borrowing across a 0 over a blank, the student treats the column with the zero as if it weren't there.

```
  5 0 5      5 0 8
-     7    -     7
---------  -------
  4   8      5 0 1
  Wrong      Correct
```

BORROW/INTO/ONE=TEN
When a borrow is caused by a 1, the student changes the 1 to a 10 instead of adding 10 to it. (71 - 38 = 32)

BORROW/NO/DECREMENT
When borrowing the student adds 10 correctly but doesn't change any column to the left. (62 - 44 = 28)

BORROW/NO/DECREMENT/EXCEPT/LAST
Decrements only in the last column of the problem. (6262 - 4444 = 1828)

BORROW/ONCE/THEN/SMALLER/FROM/LARGER
The student will borrow only once per exercise. From then on he subtracts the smaller from the larger digit in each column regardless of their positions. (7127 - 2389 = 5278)

BORROW/ONCE/WITHOUT/RECURSE
The student will borrow only once per problem. After that, if another borrow is required the student adds the 10 correctly but does not decrement. If there is a borrow across a 0 the student changes the 0 to 9 but does not decrement the digit to the left of the 0.

```
  5 3 5      4 0 8
- 2 7 8    - 2 3 9
---------  -------
  3 5 7      2 6 9
```

BORROW/ONLY/FROM/TOP/SMALLER
When borrowing, the student tries to find a column in which the top number is smaller than the bottom. If there is one he decrements that, otherwise he borrows correctly. (9283 - 3566 = 5627)

BORROW/ONLY/ONCE
When there are several adjacent borrows, the student decrements only with the first borrow. (535 - 278 = 357)

BORROW/SKIP/EQUAL
When decrementing, the student skips over columns in which the top digit and the bottom digit are the same. (923 - 427 = 406)

BORROW/TEN/PLUS/NEXT/DIGIT/INTO/ZERO
When a borrow is caused by a 0 the student does not add 10 correctly. What he does instead is add 10 plus the digit in the next column to the left. (50 - 38 = 17)

BORROW/TREAT/ONE/AS/ZERO
When borrowing from 1, the student treats the 1 as if it were 0; that is, he changes the 1 to 9 and decrements the number to the left of the 1. (313 - 159 = 144)

BORROW/UNIT/DIFF
The student borrows the difference between the top digit and the bottom digit of the current column. In other words, he borrows just enough to do the subtraction, which then always results in 0. (86 - 29 = 30)

BORROW/WONT/RECURSE
Instead of borrowing across a 0, the student stops doing the exercise. (8035 - 2662 = 3)

BORROWED/FROM/DON'T/BORROW
When there are two borrows in a row the student does the first borrow correctly but with the second borrow he does not decrement (he does add 10 correctly). (143 - 88 = 155)

CAN'T/SUBTRACT
The student skips the entire problem. (8 - 3 =)

COPY/TOP/IN/LAST/COLUMN/IF/BORROWED/FROM
After borrowing from the last column, the student copies top digit as the answer (80 - 34 = 76).

DECREMENT/ALL/ON/MULTIPLE/ZERO
When borrowing across a 0 and the borrow is caused by 0, the student changes the right 0 to 9 instead of 10. (600 - 142 =457)

DECREMENT/BY/ONE/PLUS/ZEROS
When there is a borrow across zero, decrements the number to the left of the zero(s) by an extra one for every zero borrowed across. (4005 - 6 = 1999)

DECREMENT/BY/TWO/OVER/TWO
When borrowing from a column of the form N - 2, the student decrements the N by 2 instead of 1. (83 - 29 =44)

DECREMENT/LEFTMOST/ZERO/ONLY
When borrowing across two or more 0's the student changes the leftmost of the row of 0's to 9 but changes the other 0's to 10's. He will give answers like: (1003 - 958 = 1055)

DECREMENT/MULTIPLE/ZEROS/BY/NUMBER/TO/LEFT
When borrowing across 0's the student changes the leftmost 0 to a 9, changes the next 0 to 8, etc. (8002 - 1714 = 6278)

DECREMENT/MULTIPLE/ZEROS/BY/NUMBER/TO/RIGHT
When borrowing across 0's the student changes the rightmost 0 to a 9, changes the next 0 to 8, etc. (8002 - 1714 = 6188)

DECREMENT/ON/FIRST/BORROW
The first column that requires a borrow is decremented before the column subtract is done. (832 - 265 = 566)

DECREMENT/ONE/TO/ELEVEN
Instead of decrementing a 1, the student changes the 1 to an 11. (314 - 6 = 2118)

DECREMENT/TOP/LEQ/IS/EIGHT
When borrowing from 0 or 1, changes the 0 or 1 to 8; does not decrement digit to the left of the 0 or 1. (4013 - 995 = 3778)

DIFF/0-N=0
When the student encounters a column of the form 0 - N he doesn't borrow; instead he writes 0 as the column answer. (40 - 21 = 20)

DIFF/0-N=N
When the student encounters a column of the form 0 - N, he doesn't borrow. Instead he writes N as the answer. (80 - 27 = 67)

DIFF/0-N=N/WHEN/BORROW/FROM/ZERO
When borrowing across a 0 and the borrow is caused by a 0, the student doesn't borrow. Instead he writes the bottom number as the column answer. He will borrow correctly in the next column or in other circumstances.

```
  1 0 0        4 0 0
- 3 2        - 2 4 8
-------      -------
  7 2          1 6 8
```

DIFF/1-N=1
When a column has the form 1 - N the student writes 1 as the column answer. (51 - 27 = 31)

DIFF/N-0=0
The student thinks that N - 0 is 0. (57 - 20 = 30)

DIFF/N-N=N
Whenever there is a column that has the same number on the top and the bottom, the student writes that number as the answer. (83 - 13 = 73)

DOESNT/BORROW
The student stops doing the exercise when a borrow is required. (833 - 262 = 1)

DON'T/DECREMENT/SECOND/ZERO
When borrowing across a 0 and the borrow is caused by a 0, the student changes the 0 he is borrowing across into a 10 instead of a 9. (700 - 258 =452)

DON'T/DECREMENT/ZERO
When borrowing across a 0, the student changes the 0 to 10 instead of 9. (506 - 318 = 198)

DON'T/DECREMENT/ZERO/OVER/BLANK
The student will not borrow across a zero that is over a blank. (305 - 9 = 306)

DON'T/DECREMENT/ZERO/OVER/ZERO
The student will not borrow across a zero that is over a zero. (305 - 107 = 308)

DON'T/DECREMENT/ZERO/UNTIL/BOTTOM/BLANK
When borrowing across a 0, the student changes the 0 to a 10 instead of a 9 unless the 0 is over a blank, in which case he does the correct thing.

```
  5 0 6        3 0 4
- 3 1 8      -     9
-------      -------
  1 9 8        2 9 5
  Wrong        Correct
```

DON'T/WRITE/ZERO
Doesn't write zeros in the answer. (24 - 14 = 1)

DOUBLE/DECREMENT/ONE
When borrowing from a 1, the student treats the 1 as a 0 (changes the 1 to 9 and continues borrowing to the left. (813 - 515 = 288)

FORGET/BORROW/OVER/BLANKS
The student doesn't decrement a number that is over a blank. (347 - 9 = 348)

IGNORE/LEFTMOST/ONE/OVER/BLANK
When the left column of the exercises has a 1 that is over a blank, the student ignores that column. (143 - 22 = 21)

IGNORE/ZERO/OVER/BLANK
Whenever there is column that has a 0 over a blank, the student ignores that column. (907 - 5 = 9 2)

INCREMENT/OVER/LARGER
When borrowing from a column in which the top is smaller than the bottom, the student increments instead of decrementing. (833 -277 = 576)

INCREMENT/ZERO/OVER/BLANK
When borrowing across a 0 over a blank, the student increments the 0 instead of decrementing.
(402 - 6 = 416)

N-9=N-1/AFTER/BORROW
If a column is of the form N - 9 and has been borrowed from, when the student does that column he subtracts 1 instead of subtracting 9. (834 - 796 = 127)

N-N/AFTER/BORROW/CAUSES/BORROW
Borrows with columns of the form N-N if the column has been borrowed from. (953 - 147 = 7106)

N-N/CAUSES/BORROW
Borrows with columns of the form N-N. (953 - 152 = 7101)

N-N=1/AFTER/BORROW
If a column had the form N - N and was borrowed from, the student writes 1 as the answer to that column. (944 - 348 = 616)

N-N=9/PLUS/DECREMENT
When a column has the same number on the top and the bottom the student writes 9 as the answer and decrements the next column to the left even though borrowing is not necessary.
(94 - 34 = 59)

ONCE/BORROW/ALWAYS/BORROW
Once a student has borrowed he continues to borrow in every remaining column in the exercise. (488 - 229 = 1159)

QUIT/WHEN/BOTTOM/BLANK
When the bottom number has fewer digits than the top number, the student quits as soon as the bottom number runs out. (439 - 4 = 5)

SIMPLE/PROBLEM/STUTTER/SUBTRACT
When the bottom number is a single digit and the top number has two or more digits, the student repeatedly subtracts the single bottom digit from each digit in the top number.
(348 - 2 = 126)

SMALLER/FROM/LARGER
The student doesn't borrow; in each column he subtracts the smaller digit from the larger one.
(81 - 38 = 57)

SMALLER/FROM/LARGER/INSTEAD/OF/BORROW/FROM/ZERO
The student does not borrow across 0. Instead he will subtract the smaller from the larger digit.

```
  3 0 6          3 0 6
-     8        - 1 4 8
-------        -------
  3 0 2          1 6 2
```

SMALLER/FROM/LARGER/WHEN/BORROWED/FROM
When there are two borrows in a row the student does the first one correctly but for the second one he does not borrow; instead he subtracts the smaller from the larger digit regardless of order. (824 - 157 = 747)

SMALLER/FROM/LARGER/WITH/BORROW
When borrowing the student decrements correctly, then subtracts the smaller digit from the larger as if he had not borrowed at all. (73 - 24 = 411)

STOPS/BORROW/AT/MULTIPLE/ZERO
Instead of borrowing across several 0's, the student adds 10 to the column he's doing but doesn't change any column to the left. (4004 - 9 = 4005)

STOPS/BORROW/AT/SECOND/ZERO
When borrowing across several 0's, changes the right 0 to 9 but not the other 0's. (4004 - 9 = 4095)

STOPS/BORROW/AT/ZERO
Instead of borrowing across a 0, the student adds 10 to the column he's doing but doesn't decrement from a column to the left. (404 - 187 = 227)

STUTTER/SUBTRACT
When there are blanks in the bottom number, the student subtracts the leftmost digit of the bottom number in every column that has a blank. (4369 - 22 = 2147)

SUB/BOTTOM/FROM/TOP
The student always subtracts the top digit from the bottom digit. If the bottom digit is smaller, he decrements the top digit and adds 10 to the bottom before subtracting. If the bottom digit is zero, however, he writes the top digit in the answer. If the top digit is 1 greater than the bottom he writes 9. He will give answers like this. (4723 - 3065 = 9742)

SUB/COPY/LEAST/BOTTOM/MOST/TOP
The student does not subtract. Instead he copies digits from the exercise to fill in the answer space. He copies the leftmost digit from the top number and the other digits from the bottom number. He will give answers like this: (648 - 231 = 631)

SUB/ONE/OVER/BLANKS
When there are blanks in the bottom number, the student subtracts 1 from the top digit. (548 - 2 = 436)

TREAT/TOP/ZERO/AS/NINE
In a 0-N column, the student doesn't borrow. Instead he treats the 0 as if it were a 9. (30 - 4 = 39)

TREAT/TOP/ZERO/AS/TEN
In a 0-N column, the student adds 10 to it correctly but doesn't change any column to the left. (40 - 27 = 23)

X-N=0/AFTER/BORROW
If a column has been borrowed from, students writes zero as its answer. (234 - 115 = 109)

X-N=N/AFTER/BORROW
If a column has been borrowed from, students writes the bottom digit as its answer. (234 - 165 = 169)

ZERO/AFTER/BORROW
When a column requires a borrow, the student decrements correctly but writes 0 as the answer. (65 - 48 = 10)

ZERO/INSTEAD/OF/BORROW/FROM/ZERO
The student won't borrow if he has to borrow across 0. Instead he will write 0 as the answer to the column requiring the borrow.

```
  7 0 2        7 0 2
-     8    - 3 4 8
-------    -------
  7 0 0        3 6 0
```

ZERO/INSTEADOF/BORROW
The student doesn't borrow; he writes 0 as the answer instead. (42 -16 = 30)

Appendix 2
All Diagnoses

The diagnoses of all tests of all students analyzed by DEBUGGY fall into the following categories:

Correct	112	(10%)
Slips	223	(20%)
Buggy	417	(37%)
Undiagnosed	386	(34%)
total	1138	(100%)

The diagnoses the students that were analyzed as having bugs are shown ordered by their frequency of occurrence. Diagnoses consisting of more than one bug are shown in parentheses. There are 157 distinct diagnoses, of which only 50 occurred more than once. However, these 50 diagnoses account for 310 of the 417 cases (74%).

The names used for bugs in this appendix and all others are somewhat different in form than those used in the test. Smaller-From-Larger is written here as *SMALLER/FROM/LARGER. Also, the diagnoses in the appendices sometimes contain *coercions*. A coercions is a modifier that is

included in a diagnosis to improve the fit of the bugs to the student's errors. Most often, these slightly perturb the definitions of bugs. For example, certain bugs modify the procedure so that on occasion it will write column answers that are greater than 9. Some students who have these bugs apparently know from addition that there should only be one answer digit per column, so they only write the units digit. To capture this, the coercion !WRITE/UNITS/DIGIT/ONLY is added to the diagnoses of such students by DEBUGGY. Coercions can easily be picked out in the appendices because they have exclamation points in their names. For more on coercions, see (Burton, 1981).

```
106 occurrences:
*SMALLER/FROM/LARGER

34 occurrences:
*STOPS/BORROW/AT/ZERO

18 occurrences:
*BORROW/ACROSS/ZERO

12 occurrences:
*BORROW/NO/DECREMENT

11 occurrences:
*BORROW/FROM/ZERO

9 occurrences:
(*STOPS/BORROW/AT/ZERO *DIFF/0-N=N)
(*BORROW/ACROSS/ZERO *DIFF/0-N=N)

7 occurrences:
(*BORROW/ACROSS/ZERO/OVER/ZERO *BORROW/ACROSS/ZERO/OVER/BLANK)
*BORROW/DON'T/DECREMENT/UNLESS/BOTTOM/SMALLER

6 occurrences:
(*BORROW/NO/DECREMENT *DIFF/0-N=N)
*BORROW/NO/DECREMENT/EXCEPT/LAST
*ALWAYS/BORROW/LEFT

6 occurrences:
(*SMALLER/FROM/LARGER *IGNORE/LEFTMOST/ONE/OVER/BLANK)
*BORROW/DIFF/0-N=N&SMALL-LARGE=0
(*QUIT/WHEN/BOTTOM/BLANK *SMALLER/FROM/LARGER)

4 occurrences:
(*BORROW/ACROSS/ZERO *DIFF/0-N=0)
*DON'T/DECREMENT/ZERO
(*STOPS/BORROW/AT/ZERO *BORROW/ONCE/THEN/SMALLER/FROM/LARGER
          *DIFF/0-N=N)
*STOPS/BORROW/AT/MULTIPLE/ZERO
(*BORROW/INTO/ONE=TEN *STOPS/BORROW/AT/ZERO)

3 occurrences:
*DECREMENT/MULTIPLE/ZEROS/BY/NUMBER/TO/RIGHT
*0-N=N/EXCEPT/AFTER/BORROW
(*STOPS/BORROW/AT/ZERO *BORROW/ONCE/THEN/SMALLER/FROM/LARGER)

2 occurrences:
(*DIFF/N-0=0 *SMALLER/FROM/LARGER *DIFF/0-N=0)
*SIMPLE/PROBLEM/STUTTER/SUBTRACT
*DECREMENT/ALL/ON/MULTIPLE/ZERO
(*STOPS/BORROW/AT/ZERO *DIFF/0-N=0)
*DECREMENT/LEFTMOST/ZERO/ONLY
(*BORROW/ACROSS/ZERO *0-N=N/EXCEPT/AFTER/BORROW)
*DIFF/0-N=N
*QUIT/WHEN/BOTTOM/BLANK
```

```
*STUTTER/SUBTRACT
(*BORROW/ONLY/FROM/TOP/SMALLER *BORROW/ACROSS/ZERO/OVER/ZERO
        *BORROW/ACROSS/ZERO/OVER/BLANK)
*DECREMENT/MULTIPLE/ZEROS/BY/NUMBER/TO/LEFT
(*SMALLER/FROM/LARGER/INSTEAD/OF/BORROW/FROM/ZERO *BORROW/ONCE/THEN/SMALLER/FROM/LARGER
        *DIFF/0-N=N)
*0-N=0/AFTER/BORROW
(*STUTTER/SUBTRACT *BORROW/ACROSS/ZERO/OVER/ZERO)
(*BORROW/ACROSS/ZERO *BORROW/DON'T/DECREMENT/TOP/SMALLER)
*BORROW/ACROSS/TOP/SMALLER/DECREMENTING/TO
*BORROW/DON'T/DECREMENT/TOP/SMALLER

1 occurrence:
(*STOPS/BORROW/AT/ZERO *1-1=1/AFTER/BORROW)
(*BORROW/ACROSS/ZERO/OVER/ZERO *0-N=N/EXCEPT/AFTER/BORROW
        *1-1=0/AFTER/BORROW)
(*0-N=N/EXCEPT/AFTER/BORROW *BORROW/SKIP/EQUAL)
*IGNORE/LEFTMOST/ONE/OVER/BLANK
(*STOPS/BORROW/AT/ZERO *SMALLER/FROM/LARGER/WHEN/BORROWED/FROM)
(*BORROW/NO/DECREMENT *DIFF/0-N=0)
(*BORROW/ACROSS/ZERO *DIFF/0-N=N/WHEN/BORROW/FROM/ZERO *BORROW/ACROSS/ZERO/OVER/ZERO)
(*SMALLER/FROM/LARGER/INSTEAD/OF/BORROW/FROM/ZERO *DIFF/0-N=N
        *SMALLER/FROM/LARGER/WHEN/BORROWED/FROM)
(*0-N=N/EXCEPT/AFTER/BORROW *1-1=1/AFTER/BORROW)
(*BORROW/ONCE/WITHOUT/RECURSE *DON'T/DECREMENT/ZERO/UNTIL/BOTTOM/BLANK)
(*BORROW/NO/DECREMENT *0-N=N/EXCEPT/AFTER/BORROW)
(*BORROW/ACROSS/ZERO *QUIT/WHEN/BOTTOM/BLANK *DIFF/0-N=0)
(*BORROW/ACROSS/ZERO *FORGET/BORROW/OVER/BLANKS *DIFF/0-N=N)
(*DON'T/WRITE/ZERO *BORROW/IGNORE/ZERO/OVER/BLANK)
(*BORROW/ACROSS/ZERO *0-N=0/EXCEPT/AFTER/BORROW *1-1=1/AFTER/BORROW)
(*BORROW/ACROSS/ZERO *0-N=0/EXCEPT/AFTER/BORROW *1-1=0/AFTER/BORROW)
(*STOPS/BORROW/AT/ZERO *DIFF/0-N=N *DON'T/WRITE/ZERO)
(*BORROW/ACROSS/ZERO *SIMPLE/PROBLEM/STUTTER/SUBTRACT)
*ADD/INSTEADOF/SUB
(*0-N=N/AFTER/BORROW *N-N=1/AFTER/BORROW
        *SMALLER/FROM/LARGER/INSTEAD/OF/BORROW/FROM/ZERO)
(*BORROW/INTO/ONE=TEN *DECREMENT/MULTIPLE/ZEROS/BY/NUMBER/TO/LEFT)
(*BORROW/ACROSS/ZERO *BORROW/SKIP/EQUAL)
*FORGET/BORROW/OVER/BLANKS
(*0-N=N/EXCEPT/AFTER/BORROW *1-1=0/AFTER/BORROW)
*1-1=0/AFTER/BORROW
(*BORROW/ACROSS/ZERO *BORROW/ONCE/THEN/SMALLER/FROM/LARGER)
(*SMALLER/FROM/LARGER *DIFF/N-N=N *DIFF/0-N=0)
(*BORROW/FROM/ONE/IS/TEN *DECREMENT/LEFTMOST/ZERO/ONLY *BORROWED/FROM/DON'T/BORROW)
(*STOPS/BORROW/AT/ZERO *0-N=0/EXCEPT/AFTER/BORROW
        *1-1=0/AFTER/BORROW)
*ADD/LR/DECREMENT/ANSWER/CARRY/TO/RIGHT
(*STOPS/BORROW/AT/ZERO *0-N=0/AFTER/BORROW)
(*TREAT/TOP/ZERO/AS/TEN *0-N=N/EXCEPT/AFTER/BORROW *BORROW/ACROSS/ZERO/OVER/BLANK)
*IGNORE/ZERO/OVER/BLANK
*DECREMENT/TOP/LEQ/IS/EIGHT
(*BORROW/DON'T/DECREMENT/UNLESS/BOTTOM/SMALLER *DON'T/WRITE/ZERO)
(*BORROW/ACROSS/TOP/SMALLER/DECREMENTING/TO *BORROW/ONCE/THEN/SMALLER/FROM/LARGER)
*DON'T/DECREMENT/ZERO/OVER/BLANK
(*DIFF/0-N=N *DIFF/N-0=0)
(!ONLY/WRITE/UNITS/DIGIT *BORROW/ACROSS/ZERO *BORROW/DON'T/DECREMENT/TOP/SMALLER
        *N-N/AFTER/BORROW/CAUSES/BORROW)
(*STOPS/BORROW/AT/ZERO *BORROW/DON'T/DECREMENT/TOP/SMALLER
        *DIFF/0-N=N)
(*DECREMENT/ALL/ON/MULTIPLE/ZERO *DOUBLE/DECREMENT/ONE)
(*BORROW/UNIT/DIFF *SMALLER/FROM/LARGER/INSTEAD/OF/BORROW/FROM/ZERO
        *0-N=N/EXCEPT/AFTER/BORROW)
*BORROW/ONCE/THEN/SMALLER/FROM/LARGER
(*BORROW/FROM/ZERO *0-N=0/AFTER/BORROW)
*SUB/COPY/LEAST/BOTTOM/MOST/TOP
(*N-N/AFTER/BORROW/CAUSES/BORROW *0-N=0/EXCEPT/AFTER/BORROW)
*N-9=N-1/AFTER/BORROW
(!SUB/UNITS/SPECIAL *BORROW/ACROSS/ZERO *SMALLER/FROM/LARGER)
```

```
(*DECREMENT/LEFTMOST/ZERO/ONLY *BORROWED/FROM/DON'T/BORROW)
(*BORROW/FROM/ZERO *0-N=N/AFTER/BORROW)
(*DIFF/N-0=0 *STOPS/BORROW/AT/ZERO)
(*DIFF/0-N=N/WHEN/BORROW/FROM/ZERO *STOPS/BORROW/AT/ZERO
        *BORROW/DON'T/DECREMENT/TOP/SMALLER)
(*BORROW/FROM/BOTTOM/INSTEADOF/ZERO *DIFF/0-N=N)
(*BORROW/FROM/ONE/IS/TEN *BORROW/FROM/ZERO/IS/TEN)
(*SMALLER/FROM/LARGER *0-N=0/EXCEPT/AFTER/BORROW)
(*BORROW/TREAT/ONE/AS/ZERO *N-N=1/AFTER/BORROW *DON'T/DECREMENT/ZERO/OVER/BLANK)
*SUB/BOTTOM/FROM/TOP
(*BORROW/NO/DECREMENT *DECREMENT/TOP/LEQ/IS/EIGHT *X-N=N/AFTER/BORROW
        *BORROW/ONCE/THEN/SMALLER/FROM/LARGER)
*DOUBLE/DECREMENT/ONE
(*BORROW/ACROSS/ZERO *DIFF/0-N=N/WHEN/BORROW/FROM/ZERO
        *SMALLER/FROM/LARGER/WHEN/BORROWED/FROM)
(*BORROW/ACROSS/ZERO *X-N=0/AFTER/BORROW)
(!ONLY/WRITE/UNITS/DIGIT *STOPS/BORROW/AT/MULTIPLE/ZERO
        *N-N/AFTER/BORROW/CAUSES/BORROW)
(*BORROW/FROM/ONE/IS/NINE *BORROW/FROM/ZERO *DON'T/DECREMENT/ZERO/OVER/BLANK)
(*BORROW/FROM/ZERO&LEFT/TEN/OK *0-N=N/AFTER/BORROW)
*BORROW/ACROSS/ZERO/OVER/BLANK
*BORROW/ACROSS/ZERO/OVER/ZERO
*BORROW/FROM/ALL/ZERO
(!ONLY/WRITE/UNITS/DIGIT *N-N/AFTER/BORROW/CAUSES/BORROW)
*N-N/CAUSES/BORROW
(*BORROW/DON'T/DECREMENT/UNLESS/BOTTOM/SMALLER *X-N=0/AFTER/BORROW)
(!ONLY/WRITE/UNITS/DIGIT *SIMPLE/PROBLEM/STUTTER/SUBTRACT
        *N-N/AFTER/BORROW/CAUSES/BORROW)
(*BORROW/NO/DECREMENT *SUB/ONE/OVER/BLANK)
(*DON'T/DECREMENT/ZERO/UNTIL/BOTTOM/BLANK *BORROW/ACROSS/ZERO/OVER/ZERO)
(*DON'T/DECREMENT/ZERO *DECREMENT/ONE/TO/ELEVEN)
(*FORGET/BORROW/OVER/BLANKS *BORROW/DON'T/DECREMENT/TOP/SMALLER *BORROW/SKIP/EQUAL)
(*DIFF/0-N=N/WHEN/BORROW/FROM/ZERO *DON'T/DECREMENT/ZERO)
(*SMALLER/FROM/LARGER *0-N=N/EXCEPT/AFTER/BORROW *N-N/CAUSES/BORROW)
*BORROW/ONLY/FROM/TOP/SMALLER
(*BORROW/ONLY/FROM/TOP/SMALLER *0-N=N/AFTER/BORROW)
(*BORROW/ACROSS/ZERO *BORROW/ONLY/FROM/TOP/SMALLER)
(*0-N=N/AFTER/BORROW *BORROW/ACROSS/ZERO/OVER/ZERO *BORROW/ACROSS/ZERO/OVER/BLANK)
(*FORGET/BORROW/OVER/BLANKS *STOPS/BORROW/AT/ZERO *BORROW/ONCE/THEN/SMALLER/FROM/LARGER)
(*BORROW/INTO/ONE=TEN *DECREMENT/MULTIPLE/ZEROS/BY/NUMBER/TO/RIGHT
        *BORROW/ACROSS/ZERO/OVER/ZERO)
*ZERO/INSTEADOF/BORROW
*BORROW/FROM/ZERO/IS/TEN
*BORROW/DECREMENTING/TO/BY/EXTRAS
*ADD/NOCARRY/INSTEADOF/SUB
*0-N=N/AFTER/BORROW
(*BORROW/ACROSS/ZERO/OVER/ZERO *SMALLER/FROM/LARGER/WHEN/BORROWED/FROM
        *BORROW/ACROSS/ZERO/OVER/BLANK)
(*SMALLER/FROM/LARGER *DIFF/0-N=0)
(*BORROW/ACROSS/ZERO *DIFF/N-N=N)
(*FORGET/BORROW/OVER/BLANKS *STOPS/BORROW/AT/ZERO
        *0-N=N/EXCEPT/AFTER/BORROW  *1-1=1/AFTER/BORROW)
(*BORROW/FROM/ONE/IS/NINE *BORROW/FROM/ZERO *DIFF/0-N=N)
*CAN'T/SUBTRACT
(*SIMPLE/PROBLEM/STUTTER/SUBTRACT *BORROW/ACROSS/ZERO/OVER/ZERO)
*DON'T/WRITE/ZERO
(!ONLY/WRITE/UNITS/DIGIT *BORROW/ONLY/FROM/TOP/SMALLER *DECREMENT/ALL/ON/MULTIPLE/ZERO
        *N-N/AFTER/BORROW/CAUSES/BORROW)
(*0-N=N/EXCEPT/AFTER/BORROW *N-N=9/PLUS/DECREMENT)
(*BORROW/ACROSS/ZERO *IGNORE/LEFTMOST/ONE/OVER/BLANK *BORROW/SKIP/EQUAL)
(*FORGET/BORROW/OVER/BLANKS *STOPS/BORROW/AT/ZERO)
(*BLANK/INSTEADOF/BORROW *DIFF/0-N=N)
(!WRITE/LEFT/TEN *SMALLER/FROM/LARGER)
(*DOUBLE/DECREMENT/ONE *SMALLER/FROM/LARGER/WHEN/BORROWED/FROM)
*1-1=1/AFTER/BORROW
(*BORROW/NO/DECREMENT/EXCEPT/LAST *0-N=N/EXCEPT/AFTER/BORROW)
(*BORROW/ACROSS/ZERO *SUB/ONE/OVER/BLANK *0-N=N/AFTER/BORROW
        *0-N=N/EXCEPT/AFTER/BORROW)
(*DON'T/DECREMENT/ZERO *1-1=0/AFTER/BORROW)
```

Appendix 3
Bug Occurrence Frequencies

The number of times each bug in the database occurred is shown. The first two columns show the number of times the given bug occurred in DEBUGGY's diagnoses. The first column, labelled "alone" is the number of times the bug occurred alone, as the only element of the diagnosis. The second column, labelled "cmd." is the number of times the bug occurred as part of a multi-bug diagnosis, or "compound" bug as it was called in (Brown & Burton, 1978). Thus, for example, the bug `*1-1=1/AFTER/BORROW` occurred once alone, and five times as part of a larger diagnosis. The third column indicates which bugs were added to the data base since the Southbay study began. It is interesting that some of these new bugs are not at all rare. These data include all tests of all subjects in both the Southbay and Short-term study. As always, the data come from the reanalysis that was performed after all new bugs had been entered in the database. Rows that would be all zero have been left blank to highlight those bugs in the data base which never occurred in these studies. Of the 104 bugs in the data base, 77 bugs occurred at least once.

alone	cmd.	new?	Bug
2	2		*0-N=0/AFTER/BORROW
0	5	new	*0-N=0/EXCEPT/AFTER/BORROW
1	6	new	*0-N=N/AFTER/BORROW
3	14	new	*0-N=N/EXCEPT/AFTER/BORROW
1	5		*1-1=0/AFTER/BORROW
1	4		*1-1=1/AFTER/BORROW
			*ADD/BORROW/CARRY/SUB
			*ADD/BORROW/DECREMENT
			*ADD/BORROW/DECREMENT/WITHOUT/CARRY
1	0		*ADD/INSTEADOF/SUB
1	0	new	*ADD/LR/DECREMENT/ANSWER/CARRY/TO/RIGHT
1	0		*ADD/NOCARRY/INSTEADOF/SUB
			*ALWAYS/BORROW
6	0		*ALWAYS/BORROW/LEFT
0	1	new	*BLANK/INSTEADOF/BORROW
2	1	new	*BORROW/ACROSS/TOP/SMALLER/DECREMENTING/TO
18	33		*BORROW/ACROSS/ZERO
1	12		*BORROW/ACROSS/ZERO/OVER/BLANK
1	18	new	*BORROW/ACROSS/ZERO/OVER/ZERO
			*BORROW/ADD/DECREMENT/INSTEADOF/ZERO
			*BORROW/ADD/IS/TEN
1	0	new	*BORROW/DECREMENTING/TO/BY/EXTRAS
5	0		*BORROW/DIFF/0-N=N&SMALL-LARGE=0
2	6		*BORROW/DON'T/DECREMENT/TOP/SMALLER
7	2		*BORROW/DON'T/DECREMENT/UNLESS/BOTTOM/SMALLER
1	0		*BORROW/FROM/ALL/ZERO
			*BORROW/FROM/BOTTOM
0	1		*BORROW/FROM/BOTTOM/INSTEADOF/ZERO
			*BORROW/FROM/LARGER
0	2		*BORROW/FROM/ONE/IS/NINE

0	2		*BORROW/FROM/ONE/IS/TEN
11	4		*BORROW/FROM/ZERO
0	1		*BORROW/FROM/ZERO&LEFT/TEN/OK
1	1	new	*BORROW/FROM/ZERO/IS/TEN
0	1		*BORROW/IGNORE/ZERO/OVER/BLANK
0	6		*BORROW/INTO/ONE=TEN
12	10		*BORROW/NO/DECREMENT
6	1	new	*BORROW/NO/DECREMENT/EXCEPT/LAST
1	13	new	*BORROW/ONCE/THEN/SMALLER/FROM/LARGER
0	1		*BORROW/ONCE/WITHOUT/RECURSE
1	5		*BORROW/ONLY/FROM/TOP/SMALLER
			*BORROW/ONLY/ONCE
0	4		*BORROW/SKIP/EQUAL
		new	*BORROW/TEN/PLUS/NEXT/DIGIT/INTO/ZERO
0	1		*BORROW/TREAT/ONE/AS/ZERO
0	1		*BORROW/UNIT/DIFF
			*BORROW/WONT/RECURSE
0	2	new	*BORROWED/FROM/DON'T/BORROW
1	0		*CAN'T/SUBTRACT
		new	*COPY/TOP/IN/LAST/COLUMN/IF/BORROWED/FROM
2	2		*DECREMENT/ALL/ON/MULTIPLE/ZERO
		new	*DECREMENT/BY/ONE/PLUS/ZEROS
			*DECREMENT/BY/TWO/OVER/TWO
2	2	new	*DECREMENT/LEFTMOST/ZERO/ONLY
2	1	new	*DECREMENT/MULTIPLE/ZEROS/BY/NUMBER/TO/LEFT
3	1	new	*DECREMENT/MULTIPLE/ZEROS/BY/NUMBER/TO/RIGHT
			*DECREMENT/ON/FIRST/BORROW
0	1	new	*DECREMENT/ONE/TO/ELEVEN
1	1	new	*DECREMENT/TOP/LEQ/IS/EIGHT
0	12		*DIFF/0-N=0
2	38		*DIFF/0-N=N
0	4		*DIFF/0-N=N/WHEN/BORROW/FROM/ZERO
			*DIFF/1-N=1
0	4		*DIFF/N-0=0
0	2		*DIFF/N-N=N
			*DOESNT/BORROW
			*DON'T/DECREMENT/SECOND/ZERO
4	3		*DON'T/DECREMENT/ZERO
1	2	new	*DON'T/DECREMENT/ZERO/OVER/BLANK
			*DON'T/DECREMENT/ZERO/OVER/ZERO
0	2		*DON'T/DECREMENT/ZERO/UNTIL/BOTTOM/BLANK
1	3	new	*DON'T/WRITE/ZERO
1	2		*DOUBLE/DECREMENT/ONE
1	5		*FORGET/BORROW/OVER/BLANKS
1	6	new	*IGNORE/LEFTMOST/ONE/OVER/BLANK
1	0		*IGNORE/ZERO/OVER/BLANK
			*INCREMENT/OVER/LARGER
		new	*INCREMENT/ZERO/OVER/BLANK
1	0		*N-9=N-1/AFTER/BORROW
0	6	new	*N-N/AFTER/BORROW/CAUSES/BORROW
1	1	new	*N-N/CAUSES/BORROW
0	2		*N-N=1/AFTER/BORROW
0	1		*N-N=9/PLUS/DECREMENT
			*ONCE/BORROW/ALWAYS/BORROW
2	6		*QUIT/WHEN/BOTTOM/BLANK

```
  2   3   new   *SIMPLE/PROBLEM/STUTTER/SUBTRACT
106  18         *SMALLER/FROM/LARGER
  0   5   new   *SMALLER/FROM/LARGER/INSTEAD/OF/BORROW/FROM/ZERO
  0   5   new   *SMALLER/FROM/LARGER/WHEN/BORROWED/FROM
                *SMALLER/FROM/LARGER/WITH/BORROW
  4   1         *STOPS/BORROW/AT/MULTIPLE/ZERO
          new   *STOPS/BORROW/AT/SECOND/ZERO
 34  33         *STOPS/BORROW/AT/ZERO
  2   2         *STUTTER/SUBTRACT
  1   0   new   *SUB/BOTTOM/FROM/TOP
  1   0   new   *SUB/COPY/LEAST/BOTTOM/MOST/TOP
  0   2   new   *SUB/ONE/OVER/BLANK
          new   *TREAT/TOP/ZERO/AS/NINE
  0   1         *TREAT/TOP/ZERO/AS/TEN
  0   2   new   *X-N=0/AFTER/BORROW
  0   1   new   *X-N=N/AFTER/BORROW
                *ZERO/AFTER/BORROW
                *ZERO/INSTEAD/OF/BORROW/FROM/ZERO
  0   1         *ZERO/INSTEADOF/BORROW
```

Appendix 4
DEBUGGY versus the Experts

Several different ways of comparing DEBUGGY's diagnosis with the experts' diagnoses are presented here. First, the results from the Southbay study are presented. They show excellent agreement regarding which bugs a Buggy student has, but differ a little bit on whether a student should be placed in the Buggy category or the Undiagnosed category.

	Experts' Diagnosis			
DEBUGGY's Diagnosis	Slips	Buggy	Undiagnosed	totals
Slips	148 (95%)	4 (3%)	3 (2%)	155 (100%)
Buggy	3 (1%)	233 (79%)	59 (20%)	295 (100%)
Undiagnosed	18 (8%)	30 (13%)	188 (80%)	236 (100%)
totals	169	267	250	686

Table 4.1: A comparison of DEBUGGY's diagnosis with the experts' diagnosis by diagnostic category. The second row, for example, shows that of the 295 subjects that were analyzed as Buggy by DEBUGGY, 1% were analyzed as having slips by the expert, 79% as Buggy, and 20% as Undiagnosed. Only the first tests of those subjects who were tested twice are counted in this table.

Equal	193 (83%)
Expert removed a bug	13 (6%)
Expert added a bug	4 (2%)
Overlap	10 (4%)
Otherwise	13 (6%)
total	233 (100%)

Table 4.2: A comparison of DEBUGGY's diagnosis with the expert's diagnosis by bug. Of the 233 subjects that both DEBUGGY and the expert agree were Buggy, 193 were given exactly the same diagnosis. In all but 13 cases, (i.e. 94% of the time) there was substantial agreement between the experts and DEBUGGY. Only the first tests of those subjects who were tested twice are counted in this table.

In the following four tables, a three-way comparison of expert and DEBUGGY diagnoses is presented. Two experts, namely the authors, analyzed the Short-term data, as did DEBUGGY. The first three tables compare their judgments by diagnostic category. The first compares the experts to each other, and the next two compare DEBUGGY to each expert individually. It can be seen that the experts agreed more with DEBUGGY than with each other.

Table 4.3

Friend	Slips	Bugs	Undiagnosed	totals
		VanLehn		
Slips	54 (90%)	1 (2%)	5 (8%)	60 (100%)
Bugs	1 (3%)	23 (68%)	10 (30%)	34 (100%)
Undiagnosed	6 (23%)	10 (38%)	10 (38%)	26 (100%)
totals	61	34	25	120

Table 4.4

DEBUGGY	Slips	Bugs	Undiagnosed	totals
		VanLehn		
Slips	40 (98%)	0 (0%)	1 (2%)	41 (100%)
Bugs	1 (3%)	28 (80%)	6 (17%)	35 (100%)
Undiagnosed	20 (45%)	6 (14%)	18 (41%)	44 (100%)
totals	61	34	25	120

Table 4.5

	Friend			
DEBUGGY	Slips	Bugs	Undiagnosed	totals
Slips	41 (98%)	0 (0%)	1 (2%)	42 (100%)
Bugs	1 (3%)	28 (76%)	8 (22%)	37 (100%)
Undiagnosed	21 (42%)	9 (18%)	20 (40%)	50 (100%)
totals	63	37	29	129

The following table compares the experts' diagnoses and DEBUGGY's by comparing the sets of bugs they produced for the cases where both put the student in the Buggy category.

Table 4.6	V vs. F	V vs. D	F vs. D
Equal	9 (39%)	15 (54%)	20 (71%)
One bug-set is a subset of the other	5 (22%)	6 (21%)	2 (7%)
Overlap	3 (13%)	5 (18%)	2 (7%)
Otherwise	6 (26%)	2 (7%)	4 (14%)
total	23 (100%)	28 (100%)	28 (100%)

Appendix 5
Short-term Stability

The short-term stability results are presented. The tests were given two days apart using the same or very similar tests. The tests were analyzed by DEBUGGY and by an expert. DEBUGGY's analyses will be presented first.

	Second Test				
First Test	Correct	Slips	Buggy	Undiagnosed	totals
Correct	3 (43%)	3 (43%)	0 (0%)	1 (14%)	7 (100%)
Slips	4 (20%)	11 (55%)	0 (0%)	5 (25%)	20 (100%)
Buggy	0 (0%)	2 (12%)	12 (71%)	3 (18%)	17 (100%)
Undiagnosed	0 (0%)	5 (22%)	6 (26%)	12 (52%)	23 (100%)
totals	7	21	18	21	67

Table 5.1: Short-term stability by diagnostic category. The above table show how the students changed among diagnostic classes across the two tests. The figures in parentheses show the

proportion of the first test's category that the given cell of the table represents. For example, of the 17 students who were in the Buggy category on the first test, 0% were in the Correct category on the second test, 12% were in the Slips category, 71% remained in the Buggy category, and 18% could not be diagnosed on the second test.

The switching between the Correct and Slips category was expected since slips are assumed to be a labile, "performance" phenomena. The switching among the Undiagnosed and Slips categories is probably due to students who should have been place in Slips instead of Undiagnosed, but they made so many slips that they exceeded DEBUGGY's 90%-correct threshold for the Slips category. What's unexplained is the movement into and out of the Buggy category. This movement is examined more closely in the next table.

Table 5.2: Short-term stability by bug set. This table shows DEBUGGY's diagnoses of the 12 subjects who were Buggy on both tests. Two of these had the same diagnoses both times. The other ten subjects' diagnoses had bugs appearing and disappearing, indicating instability. Cases of bug migration are marked with Ⓡ.

Diagnoses are equal:

- (*BORROW/ACROSS/ZERO *DIFF/0-N=0) *becomes*
  (*BORROW/ACROSS/ZERO *DIFF/0-N=0)
- (*DIFF/N-0=0 *SMALLER/FROM/LARGER *DIFF/0-N=0) *becomes*
  (*DIFF/N-0=0 *SMALLER/FROM/LARGER *DIFF/0-N=0)

Diagnoses overlap:

- (*0-N=N/EXCEPT/AFTER/BORROW) *becomes*
  (*0-N=N/EXCEPT/AFTER/BORROW *1-1=0/AFTER/BORROW)
- (*BORROW/ACROSS/ZERO *SIMPLE/PROBLEM/STUTTER/SUBTRACT) *becomes*
  (*BORROW/ACROSS/ZERO)
- (*STOPS/BORROW/AT/ZERO *DIFF/0-N=N) *becomes*
  (*STOPS/BORROW/AT/ZERO *DIFF/0-N=N *SMALLER/FROM/LARGER/WHEN/BORROWED/FROM)
- Ⓡ (*STOPS/BORROW/AT/ZERO *0-N=0/EXCEPT/AFTER/ZERO *1-1=1/AFTER/BORROW) *becomes*
  (*STOPS/BORROW/AT/ZERO *0-N=0/EXCEPT/AFTER/ZERO *1-1=0/AFTER/BORROW)
- (*STOPS/BORROW/AT/ZERO *0-N=0/EXCEPT/AFTER/ZERO *1-1=0/AFTER/BORROW) *becomes*
  (*STOPS/BORROW/AT/ZERO *DIFF/0-N=0)
- (*BORROW/ACROSS/ZERO/OVER/ZERO *0-N=N/EXCEPT/AFTER/ZERO *1-1=0/AFTER/BORROW) *becomes*
  (*0-N=N/EXCEPT/AFTER/BORROW *BORROW/SKIP/EQUAL)

Diagnoses do not overlap:

- (*DIFF/N-0=0 *SMALLER/FROM/LARGER *DIFF/0-N=0) *becomes*
  (*BORROW/ACROSS/ZERO)
- (*SMALLER/FROM/LARGER) *becomes*
  (*STOPS/BORROW/AT/ZERO *DIFF/0-N=N)
- (*BORROW/ONCE/WITHOUT/RECURSE *DONT/DECREMENT/ZERO/UNTIL/BOTTOM/BLANK) *becomes*
  (*BORROW/NO/DECREMENT)
- Ⓡ (*BORROW/ACROSS/ZERO *DIFF/0-N=0) *becomes*
  (*STOPS/BORROW/AT/ZERO)

Table 5.3: Short-term stability by expert's diagnostic categories: The expert was able to uncover cases of tinkering by comparing the answers of the test items across tests. Since each item was matched to a corresponding item on the other test, it was easier to come to a decision about which errors were due to slips and which were due to tinkering or bugs. The following table presents this Repair Theoretic analysis in the same format as table 5.1.

	Second Test					
First Test	Correct	Slips	Buggy	Tinkering	Undiagnosed	totals
Correct	3 (43%)	4 (57%)	0 (0%)	0 (0%)	0 (0%)	7 (100%)
Slips	4 (13%)	25 (83%)	1 (3%)	0 (0%)	0 (0%)	30 (100%)
Buggy	0 (0%)	1 (6%)	14 (78%)	2 (11%)	1 (6%)	18 (100%)
Tinkering	0 (0%)	0 (0%)	1 (14%)	5 (71%)	1 (14%)	7 (100%)
Undiagnosed	0 (0%)	1 (20%)	0 (0%)	0 (0%)	4 (80%)	5 (100%)
totals	7	31	16	7	6	67

Most of the switching is between the Correct and Slips categories, as expected by the assumed in instability of slips, and between the Buggy and Tinkering categories, as predicted by bug/tinkering migration.

Table 5.4: Short-term stability of bugs and impasses. There were 36 cases where the student was diagnosed as having the performing the correct procedure on both tests, with perhaps some slips each time. There were 4 cases where the student was undiagnosable on both tests. The other 37 cases are presented on the table which begins on the next page. They are separated into four categories: stable bugs, stable impasses, unstable bugs and unstable impasses. Tinkering is notated with parenthesized lists of the form

(<impasse> <repair> <repair> < repair>)

where <impasses> is the name of the impasses and <repair> is the name of one of the repairs being used to get past the impasse. The repairs are documented in (Brown & VanLehn, 1980). The impasses used here are:

T=0/BF	Can't borrow from zero.
T=1/BF	Can't borrow from one.
T=00/BF	Can't borrow from multiple zeros.
T=0AB/BF	Can't borrow from one's that have been decremented to zero.
T=B/BF	Can't borrow from a column where the top and bottom digits are equal.
T=0/SC	Can't process a column with a zero on top.
T=0AB/SC	Can't process a column with a top zero created by decrementing a one.
T=B/SC	Can't process a column where the top and bottom digits are equal.
T=0AB&T=BBB/SC	Can't process a column with a top zero created by decrementing a one, whose top and bottom digits were equal before the one was decremented.
ANS/OVERFLOW	Can't write two digits for a column answer.
DECR/TWICE	Can't borrow from a digit that's been borrowed from already.

Appearing and disappearing bugs:

- (*DECREMENT/ALL/ON/MULTIPLE/ZERO/EXCEPT/AFTER/BORROW
  *SIMPLE/PROBLEM/STUTTER/SUBTRACT) *becomes*
  (*DECREMENT/ALL/ON/MULTIPLE/ZERO/EXCEPT/AFTER/BORROW)
- (*BORROW/ACROSS/ZERO *SIMPLE/PROBLEM/STUTTER/SUBTRACT (T=0AB/BF NOOP WEIRD)) *becomes*
  (*BORROW/ACROSS/ZERO (T=0AB/BF NOOP QUIT))
- () slips *becomes*
  (*SMALLER/FROM/LARGER)
- (*BORROW/IGNORE/ZERO/OVER/BLANK) *becomes*
  () slips
- (*1-1=0/AFTER/BORROW) *becomes*
  () undiagnosed

Appearing and disappearing impasses (and bugs):

- ((T=0/BF NOOP REFOCUS/LEFT) (T=0AB&T=BBB/SC IGNORE DEMEMOIZE)) *becomes*
  ((T=0/BF BORROW NOOP) (ANS/OVERFLOW NOOP ERASE&PARTIAL/REDO))
- (*SIMPLE/PROBLEM/STUTTER/SUBTRACT) *becomes*
  ((SIMPLE/MULTIPLICATION/PROBLEM IGNORE WEIRD))
- (*DIFF/0-N=N *STOPS/BORROW/AT/ZERO (T=B/BF IGNORE BACKUP-REFOCUS/VERTICALLY)
  (T=BBB/SC REFOCUS/VERTICALLY DEMEMOIZE)) *becomes*
  (*DIFF/0-N=N *STOPS/BORROW/AT/ZERO *SMALLER/FROM/LARGER/WHEN/BORROWED/FROM)
- (*DIFF/0-N=0/WHEN/BORROW/FROM/ZERO *BORROW/ACROSS/ZERO
  (T=0AB/SC DEMEMOIZE IGNORE WEIRD)) *becomes*
  ((T=0/BF BORROW MOVL) (ANS/OVERFLOW FADD) (T=0AB/SC IGNORE QUIT))
- (*DIFF/N-0=0 *SMALLER/FROM/LARGER *DIFF/0-N=0) *becomes*
  (*BORROW/ACROSS/ZERO (T=0AB/BF NOOP))
- (*BORROW/ACROSS/ZERO (DECR/TWICE NOOP IGNORE) (T=0AB&T=BBB/SC IGNORE NOOP)) *becomes*
  (*STOPS/BORROW/AT/ZERO)
- ((T=1/BF NOOP ADD10 IGNORE WEIRD)(T=0/BF FSELF NOOP IGNORE)) *becomes*
  () Undiagnosed

Stable bugs:

- (*STOPS/BORROW/AT/MULTIPLE/ZERO) *becomes*
  (*STOPS/BORROW/AT/MULTIPLE/ZERO)
- (*DIFF/N-0=0 *SMALLER/FROM/LARGER *DIFF/0-N=0) *becomes*
  (*DIFF/N-0=0 *SMALLER/FROM/LARGER *DIFF/0-N=0)
- (*DECREMENT/MULTIPLE/ZEROS/BY/NUMBER/TO/RIGHT) *becomes*
  (*DECREMENT/MULTIPLE/ZEROS/BY/NUMBER/TO/RIGHT)

Stable impasses with stable bugs (n.b. impasses with just one repair are bugs):

- (*STOPS/BORROW/AT/ZERO *0-N=0/EXCEPT/AFTER/BORROW ((T=0AB/SC IGNORE DEMEMOIZE))
  becomes
  (*STOPS/BORROW/AT/ZERO *0-N=0/EXCEPT/AFTER/BORROW *0-N=0/AFTER/BORROW)
- ((T=00/BF IGNORE FSELF WEIRD)) *becomes*
  ((T=00/BF IGNORE FSELF WEIRD))
- ((T+1=B/BI WEIRD IGNORE)) *becomes*
  ((T+1=B/BI WEIRD IGNORE))
- (*BORROW/ACROSS/ZERO (T=0/SC DEMEMOIZE REFOCUS/VERTICALLY)) *becomes*
  (*BORROW/ACROSS/ZERO (T=0/SC DEMEMOIZE REFOCUS/VERTICALLY))
- (*BORROW/ACROSS/ZERO (T=0/SC IGNORE DEMEMOIZE REFOCUS/VERTICALLY)) *becomes*

```
   (*BORROW/ACROSS/ZERO (T=0/SC IGNORE DEMEMOIZE REFOCUS/VERTICALLY))
O  ((T=0AB/SC IGNORE NOOP)) becomes
   ((T=0AB/SC IGNORE NOOP))
O  (*BORROW/FROM/BOTTOM/INSTEADOF/ZERO *0-N=N/EXCEPT/AFTER/BORROW
    (T=0AB&T=BBB/SC IGNORE DEMEMOIZE)) becomes
   (*BORROW/FROM/BOTTOM/INSTEADOF/ZERO *0-N=N/EXCEPT/AFTER/BORROW
    (T=0AB&T=BBB/SC IGNORE NOOP DEMEMOIZE))
O  (*0-N=N/EXCEPT/AFTER/BORROW (T=0AB&T=BBB/SC QUIT/THE/TEST)) becomes
   (*0-N=N/EXCEPT/AFTER/BORROW *1-1=0/AFTER/BORROW)
O  (*DON'T/DECREMENT/ZERO (ANS/OVERFLOW IGNORE NOOP)) becomes
   (*DON'T/DECREMENT/ZERO !ONLY/WRITE/UNITS/DIGIT)
O  ((T=0/BF REFOCUS/LEFT BACKUP-REFOCUS/VERTICALLY)) becomes
   (*BORROW/ACROSS/ZERO)
O  (*SIMPLE/PROBLEM/STUTTER/SUBTRACT) becomes
   ((SIMPLE/MULTIPLICATION/PROBLEM IGNORE WEIRD))
O  (*DIFF/0-N=N (T=0/BF NOOP BACKUP-REFOCUS/VERTICALLY)) becomes
   (*DIFF/0-N=N *STOPS/BORROW/AT/ZERO)
```

Appendix 6
Long-term Stability

Table 6.1: Long-term stability by diagnostic category. This shows the diagnostic classes of both tests of the 154 subjects who were tested twice during the Southbay study. 84 of the 154 students (or 55%) stayed in the same diagnostic category.

	Second Test				
First Test	Correct	Slips	Bugs	Undiagnosed	totals
Correct	0	0	0	0	0
Slips	0 (0%)	4 (67%)	2 (33%)	0 (0%)	6 (100%)
Bugs	1 (2%)	5 (8%)	34 (53%)	24 (38%)	64 (100%)
Undiagnosed	3 (4%)	13 (15%)	22 (33%)	46 (55%)	84 (100%)
totals	4	22	58	70	154

Roughly the same proportion of students switched from Buggy to Undiagnosed as from Undiagnosed to Buggy. Neither category contributed significantly to the Correct category. These two facts tend to confound the hypothesis that Buggy students are more often remediated by the current curriculum than Undiagnosed students.

Table 6.2: Long-term stability by bug. The diagnoses for both tests are shown for the 34 students who were Systematic on both tests. 17 students had roughly the same diagnosis on both tests, indicating that these bugs can be persistent problems. The diagnoses of the other 17 students did not overlap. Of these, 13 subjects showed evidence of learning in their bugs in that the subskill their first bug(s) could not accomplish has been mastered, but a more advanced subskill is missing leading to a second bug(s).

Second test's diagnosis is the same as or overlaps with the first test's diagnosis:

- (*SMALLER/FROM/LARGER) *becomes*
  (*SMALLER/FROM/LARGER)
- (*SMALLER/FROM/LARGER) *becomes*
  (*SMALLER/FROM/LARGER)
- (*SMALLER/FROM/LARGER) *becomes*
  (*SMALLER/FROM/LARGER)
- (*SMALLER/FROM/LARGER) *becomes*
  (*SMALLER/FROM/LARGER)
- (*SMALLER/FROM/LARGER) *becomes*
  (*SMALLER/FROM/LARGER)
- (*SMALLER/FROM/LARGER) *becomes*
  (*SMALLER/FROM/LARGER)
- (*SMALLER/FROM/LARGER) *becomes*
  (*SMALLER/FROM/LARGER)
- (*SMALLER/FROM/LARGER) *becomes*
  (*SMALLER/FROM/LARGER)
- (*QUIT/WHEN/BOTTOM/BLANK *SMALLER/FROM/LARGER) *becomes*
  (*SMALLER/FROM/LARGER)
- (*QUIT/WHEN/BOTTOM/BLANK *SMALLER/FROM/LARGER) *becomes*
  (*QUIT/WHEN/BOTTOM/BLANK *SMALLER/FROM/LARGER)
- (*BORROW/DIFF/0-N=N&SMALL-LARGE=0) *becomes*
  (*BORROW/DIFF/0-N=N&SMALL-LARGE=0)
- (*STOPS/BORROW/AT/ZERO *SMALLER/FROM/LARGER/WHEN/BORROWED/FROM) *becomes*
  (*STOPS/BORROW/AT/ZERO)
- (*STOPS/BORROW/AT/ZERO *DIFF/0-N=N) *becomes*
  (*STOPS/BORROW/AT/ZERO)
- (*BORROW/ACROSS/ZERO *DIFF/0-N=N) *becomes*
  (*BORROW/ACROSS/ZERO)
- (*BORROW/ACROSS/ZERO *QUIT/WHEN/BOTTOM/BLANK *DIFF/0-N=0) *becomes*
  (*BORROW/ACROSS/ZERO *DIFF/0-N=N)
- (*STOPS/BORROW/AT/ZERO *BORROW/ONCE/THEN/SMALLER/FROM/LARGER *DIFF/0-N=N) *becomes*
  (*STOPS/BORROW/AT/ZERO *1-1=1/AFTER/BORROW)
- (*STOPS/BORROW/AT/MULTIPLE/ZERO) *becomes*
  (!ONLY/WRITE/UNITS/DIGIT *STOPS/BORROW/AT/MULTIPLE/ZERO
  *N-N/AFTER/BORROW/CAUSES/BORROW)

No overlap between the two tests' diagnoses, some evidence of learning:

- (*ADD/NOCARRY/INSTEADOF/SUB) *becomes*
  (*SMALLER/FROM/LARGER)
- (*SMALLER/FROM/LARGER) *becomes*
  (*DIFF/0-N=N/WHEN/BORROW/FROM/ZERO *DON'T/DECREMENT/ZERO)
- (*SMALLER/FROM/LARGER) *becomes*

(*STOPS/BORROW/AT/ZERO)
- (*SMALLER/FROM/LARGER) *becomes*
  (*BORROW/FROM/BOTTOM/INSTEADOF/ZERO *DIFF/0-N=N)
- (*SMALLER/FROM/LARGER) *becomes*
  (*BORROW/NO/DECREMENT *DIFF/0-N=N)
- (*SMALLER/FROM/LARGER) *becomes*
  (*ALWAYS/BORROW/LEFT)
- (*SMALLER/FROM/LARGER) *becomes*
  (*BORROW/ACROSS/ZERO *DIFF/0-N=N/WHEN/BORROW/FROM/ZERO
  *SMALLER/FROM/LARGER/WHEN/BORROWED/FROM)
- (*SMALLER/FROM/LARGER) *becomes*
  (*BORROW/NO/DECREMENT)
- (*SMALLER/FROM/LARGER) *becomes*
  (*STOPS/BORROW/AT/ZERO *BORROW/ONCE/THEN/SMALLER/FROM/LARGER)
- (*BORROW/NO/DECREMENT/EXCEPT/LAST) *becomes*
  (*STOPS/BORROW/AT/ZERO)
- (*ALWAYS/BORROW/LEFT) *becomes*
  (*BORROW/INTO/ONE=TEN *STOPS/BORROW/AT/ZERO)
- (*BORROW/DON'T/DECREMENT/UNLESS/BOTTOM/SMALLER) *becomes*
  (*BORROW/INTO/ONE=TEN *STOPS/BORROW/AT/ZERO)
- (*BORROW/FROM/ZERO) *becomes*
  (*STOPS/BORROW/AT/MULTIPLE/ZERO)

No overlap between diagnoses, no evidence of learning:

- (*BORROW/ACROSS/ZERO *0-N=N/EXCEPT/AFTER/BORROW) *becomes*
  (*DON'T/DECREMENT/ZERO)
- (*FORGET/BORROW/OVER/BLANKS *BORROW/DON'T/DECREMENT/TOP/SMALLER *BORROW/SKIP/EQUAL)
  becomes (*BORROW/NO/DECREMENT)
- (*BORROW/ACROSS/ZERO *FORGET/BORROW/OVER/BLANKS *DIFF/0-N=N) *becomes*
  (*DON'T/WRITE/ZERO *BORROW/IGNORE/ZERO/OVER/BLANK)
- (*BORROW/NO/DECREMENT *SUB/ONE/OVER/BLANK) *becomes*
  (*BORROW/DON'T/DECREMENT/UNLESS/BOTTOM/SMALLER)