TESTING TECHNIQUES

Defect testing is aimed at uncovering problems in a program.

The goal is to find errors.

Black box testing assumes no knowledge of how a given program is implemented. Test cases are often derived from the requirements.

An **equivalence category** is a collection of inputs that are expected to produce similar outputs. Should select test cases from different equivalence categories.

Glass box testing assumes knowledge of the internals of the program, and different test cases trace the different paths through the code.

THREE HANDY TOOLS

1. **RCS** (or **CVS**)

2. **GDB**

3. **MAKE**

Revision Control System (RCS)

Manages multiple revisions of files.

Automates the storing, retrieval, logging, identification, and merging of revisions.

Useful for text that is revised frequently, including source code, documentation, papers, etc.

`rcs -i file.c`
create and initialize a new RCS file, but do not deposit any revision.

`ci file.c`
store new revision into RCS file.

`co -l file.c`
retrieve a revision from RCS file, store it into the corresponding working file, and lock it for the caller.

```
$ touch test.c
$ ls
test.c
$ rcs -i test.c
RCS file: test.c,v
enter description, terminated with single '.' or eof:
NOTE: This is NOT the log message!
>> Test file in C.
>> .
done
$ ls
test.c  test.c,v
$ ci test.c
test.c,v  <--  test.c
initial revision: 1.1
done
$ co -l test.c
test.c,v  -->  test.c
revision 1.1 (locked)
done
$ vim test.c
$ ci test.c
test.c,v  <--  test.c
new revision: 1.2; previous revision: 1.1
enter log message, terminated with single '.' or eof:
>> First modification.
>> .
done
$
```

Gnu Debugger (GDB)

See what is going on "inside" a program while it executes, or what was going on when it crashed (i.e., Segmentation Fault!).

GDB can do:

- Start your program, specifying anything that might affect its behavior.
- Make your program stop on specified conditions.
- Examine what has happened, when your program has stopped.
- Change things in your program, so you can experiment with correcting the effects of one bug and go on to learn about another.

```
$ more debug.c
main()
{
        int i;
        int a[10];

        for (i=0; i<100; i++)
                a[i]=i;
}
$ gcc -g -o debug debug.c
$ gdb debug
GNU gdb Red Hat Linux (5.3post-0.20021129.18rh)
...
(gdb) break 1
Breakpoint 1 at 0x80482f4: file debug.c, line 1.
(gdb) run
Starting program: debug
Breakpoint 1, main () at debug.c:2
2       {
(gdb) step
main () at debug.c:6
6               for (i=0; i<100; i++)
(gdb) step
7                       a[i]=i;
(gdb) print i
$2 = 0
(gdb)
```

Make

Automatically determines which pieces of a large program need to be recompiled, and issues the commands to recompile them. A file called a `makefile` tells `make` what to do:

```
edit : main.o kbd.o command.o display.o \
            insert.o search.o files.o utils.o
             cc -o edit main.o kbd.o command.o display.o \
                        insert.o search.o files.o utils.o
main.o : main.c defs.h
        cc -c main.c
kbd.o : kbd.c defs.h command.h
        cc -c kbd.c
command.o : command.c defs.h command.h
        cc -c command.c
display.o : display.c defs.h buffer.h
        cc -c display.c
insert.o : insert.c defs.h buffer.h
        cc -c insert.c
search.o : search.c defs.h buffer.h
        cc -c search.c
files.o : files.c defs.h buffer.h command.h
        cc -c files.c
utils.o : utils.c defs.h
        cc -c utils.c
clean :
        rm edit main.o kbd.o command.o display.o \
           insert.o search.o files.o utils.o
```