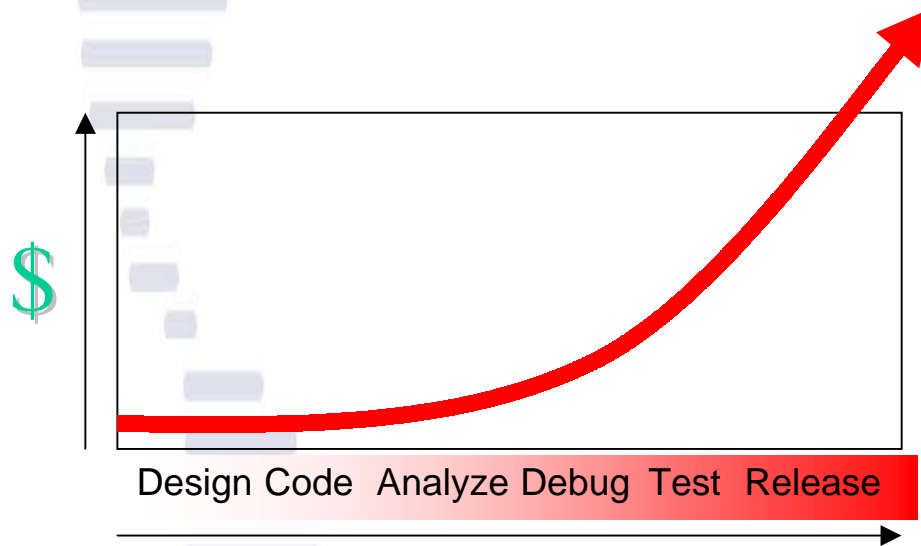


# Cleanscape Testwise

*Software visualization and analysis tool box*



## The Cost of Bugs



**The cost of fixing a bug goes up dramatically the later a bug is found in the development cycle**

### ♦ Cost of Discovery

- ♦ Very high if the customer finds the bug
- ♦ Lowest if the developer finds it him/herself

### ♦ Cost of Correction

- ♦ Very high when discovered while running the application
- ♦ Lowest when spotted in the source code

## Code

CLEANSCAPE  
SourceMill

Automatically  
generate source  
code by  
synthesizing  
object models  
with templates

## Analyze

CLEANSCAPE  
FortranLint  
CLEANSCAPE  
LintPlus

Stop  
software  
problems at  
their source

## Test

CLEANSCAPE  
Testwise

Integrate  
redundant test  
processes into  
an automated  
test process

## Build & Maintain

CLEANSCAPE  
qef

Automate the  
software  
development  
process  
  
Manage  
software  
construction



## Cleanscape Lint Tools Value Proposition

- ♦ **Cleanscape provides software development teams with powerfully simple tools that reduce organizational exposure to risks from latent software problems by automatically identifying problems at their source--in the code prior to compiling or executing programs.**
- ♦ **These tools can save software developers hundreds of hours in problem eradication efforts, more than returning their investment on the first use.**
- ♦ **Key Values**
  - ♦ Shorten the software development cycle
  - ♦ Prevent project delays that result from post-compile testing
  - ♦ Reduce costs by eliminating problems with cheaper resources earlier in the development process
  - ♦ Increase the competitive viability of the software development organization by helping teams produce higher quality products cheaper

## Cleanscape Lint Tools

### CLEANSCAPE FortranLint

Unix  
Windows  
Linux  
Xlint  
Online

#### ♦ Classification

- ♦ Enterprise-class static source code analysis for C and Fortran

#### ♦ Function

- ♦ Identify problems in source code that compilers can't detect
- ♦ Document code
- ♦ Generate call trees & cross-references
- ♦ Web-enabled version available

### CLEANSCAPE LintPlus

Unix  
Windows  
Linux  
Online

#### ♦ Benefits

- ♦ Faster
  - Find problems in seconds, not days
  - Prevent project delays that result from post-compile testing
- ♦ Better
  - Produce higher quality products cheaper
- ♦ Smarter
  - Improved processes, standardized reports increase competitive viability of the development organization
- ♦ Cheaper
  - Eliminate coding problems with cheaper resources earlier in the development process
- ♦ Cleaner code

# Cleanscape FortranLint

CLEANSCAPE  
FortranLint

Unix  
Windows  
Linux  
Xlint  
Online

CLEANSCAPE  
LintPlus

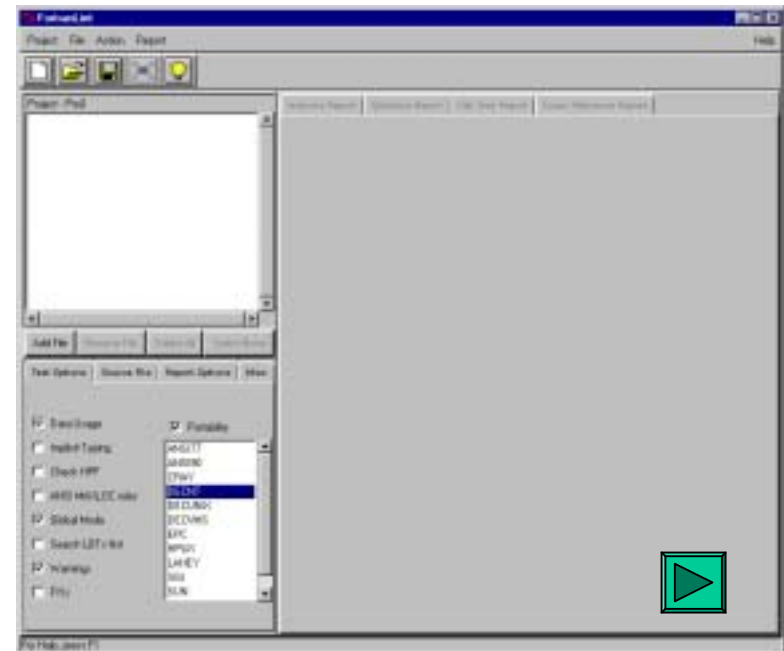
Unix  
Windows  
Linux  
Online

## Classification

- ♦ Fortran source code analysis tool

## Function

- ♦ Processes entire programs written in Fortran for inconsistencies that would prevent applications from running properly
- ♦ Analyzes F77, F90, F95, and many Fortran dialects
- ♦ Conducts local dataflow analysis
- ♦ Performs advanced portability checking
- ♦ Extends static analysis to OpenMP
- ♦ Detects
  - Inconsistencies in variables and argument lists between modules
  - Inconsistencies in common block definitions
  - Non-portable or unused code
  - Unassigned variables or variable type conflicts



## Cleanscape Testwise Value Proposition

- ♦ **Adequate testing can devour about 70% of resources in a typical software development project.**
  - ♦ **Quality often becomes the first casualty when software developers are faced with tight budget constraints and approaching deadlines.**
  - ♦ **Software developers need a way to achieve dramatic cost reductions and quality improvements by automatically analyzing the dynamic behavior of software.**
- ♦ **Key Value**
    - ♦ Testwise allows developers to integrate redundant test processes into an automated test process that helps them identify and eliminate problems early in the development cycle — dramatically reducing resource allocation while increasing software quality.
    - ♦ A powerful, yet easy-to-use, software visualization and analysis toolset, Testwise allows software test engineers to automate software testing for faster, better, smarter, cheaper and — of course — *cleaner* software development

# Cleanscape Testwise Overview

## CLEANSCAPE Testwise

xAtac  
xRetress  
xProf  
xSlice  
xFind  
xVue  
xDiff

### ♦ Classification

- ♦ Software test automation tools for C, C++ programs on Unix, Linux, or Windows platforms.

### ♦ Function

- ♦ Coverage analysis
- ♦ Regression testing
- ♦ Advanced software maintenance
- ♦ Dynamic debugging
- ♦ Performance analysis
- ♦ Dependency tracing
- ♦ File difference display

### ♦ Benefits

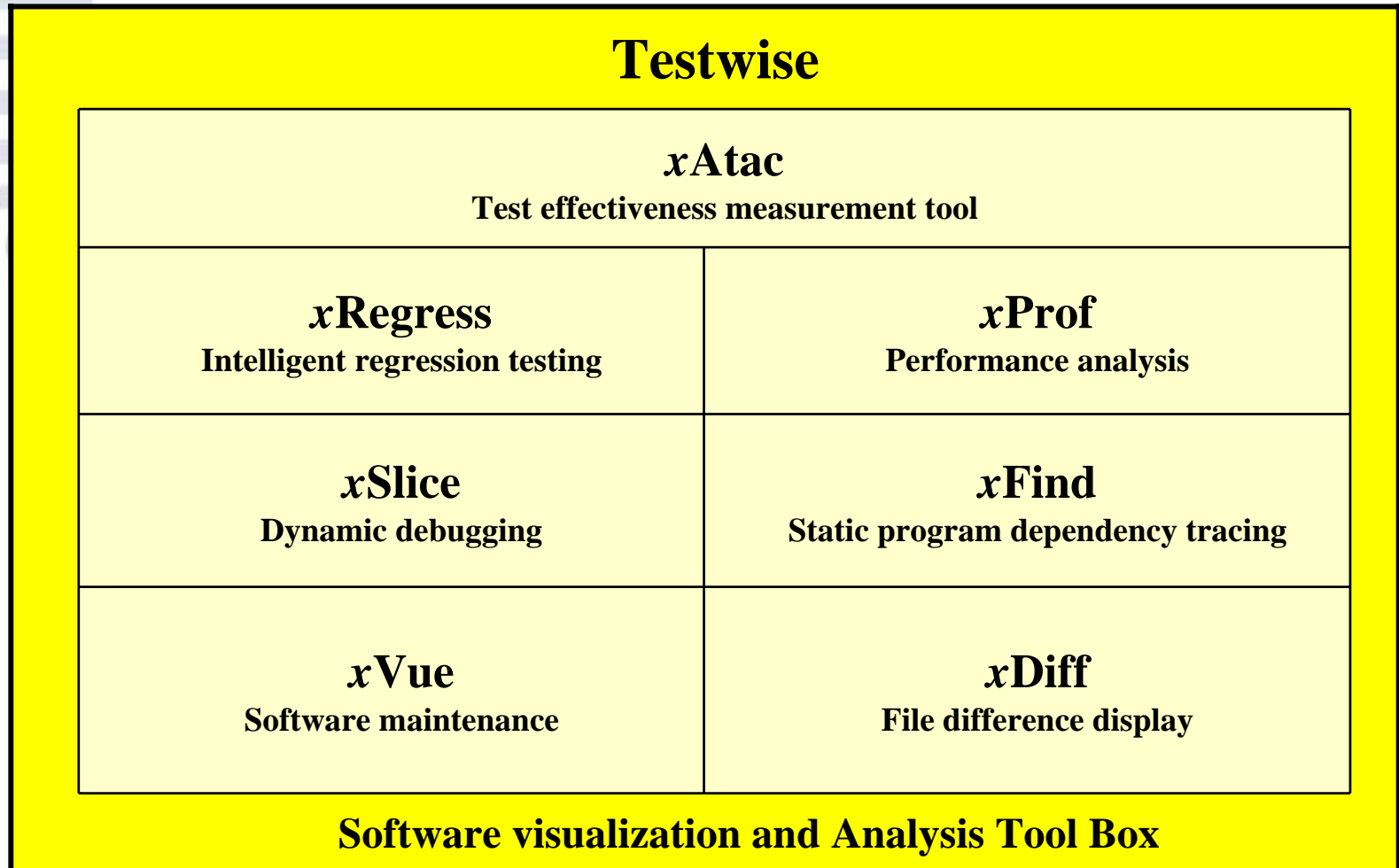
- ♦ Shortens test and maintenance phases of software while providing
  - Faster time to market
  - Better quality product
  - Stronger product
- ♦ Facilitates cross-platform development by providing common test environment for Unix, Linux, and Windows
- ♦ Aids software development, selection, and improvement of test sets
- ♦ Maximizes productivity
- ♦ Easily standardizes and automates established test processes
- ♦ Eliminates need to build test software (Stubs and Drivers)



## Cleanscape Testwise Tool Box

CLEANSCAPE  
Testwise

xAtac  
xRegress  
xProf  
xSlice  
xFind  
xVue  
xDiff



## Cleanscape Testwise process

CLEANSCAPE  
Testwise

**xAtac**

xRegress

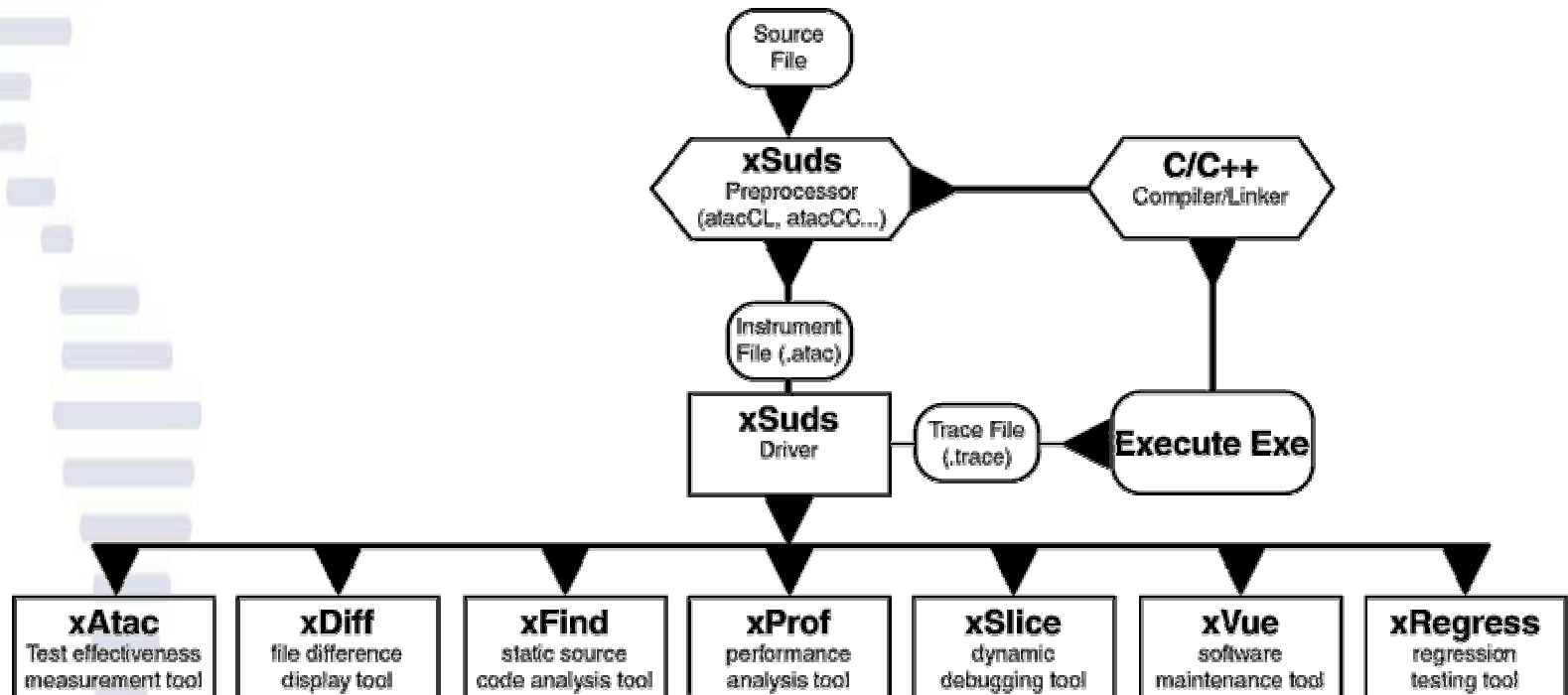
xProf

xSlice

xFind

xVue

xDiff



### xSuds Software Visualization and Analysis Process

# Cleanscape Testwise xAtac

CLEANSCAPE  
Testwise

xAtac

xRegress

xProf

xSlice

xFind

xVue

xDiff

## ◆ Test effectiveness measurement tool

- ◆ Determines how much of your code is currently tested
- ◆ Facilitates test creation
- ◆ Determines what is missing
- ◆ Identifies redundant test cases
- ◆ Determines whether product testers are finding bugs that have been found and fixed by developers
- ◆ Improves the software testing process

The screenshot shows the xAtac tool interface. At the top, there's a menu bar with 'File', 'Tool', 'Options', 'Summary', 'TestCases', 'Update', 'GoBack', and 'Help'. Below the menu is a toolbar with buttons numbered 1 through 8. The main area is a code editor displaying C code for a date parser. The code includes comments and function logic for parsing dates like 'MM/DD/YYYY'. At the bottom, there's a status bar with the xAtac logo and a table showing file, line, coverage, and highlighting information.

File:	Line:	Coverage:	Highlighting:
cmp2.c	121 of 151	block	all prioritized

## Coverage testing with xAtac

CLEANSCAPE  
Testwise

The screenshot displays the xAtac coverage testing interface. At the top, a navigation bar contains buttons for File, Tool, Options, Summary, TestCases, Update, GoBack, and Help. Below this, a series of colored blocks (0-8) represent different code segments. The main code editor shows a C program with various blocks highlighted in different colors (white, yellow, green, red). A blue callout box points to white-highlighted code, stating: "Code in white has already been covered by a test case and covering it again will not add new coverage". A red callout box points to a red-highlighted block, stating: "Covering this red block guarantees the execution of at least 8 additional blocks." The bottom status bar shows the xATAC logo, the file name (cmp2.c), the current line (121 of 151), the coverage level (block), and the highlighting mode (all prioritized).

File Tool Options Summary TestCases Update GoBack Help

0 1 2 3 4 5 6 7 8

```
do ++p; while (isalpha(*p));
break;
case 'o':
    month = 11;
    do ++p; while (isalpha(*p));
    break;
default:
    return -1;
}

value = 0; nDigits = 0;
while (isdigit(*p)) {
    ++nDigits;
    value = value * 10 + *p++ - '0';
}

if (delim == '-') {
    if (value < 1 || value > 31 || nDigits > 2)
        return -1;
    day = value;
} else {
    if (nDigits == 2)
        year = 1900 + value;
    else if (nDigits == 4)
        year = value;
    else return -1;
}

return year * 10000 + month * 100 + day;
}
```

Code in white has already been covered by a test case and covering it again will not add new coverage

Covering this red block guarantees the execution of at least 8 additional blocks.

xATAC

File: cmp2.c Line: 121 of 151 Coverage: block Highlighting: all prioritized

# Coverage testing with xAtac

CLEANSCAPE  
Testwise

File	Tool	Options	Summary	TestCases	Update	GoBack	Help
0	1	2	3	4	5	6	7
8							

do ++p; while (isalpha(\*p));

break;

case 'o':

month = 11;

do ++p; while (isalpha(\*n));

break;

default:

return -1;

}

value = 0; nDigits = 0;

while (isdigit(\*p)) {

++nDigits;

value = value \* 10 + \*p++ - '0';

}

if (delim == '-' ||

if (value < 0 || value > 31 || nDigits > 4)

return

day = value

} else {

if (nDigits

year = 1900 + value;

else if (nDigits == 4)

year = value;

else return -1;

}

return year \* 10000 +

100 + day;

}

true

false

dismiss

true

false

dismiss

true

false

dismiss

true

false

dismiss

Covering either *true* or *false* branch guarantees the execution of at least another 8 branches.

χATAC

File: cmp2.c

Line: 121 of 151

Coverage: decision

Highlighting: all prioritized

xAtac  
xRegress  
xProf  
xSlice  
xFind  
xVue  
xDiff

# Cleanscape Testwise xRegress

CLEANSCAPE  
Testwise

xAtac

**xRegress**

xProf

xSlice

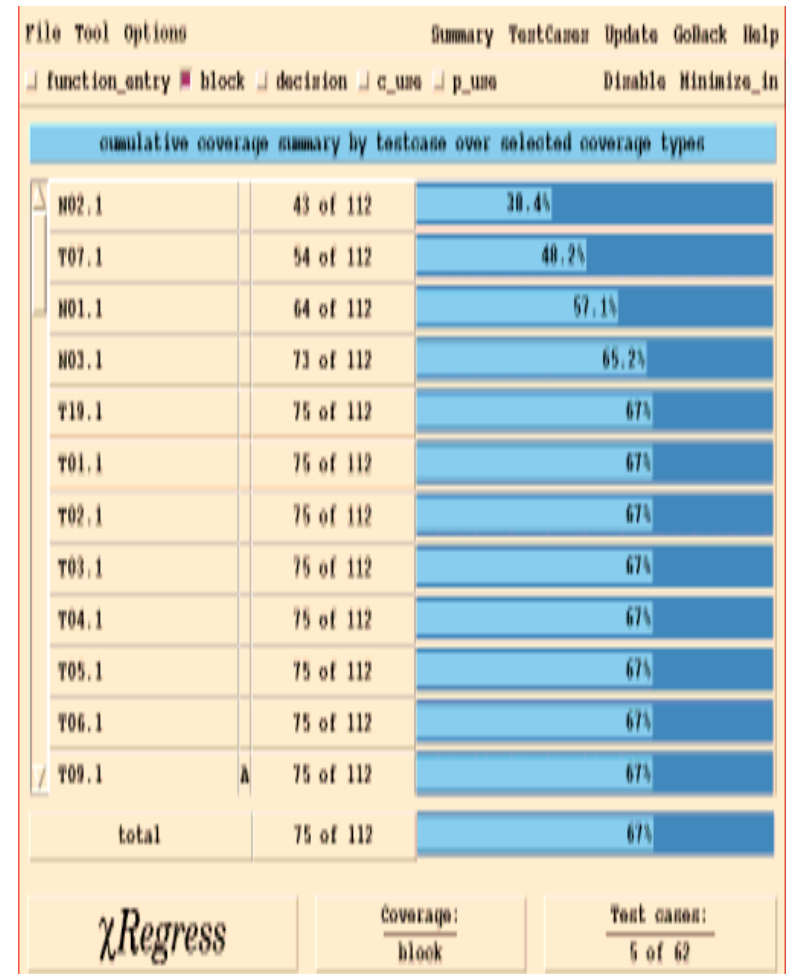
xFind

xVue

xDiff

## ◆ Intelligent regression testing tool

- ◆ Significantly reduce regression test costs
- ◆ Determine whether you are spending excessive resources in regression testing
- ◆ Understand how to select effective regression tests



## Test set minimization with xRegress

CLEANSCAPE  
Testwise

xAtac

xRegress

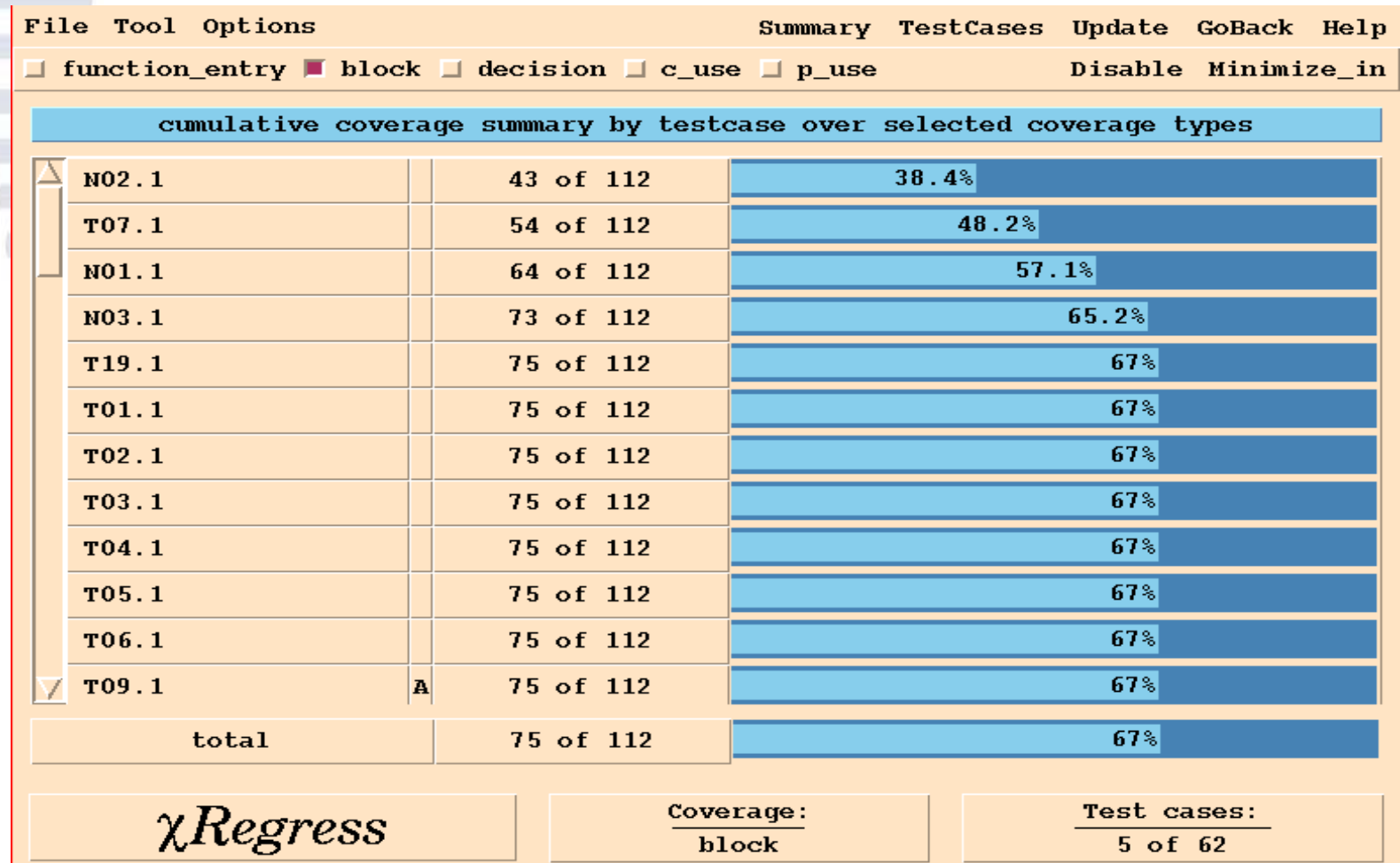
xProf

xSlice

xFind

xVue

xDiff



## Cleanscape Testwise xProf

CLEANSCAPE  
Testwise

### ◆ Performance analysis tool

- ◆ Identify performance bottlenecks visually
- ◆ Improve the performance of your program
- ◆ Identify which part of the program slows execution
- ◆ Visualize the most frequently executed pieces in code
- ◆ Develop repeatable performance measurements

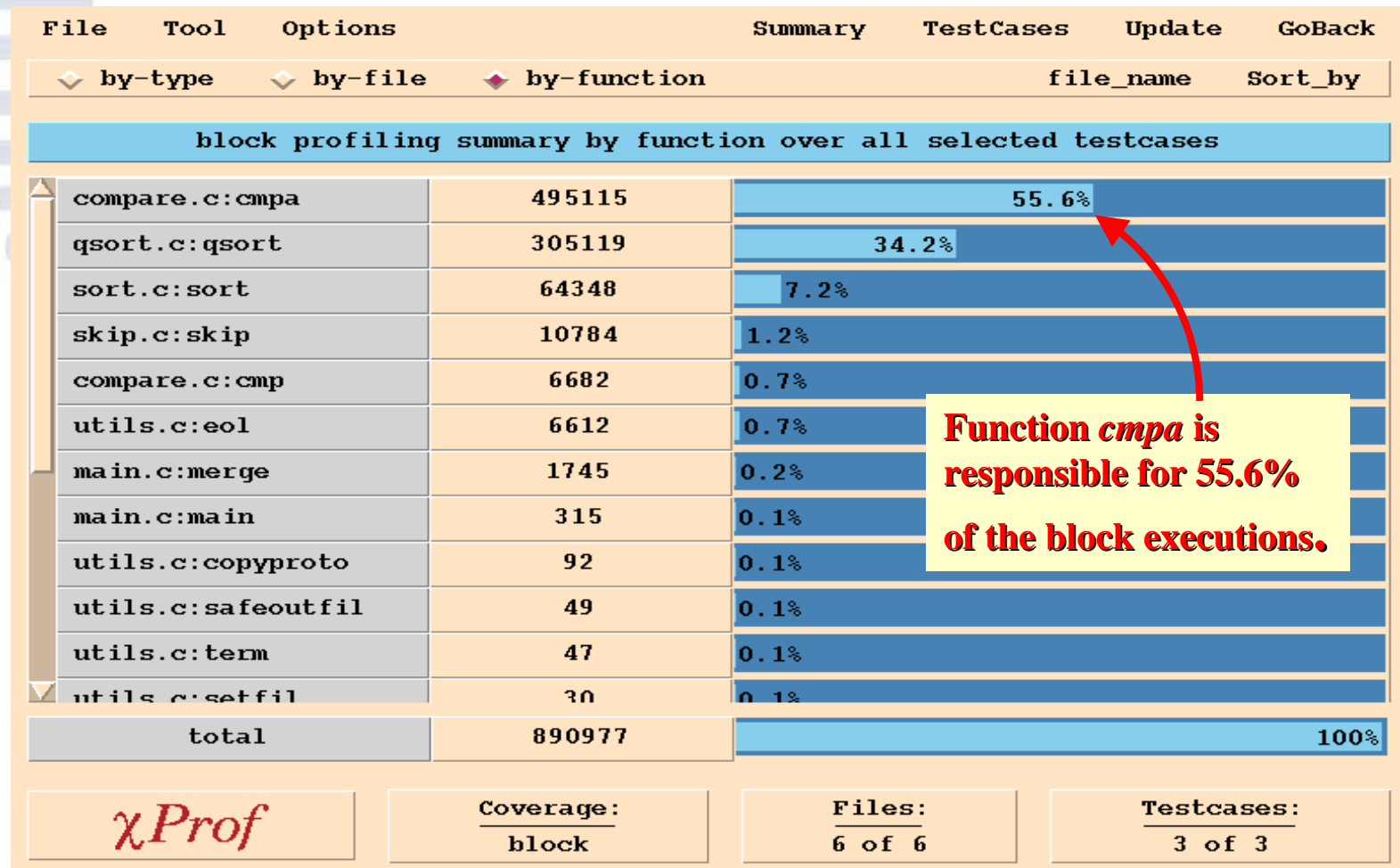
File	Tool	Options	Summary	Testcases	Update	GoBack
by-type	by-file	by-function	file_name	Sort_by		
block profiling summary by function over all selected testcases						
compare.c:cmpa	495115	55.6%				
qsort.c:qsort	305119	34.2%				
sort.c:sort	64348	7.2%				
skip.c:skip	10784	1.2%				
compare.c:cmp	6682	0.7%				
util.c:ool	6612	0.7%				
main.c:merge	1745	0.2%				
main.c:main	315	0.1%				
util.c:copyproto	92	0.1%				
util.c:safooutfil	49	0.1%				
util.c:term	47	0.1%				
util.c:netfil	30	0.1%				
total	890977	100%				
<div> <div>xProf</div> <div>Coverage: block</div> <div>Files: 6 of 6</div> <div>Testcases: 3 of 3</div> </div>						



# Identifying frequently executed code with xProf

CLEANSCAPE  
Testwise

xAtac  
xRegress  
**xProf**  
xSlice  
xFind  
xVue  
xDiff



# Identifying frequently executed code with xProf

CLEANSCAPE  
Testwise

xAtac  
xRegress  
**xProf**  
xSlice  
xFind  
xVue  
xDiff

File	Tool	Options	Summary	TestCases	Update	GoBack	Help
0	1	15403	30805	46207	61609	77011	92412
							107813

```

while(ignore[*pb])
    pb++;
if(pa>=la || *pa=='\n')
    if(pb<lb && *pb!='\n')
        return(fp->rflg);
    else continue;
if(pb>=lb || *pb=='\n')
    return(-fp->rflg);
if((sa = code[*pb++] - code[*pa++]) == 0)
    goto loop;
return(sa*fp->rflg);
}
if(uflg)
    return(0);
return(cmpa(i, j));
}

cmpa(pa, pb)
register char *pa, *pb;
{
    while(*pa == *pb) {
        if(*pa++ == '\n')
            return(0);
        pb++;
    }
    return(
        *pa == '\n' ? fields[5].rflg:
        *pb == '\n' ? -fields[0].rflg:
        *pb > *pa ? fields[0].rflg:
        -fields[0].rflg
    );
}
  
```

Code in red is executed 107813 times

Code in green is executed 30805 times

**xProf**

File: compare.c      Line: 88 of 121      Coverage: block      Highlighting: all prioritized

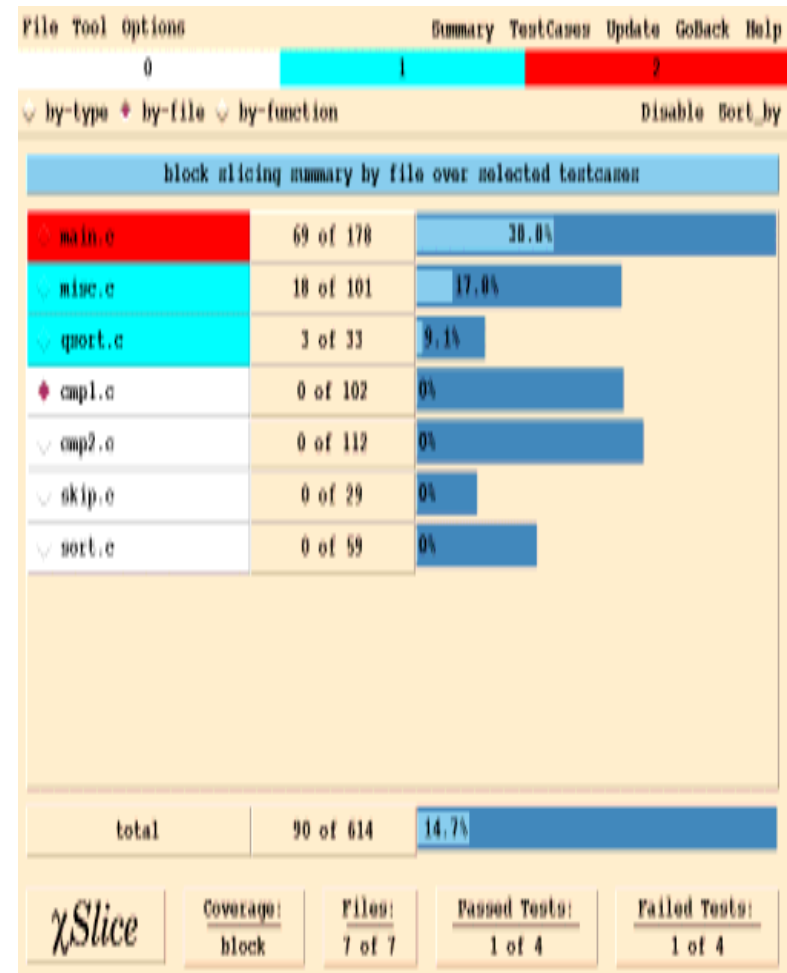
## Cleanscape Testwise xSlice

CLEANSCAPE  
Testwise

### ◆ Dynamic debugging tool

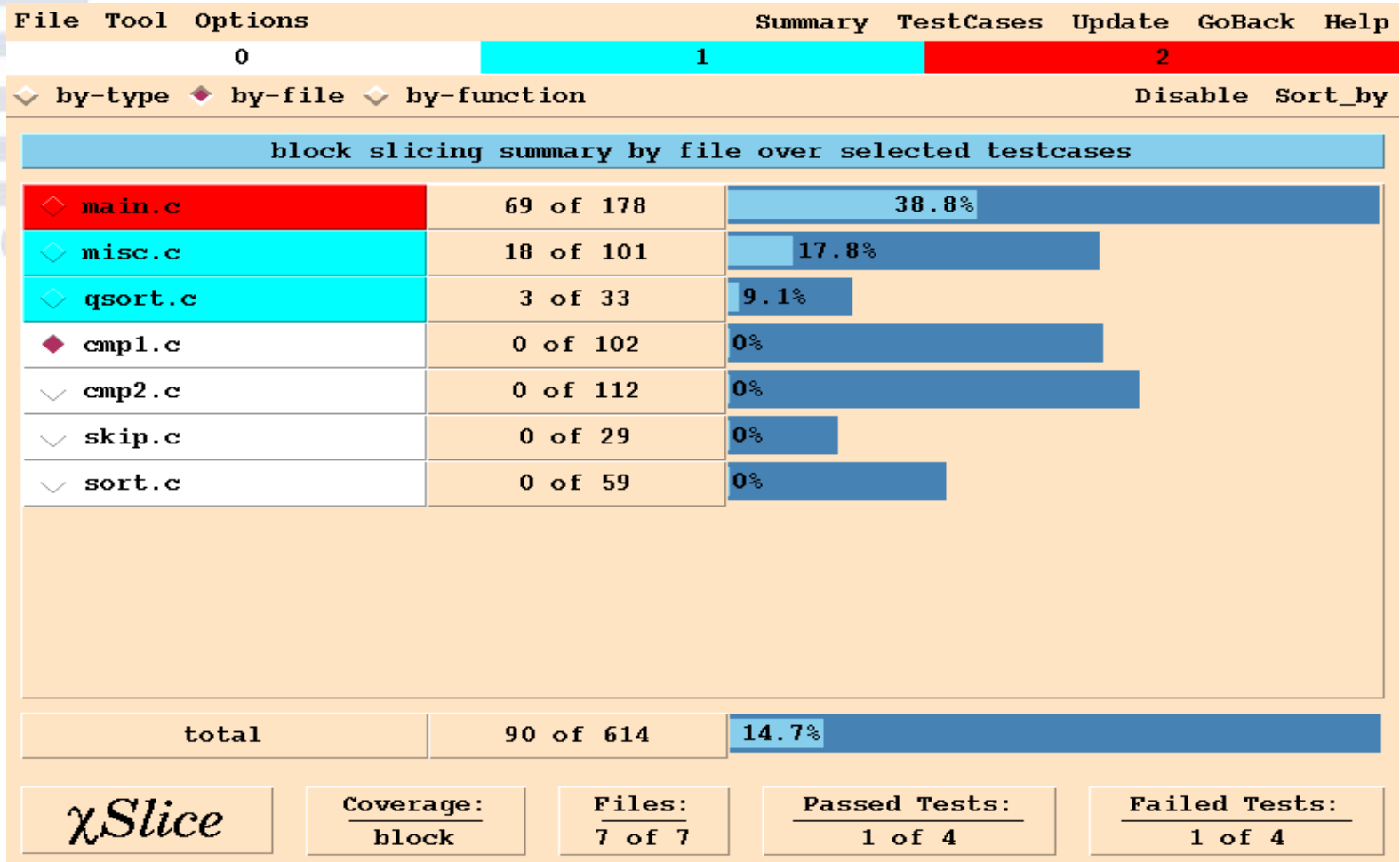
xAtac  
xRegress  
xProf  
**xSlice**  
xFind  
xVue  
xDiff

- ◆ Pinpoint the location of faults from failures
- ◆ Locate bugs quickly
- ◆ Narrow down bugs to files, then functions, then lines of code



# Localizing programming bugs with xSlice

CLEANSCAPE  
Testwise



xAtac  
xRegress  
xProf  
**xSlice**  
xFind  
xVue  
xDiff

# Localizing programming bugs with xSlice

CLEANSCAPE  
Testwise

File Tool Options Summary TestCases Update GoBack Help

0 1 2

```

p = (struct merg *)lspace;
j = 0;
for(i=a; i < b; i++) {
    f = setfil(i);
    if(f == 0)
        p->b = stdin;
    else if((p->b = fopen(f, "r")) == NULL)
        cant(f);
    ibuf[j] = p;
    if(!rline(p))    j++;
    p++;
}
}

i = j;
qsort((char **)ibuf, (char **) (ibuf+i));
l = 0;
while(--i) {
    cp = ibuf[i]->l;
    if(*cp == '\0') {
        l = 1;
        if(rline(ibuf[i])) {
            k = i;
            while(++k < j)
                ibuf[k] = ibuf[i];
            j--;
        }
    }
}

```

Code in blue is executed by the failed test AND the successful one

Code in red is executed by the failed test BUT NOT the successful one

Code in white is not executed By any of the failed tests

*xSlice* File: main.c Line: 152 of 240 Coverage: block Highlighting: all prioritized

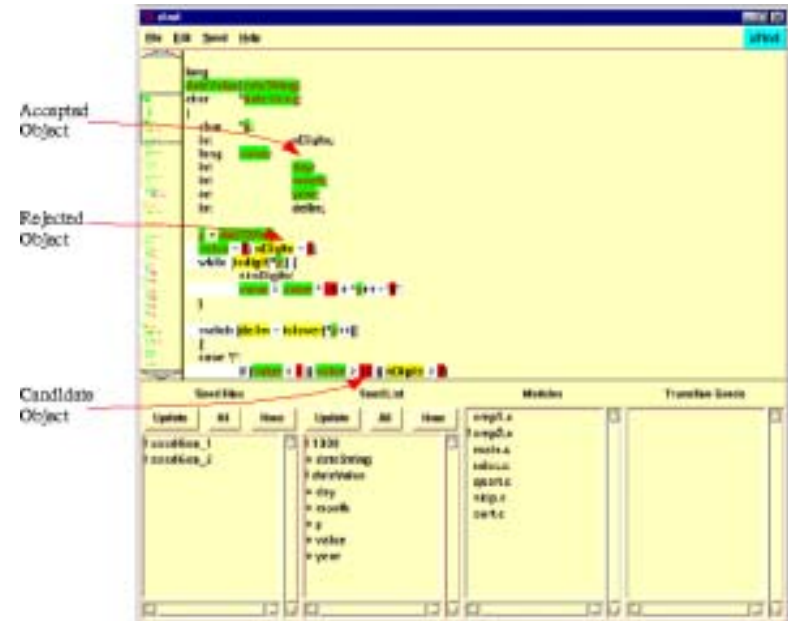
xAtac  
xRegress  
xProf  
**xSlice**  
xFind  
xVue  
xDiff

## Cleanscape Testwise xFind

CLEANSCAPE  
Testwise

### ◆ Static source code analysis tool

- ◆ xAtac
- ◆ xRegress
- ◆ xProf
- ◆ xSlice
- ◆ **xFind**
- ◆ xVue
- ◆ xDiff
- ◆ Trace static program dependencies
- ◆ Identify data sensitivities in your applications
- ◆ Analyze difficult languages like C and C++
- ◆ Identify redundant test cases

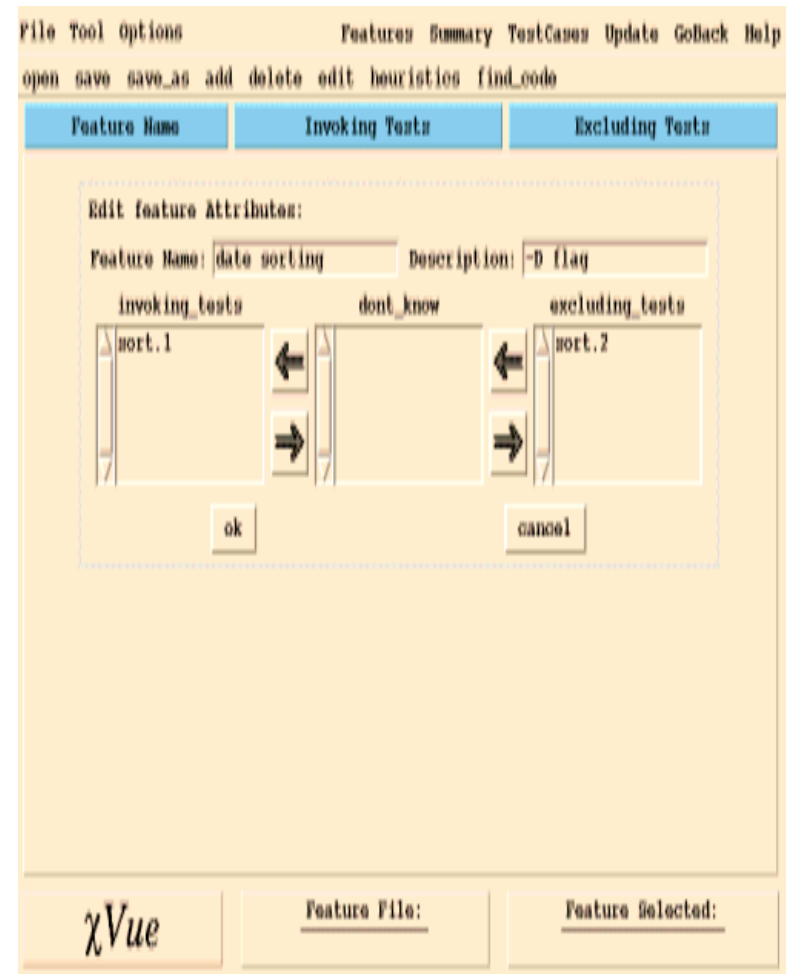


# Cleanscape Testwise xVue

CLEANSCAPE  
Testwise

## ♦ Software maintenance tool

- ♦ xAtac
- ♦ xRegress
- ♦ xProf
- ♦ xSlice
- ♦ xFind
- ♦ **xVue**
- ♦ xDiff
- ♦ See where user functions are implemented in the software
- ♦ Know where features are implemented
- ♦ Visualize features in code
- ♦ Reduce time involved with resolving reports



## Visualizing features in code with xVue

CLEANSCAPE  
Testwise

File Tool Options Features Summary TestCases Update GoBack Help

open save save\_as add delete edit heuristics find\_code

Feature Name	Invoking Tests	Excluding Tests
<b>Edit feature Attributes:</b>		
Feature Name: <u>date sorting</u>	Description: <u>-D flag</u>	
invoking_tests	dont_know	excluding_tests
sort.1		sort.2
←	←	←
→	→	→
ok		cancel

*sort.1* is an invoking test &  
*sort.2* is an excluding test

xVue Feature File: Feature Selected:

xAtac  
xRegress  
xProf  
xSlice  
xFind  
**xVue**  
xDiff



## Visualizing features in code with xVue

CLEANSCAPE  
Testwise

File Tool Options Features Summary TestCases Update GoBack Help

xAtac  
xRegress  
xProf  
xSlice  
xFind  
**xVue**  
xDiff

```

    );
}
long
cmpd(dString)
char    *dString;
{
    char    *p;
    int     nDigits;
    long    value;
    int     day;
    int     month;
    int     year;
    int     delim;

    p = dString;
    value = 0; nDigits = 0;
    while (isdigit(*p)) {
        ++nDigits;
        value = value * 10 + *p++ - '0';
    }

    switch (delim = tolower(*p++))
    {
    case '/':
        if (value < 1 || value > 12 || nDigits > 2)
            return -1;
        month = value;
        break;
    case '.':

```

Code in red is *uniquely* related to the *date sorting* feature.

xVue

File: cmp2.c	Line: 16 of 151	Coverage: block	Highlighting: highest weight
-----------------	--------------------	--------------------	---------------------------------

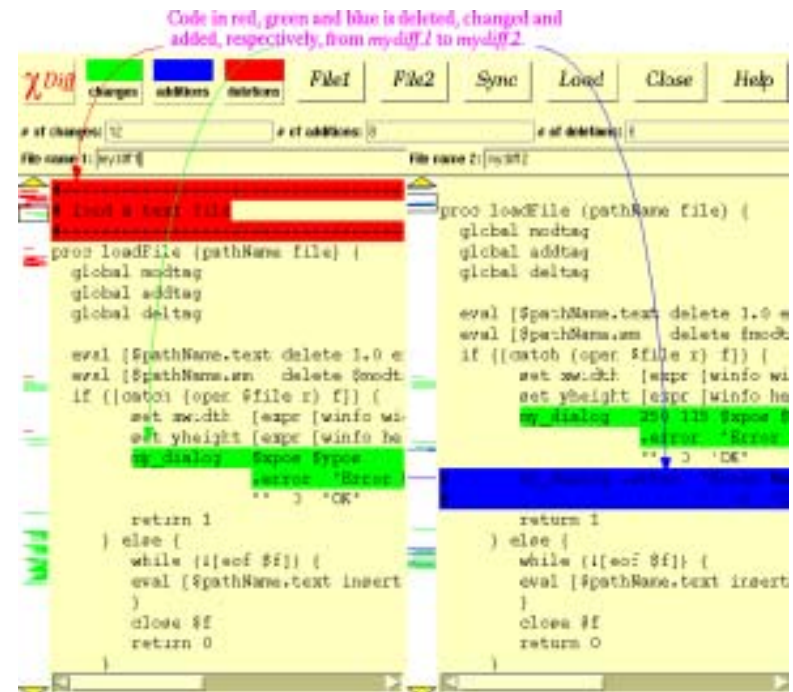
## Cleanscape Testwise xDiff

CLEANSCAPE  
Testwise

### ♦ File difference display tool

- ♦ Display program differences
- ♦ Visualize the difference between two files
- ♦ Compare versions of code, documents, data, computer output

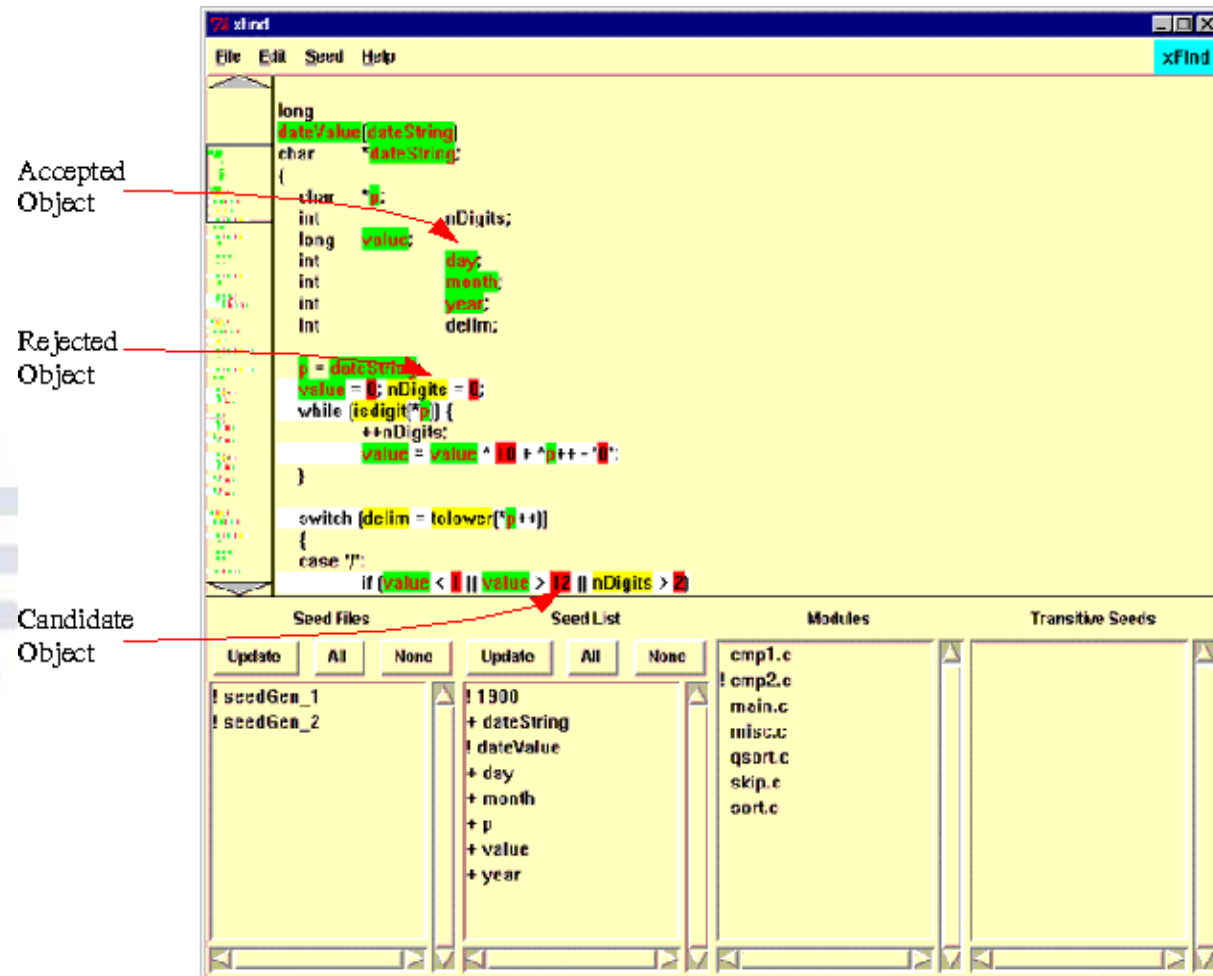
xAtac  
xRegress  
xProf  
xSlice  
xFind  
xVue  
**xDiff**



# Identifying problems in source code with xDiff

CLEANSCAPE  
Testwise

xAtac  
xRegress  
xProf  
xSlice  
xFind  
xVue  
**xDiff**



## Displaying program differences with xDiff

CLEANSCAPE  
Testwise

xAtac  
xRegress  
xProf  
xSlice  
xFind  
xVue  
**xDiff**

Code in red, green and blue is deleted, changed and added, respectively, from mydiff.1 to mydiff.2.

```

File name 1: mydiff1
File name 2: mydiff2

# load a text file
proc loadFile {pathName file} {
    global modtag
    global addtag
    global deltag

    eval [$pathName.text delete 1.0 e
    eval [$pathName.sm delete $modt
    if {[catch {open $file r} f]} {
        set xwidth [expr [wininfo wi
        set yheight [expr [wininfo he
        my_dialog 250 115 $xpos $
        .error 'Error
        "" 0 'OK'
    } else {
        while {[eof $f]} {
            eval [$pathName.text insert
        }
        close $f
        return 0
    }
}

proc loadFile {pathName file} {
    global modtag
    global addtag
    global deltag

    eval [$pathName.text delete 1.0 e
    eval [$pathName.sm delete $modt
    if {[catch {open $file r} f]} {
        set xwidth [expr [wininfo wi
        set yheight [expr [wininfo he
        my_dialog 250 115 $xpos $
        .error 'Error
        "" 0 'OK'
    } else {
        while {[eof $f]} {
            eval [$pathName.text insert
        }
        close $f
        return 0
    }
}
  
```

