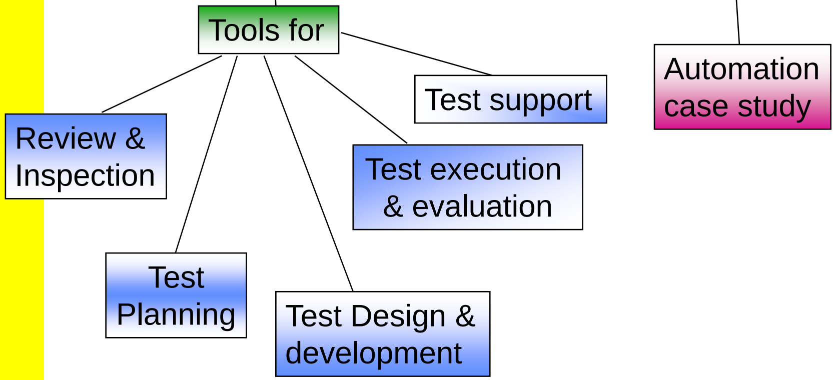
Course Structure

1. Software Quality Assurance
2. Testing Fundamentals
3. Code-based Techniques
4. Specification-based Techniques
5. Inspection Technique

6 Test Tools

7. Measuring Software Quality
8. TDD

Test Tools



Tools for

Review & Inspection

Test Planning

Test Design & development

Test execution & evaluation

Test support

Automation case study

A fool with a tool is still a fool!

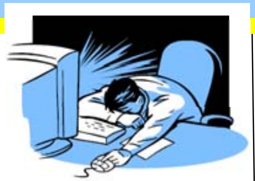
Learning Objectives

- identify the advantages of automated testing
- describe testing tools for supporting:
  - Reviews and inspections
  - Test planning
  - Test design and development
  - Test execution and evaluation
  - Test support
- identify some commercial test tools
- learn from a case study

Opening Questions

1. Name some testing tools being used in your organization.
2. Identify when to use test tools during the development life cycle.

Problems with Manual Test



- ☹ Costly
- ☹ Slow
  - ☹ inputs entered at human speed
  - ☹ testing during work hours
  - ☹ outputs checks at human speed
- ☹ Error-prone
  - ☹ inexact repetition of tests
  - ☹ inaccurate results checking

Impact of “Bad” Test Technology

- Increased failures due to poor quality
- Increased software development costs
- Increased time to market due to inefficient testing
- Increased market transaction costs

Statistics

Introduction

Tools for Review & insp

Test planning

Test develop

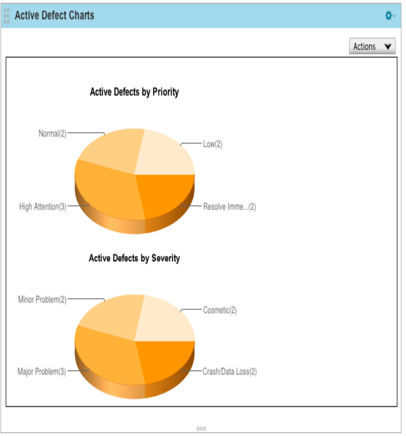
Test exe. & ev

Test support

Automate eg

Many studies indicate that

- ***Around 40% of all software defects could have been detected using static analysis tools*** (e.g., inspection tool, dataflow analysis tool, etc).
- ***These tools help in reducing the product failures and reducing the time to market.***



Active Defect Charts

What Activities can be Automated?

Introduction

Tools for Review & insp

Test planning

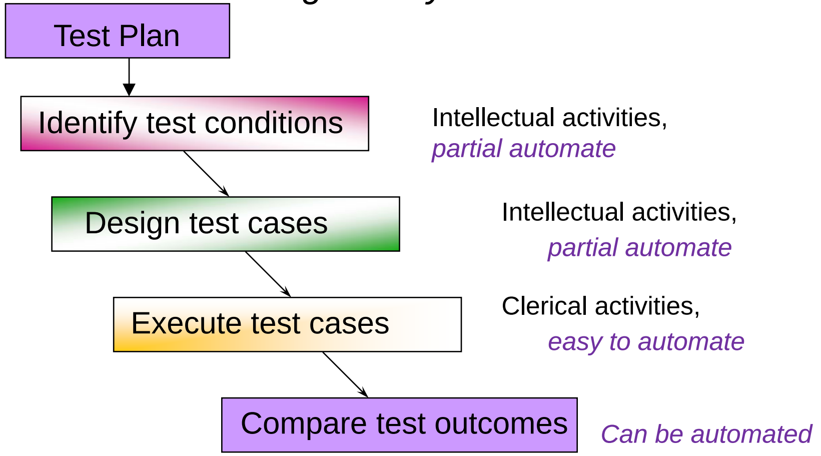
Test develop

Test exe. & ev

Test support

Automate eg

Testing Life Cycle



Intellectual activities, *partial automate*

Intellectual activities, *partial automate*

Clerical activities, *easy to automate*

Can be automated

Automation Experience (1)

Introduction

Automation

Tools for Review & insp

Test planning

Test develop

Test exe. & ev

Test support

Automate eg

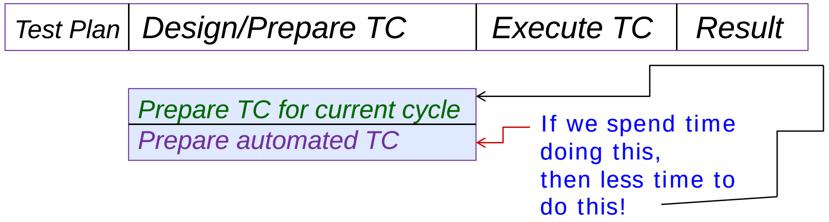
Test automation is the use of software to control the execution of tests, the comparison of actual outcomes to predicted outcomes, the setting up of test preconditions, and other test control and test reporting functions.

That is, using tools!

- >50% of the companies require more than 4 weeks to implement automated testing.
- Implementing automated testing requires technical skills and knowledge many Test Groups do not have.
- Require **significant up-front cost** to train staff and to develop automated tests.
- Some reports it takes **2-10 times** the effort to automate a test compared to running it manually!!

From Experience (2)

Introduction
Automation
Tools for Review & insp
Test planning
Test develop
Test exe. & ev
Test support
Automate eg



Test Plan	Design/Prepare TC	Execute TC	Result

Prepare TC for current cycle

Prepare automated TC

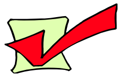
If we spend time doing this, then less time to do this!

- Automated tests generally cost more to maintain.
- Automated tests may not find many defects. *Why?*

  A test is most likely to find a defect the first time it is run.
- If we spend time automating the test cases rather than testing during the early part, we will delay finding defects!
- Do not use tools before we have a testing process!

Advantages of Automating Testing

- Improve the repeatability of the tests by minimizing the possibility of human errors during testing, using exactly the same inputs & same sequence.
- Execute more tests for the limited time given for testing
- Improve productivity



Automation is not Worthwhile if ...

- ➔ Testing is informal and mostly ad-hoc
- ➔ user interface is incomplete, change often
- ➔ there will be only a few application builds, or application is one-time only
- ➔ manually testing entire application provides results quickly enough and manual test execution is not tedious
- ➔ need to test the application immediately and provide test results NOW.

Guidelines on Test Automation

Introduction
Automation
Tools for Review & insp
Test planning
Test develop
Test exe. & ev
Test support
Automate eg

- ➔ Do not automate tests we will rarely use (why? Costly to build the automated tests). Make sure test automation pay off.
- ➔ Test suite should be flexible (same test scripts can be reused)
- ➔ Test suite should be well documented.
- ➔ Test suite development requires careful thought
- ➔ Specify the dependencies between tests

Be Careful

"Men have become the tools of their tools"

David Thoreau

What to Automate?

Introduction
Automation
Tools for Review & insp
Test planning
Test develop
Test exe. & ev
Test support
Automate eg

- Tests that are run many times, such as "smoke" (build verification) tests, regression tests
- Testing core (critical) functionality
- Testing high priority/data intensive areas
- Mundane tests (tests that include many simple and repetitive steps)
- Test that would be impossible (or prohibitively expensive) to perform manually (e.g., simulating 1000 multi-user accesses – stress test, concurrency, performance tests)

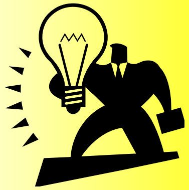
"Automating chaos just gives faster chaos"

What Test Cases to Automate?

Introduction
Automation
Tools for Review & insp
Test planning
Test develop
Test exe. & ev
Test support
Automate eg

- Can the test sequence of actions be defined?
- Is it useful to repeat the sequence of actions many times? Examples: acceptance tests, compatibility tests, performance tests, and regression tests.
- Is it necessary to repeat the sequence of actions many times?
- Is it possible to automate the sequence of actions?
- Is it possible to "semi-automate" a test? Automating portions of a test can speed up test execution time.
- Is the behavior of the software under test the same with automation as without? (for performance testing.)
- Are we testing non-user interface (non-UI) aspects of the program? *Almost all non-UI functions can be automated tests.*
- Do we need to run the same tests on multiple hardware configurations?

How many software testers does it take to change a light bulb?

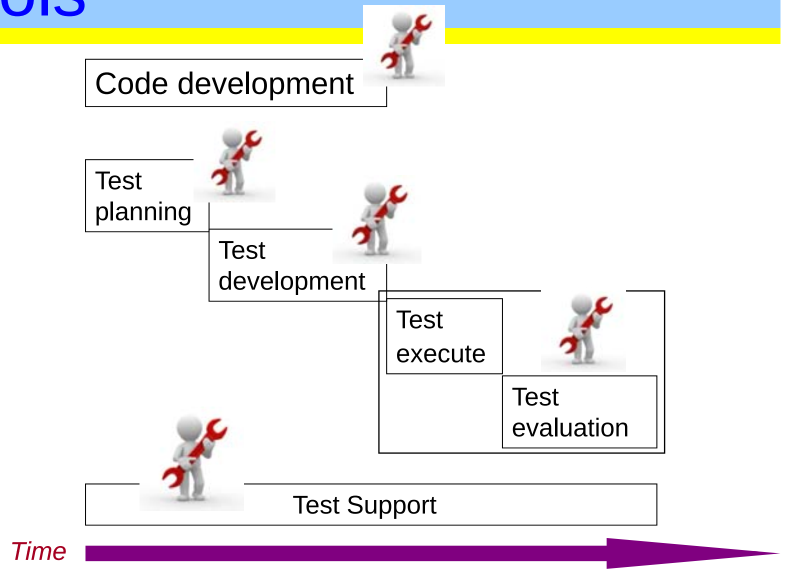


None.

“Testers just recognized darkness, fixing it is someone else's problem."

Tools

Introduction
Tools for Review & insp
Test planning
Test develop
Test exe. & ev
Test support
Automate eg



Tools for Reviews and Inspections

Introduction
Tools for Review & insp
Test planning
Test develop
Test exe. & ev
Test support
Automate eg

Assist with performing reviews, walkthroughs, and inspections of requirements, functional design, detailed design, and code.

1. **Complexity analyser**
   Calculate complexity metrics: McCabe Cyclomatic number, LOC, etc.
2. **Code comprehension**
   help to understand dependencies, trace program logic, view graphical representations of the program, identify dead code
3. **Standard enforcer** (**Syntax and semantic analysis)**
   Perform error checking to find errors that a compiler would miss. Language dependent.

Complexity Analyser - Static Analysis

Introduction
Tools for Review & insp
Complexity analyzer
Standard enforcer
Rule checker
Test planning
Test develop
Test exe. & ev
Test support
Automate eg

The complexity metric is used

(1) to identify the complexity of the program, and

(2) to show how much of the logic has been exercised during testing.

⌘ Understanding the complexity can be used in both evaluating the test and developing a test plan.

⌘ Example Metrics:

- ✓ Cyclomatic complexity
- ✓ LOC
- ✓ Fan out
- ✓ Halstead software metrics

Complexity Analyser

Introduction
Tools for Review & insp
Complexity analyzer
Standard enforcer
Rule checker
Test planning
Test develop
Test exe. & ev
Test support
Automate eg

Some analyser provides a graphical display such as a ***call graph***, showing the usage of various functions.

Advantage:

☺ Help in designing & evaluating tests, & in simplifying logic by identifying highly complex conditions

☺ Help to identify error prone, difficult to test modules

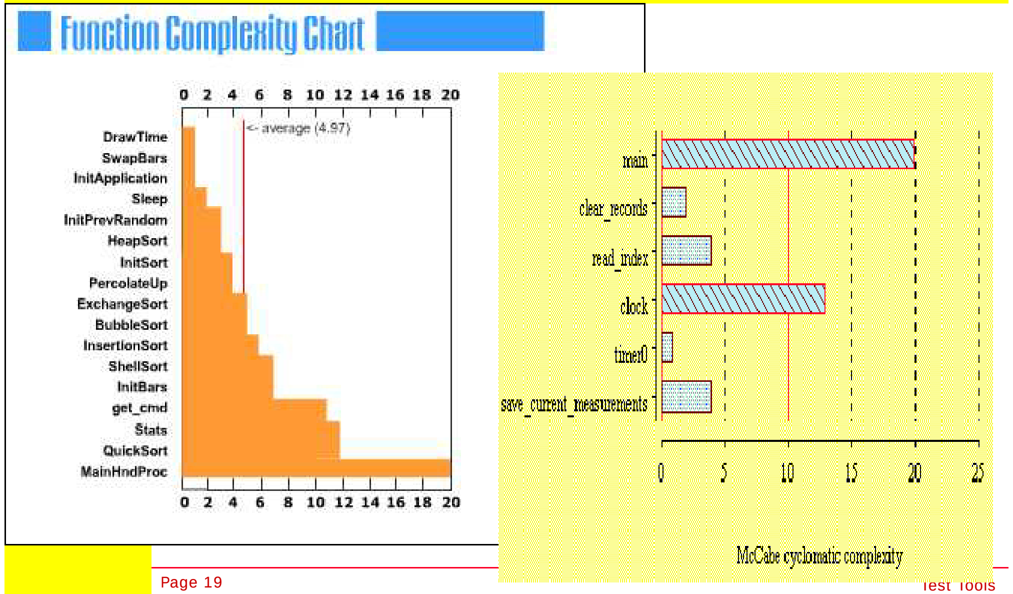
Disadvantage:

☹ If the complexity metrics are not predictors of actual results, they can cause inappropriate decisions to be made and actions to be taken.

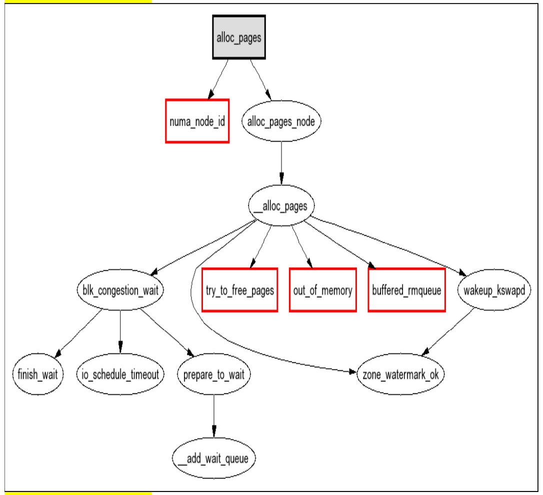
Example Tools:

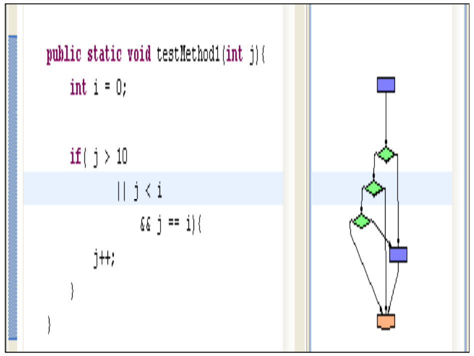
- Panorama, http://www.softwareautomation.com/
- STW/Advisor from Software Research, http://www.soft.com/TestWorks/

Example complexity



Example call graph generation





```
public static void testMethod1(int j){
    int i = 0;

    if( j > 10
            || j < i
                && j == i){
        j++;
    }
}
```

Standard Enforcers

Introduction

Tools for
Review & insp

Complexity
analyzer
Standard
enforcer
Rule checker

Test planning

Test develop

Test exe. & ev

Test support

Automate eg

⌘ Enforce coding standards by looking at the program statements.

⌘ Enhance the readability of the programs.

☺ **Advantage:** Helpful as a filter protecting against completely unreadable codes

☹ **Disadvantage:** Programmers resist and view unimportance of program format.

Example: CodeWizard (1)

http://www.parasoft.com/products/

Introduction

Tools for
Review & insp

Complexity
analyzer
Standard
enforcer
Rule checker

Test planning

Test develop

Test exe. & ev

Test support

Automate eg

- Automatically performs static analysis on C++ code.
- Enforces over 70 C++ **coding standards**.
- Provides coding suggestions on better coding constructs.
- Makes the code easier for other developers to understand and use.
- Pinpoints exact location of programming and design violations.
- Checks standards that are too sophisticated to be enforced by compilers.

CodeWizard (2)

Rules enforced by **CodeWizard 3.0** on C++

Prefer iostream.h to stdio.h
Use new and delete instead of malloc and free
Use the same form in corresponding calls to new and delete
Call delete on pointer members in destructors
Check the return value of new
Adhere to convention when writing new
Avoid hiding the global new
Write delete if you write new
Define a copy constructor and assignment operator for classes with dynamically allocated memory
Prefer initialization to assignment in constructors
List members in an initialization list in the order in which they are declared
Make destructors virtual in base classes
Have operator= return a reference to this*
Assign to all data members in operator=
Check for assignments to self in operator=
Differentiate among member functions, global functions, and friend functions
Avoid data members in the public interface
Pass and return objects by reference instead of by value
Don't try to return a reference when you must return an object
Avoid overloading on a pointer and a numerical type
Avoid returning "handlesl" to internal data from const member functions
Avoid member functions that return pointers or reference to members less accessible than themselves
Never return a reference to a local object or a dereferenced pointer initialized by new with the function

Never redefine an inherited nonvirtual function
Never redefine an inherited default parameter value
Avoid casts down the inheritance hierarchy
Prefer C++ style casts
Be wary of user-defined conversion functions
Distinguish between prefix and postfix forms of increment and decrement operators
Never overload &&,||, or ,
Consider using op= instead of stand=alone op
Understand the costs of virtual functions, multiple inheritance, virtual base classes, and RTTI
Limiting the number of objects of a class
Avoid calling virtual functions from constructors and destructors
Avoid using "..." in function parameter list
Do not declare protected data members
Do not declare the constructor or destructor to be inline
Declare at least one constructor to prevent the compiler from doing so
Pointers to functions should use a typedef
Never convert a const to a non-const
Do not use the ?: operator
Each class must declare the public, protected, and private sections in that order
In the public section entities shall be declared in the following order: Constructors, Destructors, Member functions, Member conversion functions, Enumerations, and other
In the protected section entities shall be declared in the following order: Constructors, Destructors, Member functions,Member conversion functions, Enumerations, and other
In the private section entities shall be declared in the following order: Constructors, Destructors, Member functions, Member conversion functions, Enumerations, and other

CodeWizard (3)

Introduction

Tools for
Review & insp

Complexity
analyzer
Standard
enforcer
Rule checker

Test planning

Test develop

Test exe. & ev

Test support

Automate eg

If a function has no parameters, use () instead of (void)
If, else, while and do statements shall be followed by a block, even if it is empty
If a block is single statement, enclose it in braces
Whenever a global variable or function is used, use the :: operator
Do not use public data members
If a class has any virtual function it shall have a virtual destructor
Public member functions shall return const handles to member data
A class that has pointer members shall have an operator and a copy constructor
If a subclass implements a virtual function, use the virtual keyword
Member functions shall not be defined in the class definition
Ellipses shall not be used
Functions shall explicitly declare their return types
A pointer to a class shall not be converted to a pointer of a second class unless it inherits from the second
A pointer to an abstract class shall not be converted to a pointer that inherits from that class
Do not use the friend mechanism
When working with float or double values, use <= and => instead of ==
Do not overload functions within a template class
Do not define structs that contain member functions
Do not directly access global data from a constructor
Do not use multiple inheritance
Initialize all variables
All pointers should always be initialized to zero
Always terminate a case statement with a break
Always provide a default branch for switch statements
Do not use the goto statement
Provide only one return statement in a function

Rule Checker (or Static Checker)

Introduction
Tools for Review & insp
Complexity analyzer
Standard enforcer
Rule checker
Test planning
Test develop
Test exe. & ev
Test support
Automate eg

- ⌘ Look for likely ***coding errors*** (dangerous constructs, maintenance and portability concerns) – known *error patterns*

  E.g., calling virtual functions from constructors, local variables hiding data members
- ⌘ Each rule is a set of instructions that queries a database and check if a potential error condition is present.
- **Assumption:** human fallibility is somewhat predictable.
- Programmers usually employ static checkers after compilation and before testing.

☺ **Advantage:** Compliment code inspections, freeing humans to focus on higher level problems.

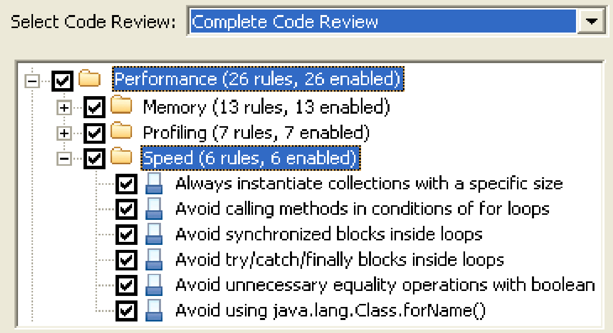
☹ **Disadvantage:** Usefulness depends highly on the rule set used by the tool. Cannot take all possible bugs into account!!

Example Rule Checker

Introduction
Tools for Review & insp
Complexity analyzer
Standard enforcer
Rule checker
Test planning
Test develop
Test exe. & ev
Test support
Automate eg

Rational Application Developer

- ⌘ Has many rule categories: a collection of code review rules that focus on a particular aspect of quality (e.g., ***performance***, design principles, globalization, J2EE best practices, J2SE best practices, naming conventions)



Select Code Review: Complete Code Review

- ☑ Performance (26 rules, 26 enabled)
  - ☑ Memory (13 rules, 13 enabled)
  - ☑ Profiling (7 rules, 7 enabled)
  - ☑ Speed (6 rules, 6 enabled)
    - ☑ Always instantiate collections with a specific size
    - ☑ Avoid calling methods in conditions of for loops
    - ☑ Avoid synchronized blocks inside loops
    - ☑ Avoid try/catch/finally blocks inside loops
    - ☑ Avoid unnecessary equality operations with boolean
    - ☑ Avoid using java.lang.Class.forName()

Another Example Rule checker

Introduction
Tools for Review & insp
Complexity analyzer
Standard enforcer
Rule checker
Test planning
Test develop
Test exe. & ev
Test support
Automate eg

FindBugs (http://findbugs.sourceforge.net) for Java

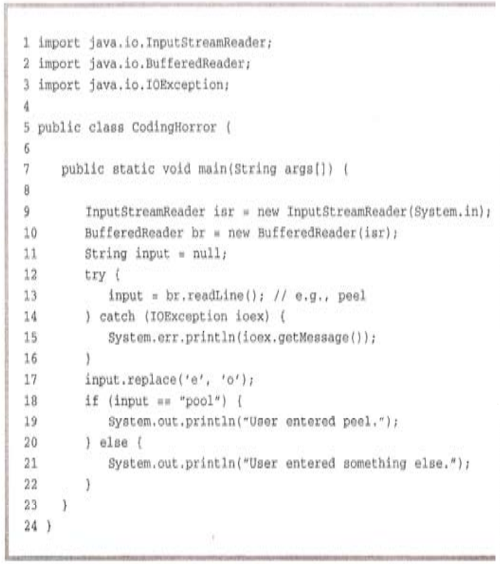
- This tool looks for 200 bug patterns
- It reads the program and construct some model of it, a kind of abstract representation that it can use for matching the ***error patterns*** they recognize.
- Also perform some kind of data-flow analysis, trying to infer the possible values that variables might have at certain points in the program

Example tools:

CodeReview http://www.microfocus.com/products/micro-focus-developer/devpartner/index.aspx

CodeWizard http://www.parasoft.com/products/

Example Code

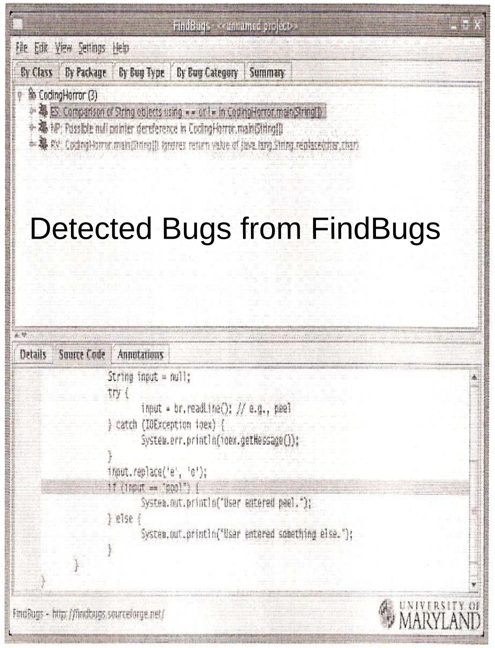


```
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;

public class CodingHorror {

   public static void main(String args[]) {

      InputStreamReader isr = new InputStreamReader(System.in);
      BufferedReader br = new BufferedReader(isr);
      String input = null;
      try {
         input = br.readLine(); // e.g., peel
      } catch (IOException ioex) {
         System.err.println(ioex.getMessage());
      }
      input.replace('e', 'o');
      if (input == "pool") {
         System.out.println("User entered peel.");
      } else {
         System.out.println("User entered something else.");
      }
   }
}
```



CodingHorror (3)
- ES: Comparison of String objects using == or != in CodingHorror.main(String[])
- NP: Possible null pointer dereference in CodingHorror.main(String[])
- RV: CodingHorror.main(String[]) ignores return value of java.lang.String.replace(char, char)

Detected Bugs from FindBugs

```
            String input = null;
            try {
                    input = br.readLine(); // e.g., peel
            } catch (IOException ioex) {
                    System.err.println(ioex.getMessage());
            }
            input.replace('e', 'o');
            if (input == "pool") {
                    System.out.println("User entered peel.");
            } else {
                    System.out.println("User entered something else.");
            }
      }
}
```

FindBugs - http://findbugs.sourceforge.net/

Example Bugs

Introduction

Tools for Review & insp

Complexity analyzer
Standard enforcer
Rule checker

Test planning

Test develop

Test exe. & ev

Test support

Automate eg

3 possible bugs:

- **Line 18**: the == operator compares object references, not the objects themselves. Unless the two strings we compare are stored in the same object, the comparison will fail.
- If, for some reason, the read operation at **line 13** fails, `input` will be null, and the program will try to call the method `replace` on a null object. (null pointer)
- Because strings are immutable in Java, replace does not modify the original string. Instead, it returns a new string with the results of any replacements it carries out. The program simply ignores the result string in **line 17**. The remainder of the program continues to use the original string. This bug is the most subtle of the 3 bugs.

Tools for Test Planning

Introduction

Tools for Review & insp

Test planning

Test develop

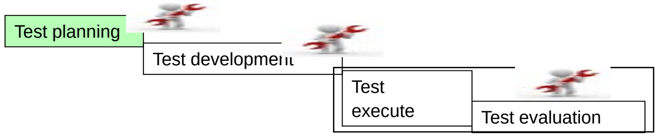
Test exe. & ev

Test support

Automate eg

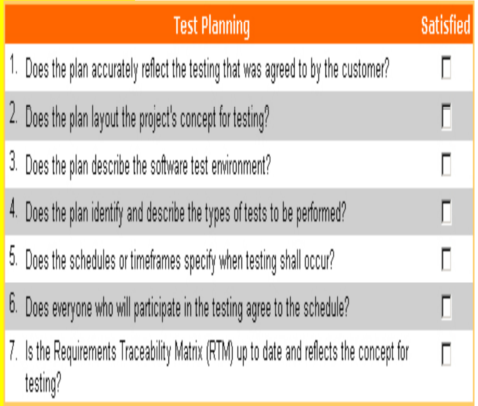
1. Templates for test plan documentation (see sample test plan template from IEEE)
2. Test schedule and staffing estimates
3. Complexity analyser
4. **Checklist on Test Plan**

We will not discuss the first 3 types.



Example Checklist: Test Planning

After drafting a test plan, use the checklist to ensure a high quality test plan.



Test Planning	Satisfied
1. Does the plan accurately reflect the testing that was agreed to by the customer?	□
2. Does the plan layout the project's concept for testing?	□
3. Does the plan describe the software test environment?	□
4. Does the plan identify and describe the types of tests to be performed?	□
5. Does the schedules or timeframes specify when testing shall occur?	□
6. Does everyone who will participate in the testing agree to the schedule?	□
7. Is the Requirements Traceability Matrix (RTM) up to date and reflects the concept for testing?	□

1. Have both white and black box test been specified?
2. Have all paths through the logic been tested?
3. Have test cases been identified and listed with what result is expected?
4. Has error handling been tested?
5. Are boundary conditions identified and tested?
6. Stress and performance testing?
7. Have unexpected inputs been considered?
8. Does the test plan contain a flow diagram representing roles, sources and destinations of data?
9. Does the test plan connect well with the feature as explained in the requirements document?
10. Does the test plan explain user actions in a way that is well understood?
11. Does the test plan capture details of back-end integration, along with the expected data change?
12. Is the test plan documentation simple enough for the test engineers to execute?
13. Does the test plan explain any particular test setup that may be required for successful execution of the plan?

Tools for Test Design & Development

Introduction

Tools for Review & insp

Test planning

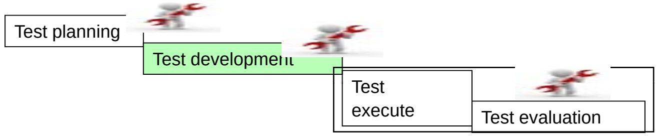
Test develop

Test exe. & ev

Test support

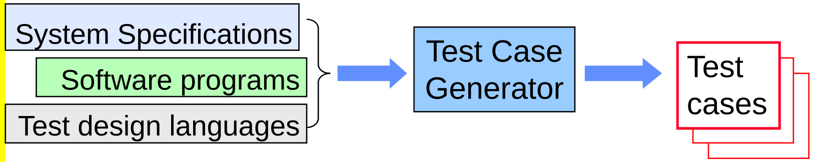
Automate eg

1. **Test generator**
2. **Test driver**
3. **Capture/playback tool**
4. **Coverage analyser**



Test Case Generator

Introduction
Tools for
Review & insp
Test planning
Test develop
Test genera
Test driver
Capture/
playback
Coverage
analyzer
Test exe. & ev
Test support
Automate eg

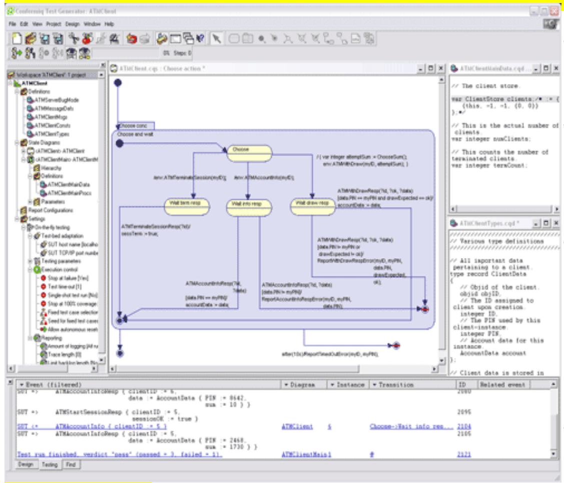

System Specifications
Software programs
Test design languages
Test Case Generator
Test cases

- Automate the generation of test data.
- Most test generators are Specification-based generator: reads requirements for a software product, parse the requirements, and create test cases.

Example Tool:

http://www.atego.com/products/technology-overview/

Example: Conformiq Test Case Generator



- Use dynamic model-based test generation
- Describe the system under test as a graphical test model using extended UML statecharts.
- The test generator analyzes this model and generates test cases, including testing for synchronization issues, concurrent functionalities.
- It also executes the tests against the SUT, and records the progress and outcome of the test run.

Reference:
http://www.verifysoft.com/en_conformiq_testgenerator.html

Types of Test Generator (1)

Introduction
Tools for
Review & insp
Test planning
Test develop
Test genera
Test driver
Capture/
playback
Coverage
analyzer
Test exe. & ev
Test support
Automate eg

(1) Algorithmic test design rule

- for every specified action or function, the generator creates at least 1 *normal* test case
- create 1 or more *abnormal* test cases for every action
- this action-driven testing addresses functions only. Provide no guidance for selecting the input values.

(2) Data-driven

- Identify typical data errors: errors of handling primitive data values such as numbers, lists, or strings; errors in data structures such as arrays, records, or files.
- Use boundary analysis, partition analysis, domain analysis

Types of Test Generator (2)

(3) Events-driven

- An ***event*** is an occurrence in the outside world that causes an action, such as a state transition, to take place.
- generate test cases which exercise every event at least once.

(4) State-driven

- A ***state*** is a set of data values or a logical relationship among data values.
- A ***state transition*** is a change in those values.
- create test cases that exercise transitions in prescribed valid and invalid sequences.

(5) Heuristic Test Generator

- Use information from the tester
- Use a failure-directed approach. Failures that the tester discovered frequently are entered into the tool.
- Use this knowledge of historical failures to generate tests

Types of Test Generator (3)

(6) Statistical Test Generator

⌘ Choose input structures and values to form a statistically random distribution, or a distribution that matches the usage profile of the system under test.

Example: ATM: assume there are 4 user functions: withdraw, deposit, lookup account, and transfer

Function	Usage distribution
Withdraw	60%
Deposit	6%
Lookup	20%
Transfer	14%

Generate test cases according to this distribution (i.e., use more test cases for Withdraw function)

Test Case Generator

Advantages:

☺ Create test cases with less effort than manually.

☺ More thorough and more accurate than a human tester using the same strategy.

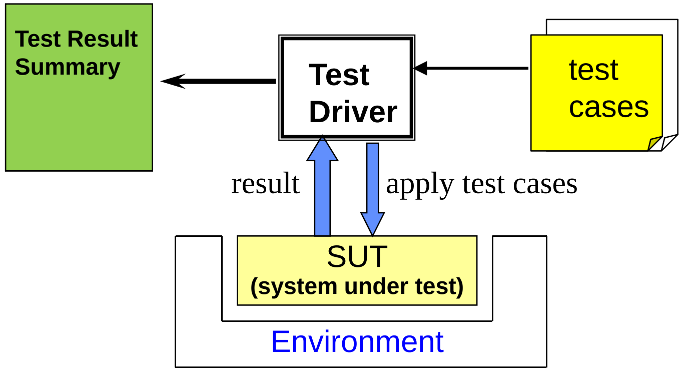
Disadvantages:

☹ May not be able to produce the wide range of transaction data that is needed to properly test an application

☹ Some Tools can only read one specification format!

☹ Human expertise is still needed to prioritize tests, to assess the usefulness of the generated tests, and to think of other useful tests.

Test Driver



⌘ Used to run tests automatically

⌘ Use a scripting language to specify testing.

Example tool: ATTOL UniTest from Cleanscape, http://www.cleanscape.net/docs_lib/data_attol_unitest.pdf

Key Steps of Test Driver

1. Setup: Confirm the hardware/software configuration; loads and initializes prerequisite or corequisite components; sets up access to data structures as required.

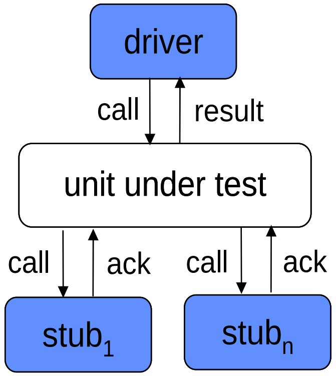
2. Execution: Reinitializes as appropriate for every test; loads and controls inputs for every test; evaluates assertions; captures outputs.

3. Postmortem: reports test failures by exception; compares actual to predicted outcomes; passes execution data to coverage tool; confirms path; passes control to debug package on test failure.

Test Harness (include driver and stub)

A *test harness* is the auxiliary code developed to support testing of units, consisting of:

- Drivers that call the target code, and
- Stubs that represents modules it calls.



Capture/Playback Tool (1)

- A capture/playback tool captures user inputs and stores it into a script suitable to be used at a later time to replay the user inputs.
- It is the **most popular tool**

Test script do 3 things:

- **Input:** trigger some behavior on the application
- **Output:** the results
- **Comparison (output analysis):** evaluate the output against the expected output

Advantages

- Makes complete regression testing possible.
- No need to manually re-run tests when defect fixes and enhancements change the product.

Disadvantage

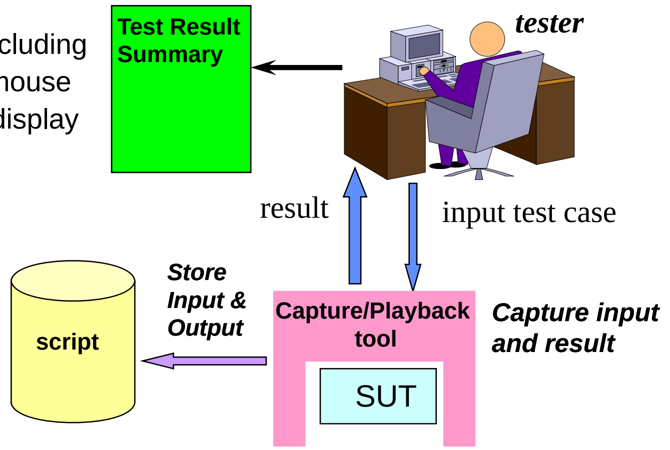
- Stored test cases may be difficult to understand and maintain.

Capture/Playback Tool (2)

First Time Testing

1. Capture phase:

Capture user operations including keystrokes, mouse activity, and display output.

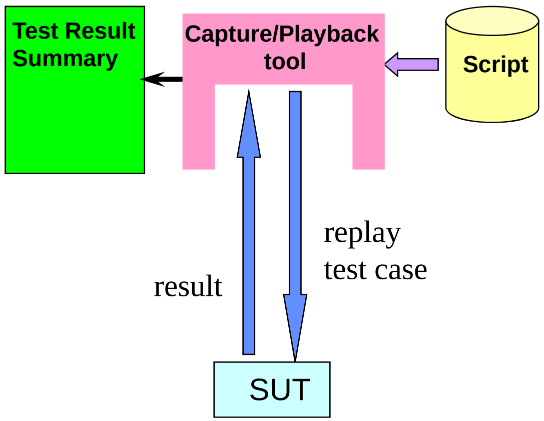


Capture/Playback Tool (3)

2nd, 3rd, ... Time Testing

2. Playback phase:

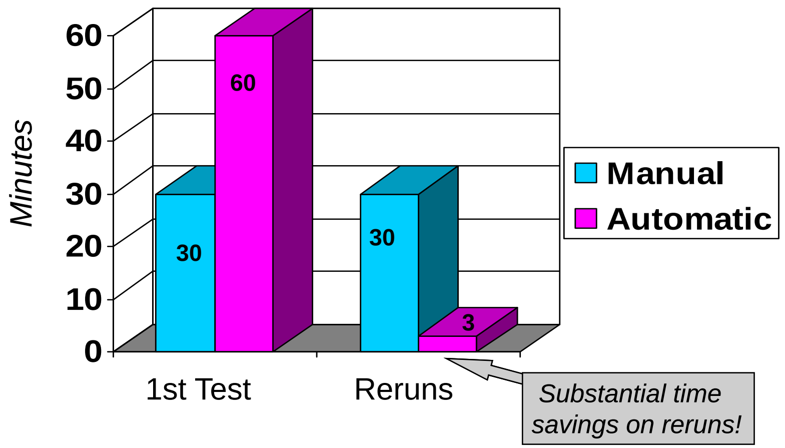
Play back the previously captured tests and validate the results by comparing them to the previously saved baseline.



Manual vs. Automated Testing

Introduction
Tools for Review & insp
Test planning
Test develop
Test genera
Test driver
Capture/ playback
Coverage analyzer
Test exe. & ev
Test support
Automate eg

Test Times



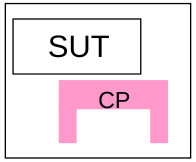
Substantial time savings on reruns!

Types of Capture/Playback Tool (1)

Introduction
Tools for Review & insp
Test planning
Test develop
Test genera
Test driver
Capture/ playback
Coverage analyzer
Test exe. & ev
Test support
Automate eg

1. native (or intrusive)

- ⌘ Both the c/p tool and the SUT reside in the **same computer system.**
- ⌘ The tool may distort the operating performance to some extent. But, this distortion may be irrelevant for most applications
- ⌘ 2 types:
  - ➔ software intrusive (introduces distortion at software level within the SUT)
  - ➔ hardware intrusive (introduces distortion at hardware level only).

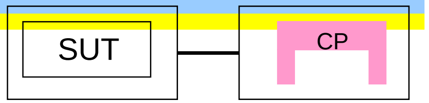


Same computer

Types of Capture/Playback Tool (2)

Introduction
Tools for Review & insp
Test planning
Test develop
Test genera
Test driver
Capture/ playback
Coverage analyzer
Test exe. & ev
Test support
Automate eg

2. non-intrusive



Computer A *Computer B*

- ⌘ Require additional hardware
- ⌘ Host system has a special hardware connection to the c/p tool; the c/p tool performs its required functions transparently to the host software.
- ⌘ Needed for product which is an integrated hardware and software system where extra hardware or software cannot be tolerated, e.g., real-time embedded systems.

Most common c/p tool is native/software intrusive.

Example Tools

- Load2Test http://www.enteros.com/
- http://www.sharewareconnection.com/author.php?name=verisium,+inc.

Coverage Analyser

Introduction
Tools for Review & insp
Test planning
Test develop
Test genera
Test driver
Capture/ playback
Coverage analyzer
Test exe. & ev
Test support
Automate eg

Code coverage measures the elements of the code that have been exercised during testing.

Typical coverage analysers provide these functions:

- ▪ provide quantitative measure of the quality of tests, i.e., they find out if the software is being thoroughly tested.
- ▪ provide statistics on the most common coverage measures: statement, branch, path, dataflow and function coverage;
- ▪ highlight which elements of the system have been **executed (covered)** by the current tests and point out elements not yet exercised.
- generate graphical reports
- track change of coverage as more tests are run.

Coverage analysers are usually used at **unit test**.

Example

Introduction

Tools for
Review & insp

Test planning

Test develop

Test genera
Test driver
Capture/
playback
Coverage analyzer

Test exe. & ev

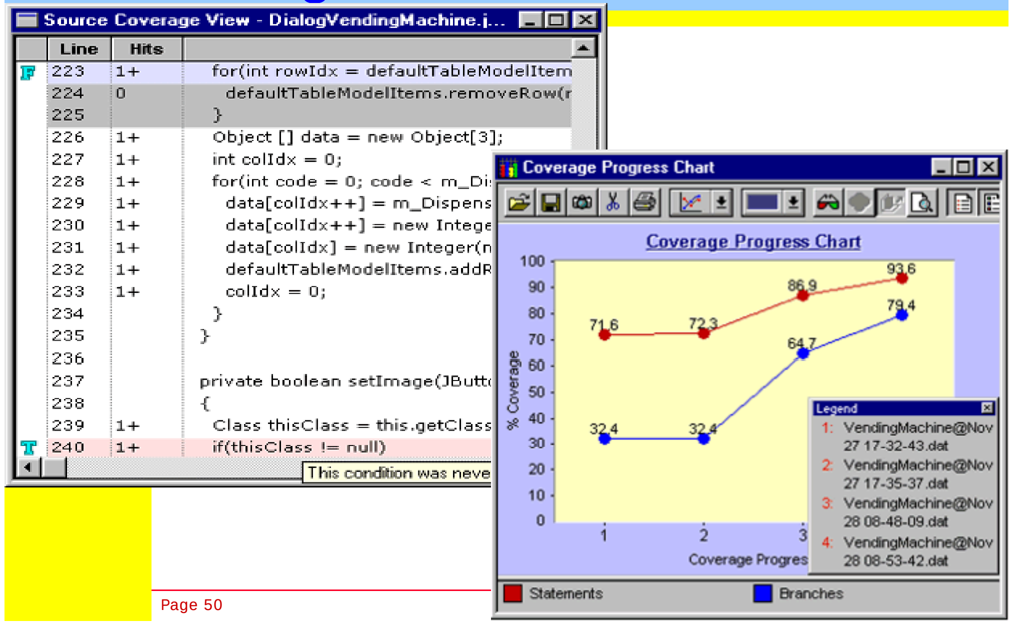
Test support

Automate eg

```
while (a<33) {
    ..
    if (whatever) break;
    ..
}
Post_loop_cleanup();
```

A code coverage tool can tell us whether the ***while*** test has ever been false, or whether the ***if*** statement has been executed during testing.

Coverage Tool: JCover for Java



Source Coverage View - DialogVendingMachine.j...

```
      Line  Hits
F     223   1+      for(int rowIdx = defaultTableModelItem
      224   0         defaultTableModelItems.removeRow(r
      225             }
      226   1+      Object [] data = new Object[3];
      227   1+      int colIdx = 0;
      228   1+      for(int code = 0; code < m_Di
      229   1+        data[colIdx++] = m_Dispens
      230   1+        data[colIdx++] = new Intege
      231   1+        data[colIdx] = new Integer(n
      232   1+        defaultTableModelItems.addR
      233   1+        colIdx = 0;
      234           }
      235         }
      236
      237         private boolean setImage(JButt
      238         {
      239   1+      Class thisClass = this.getClass
T     240   1+      if(thisClass != null)
```

This condition was neve

Example: Many Coverage Types

Introduction

Tools for
Review & insp

Test planning

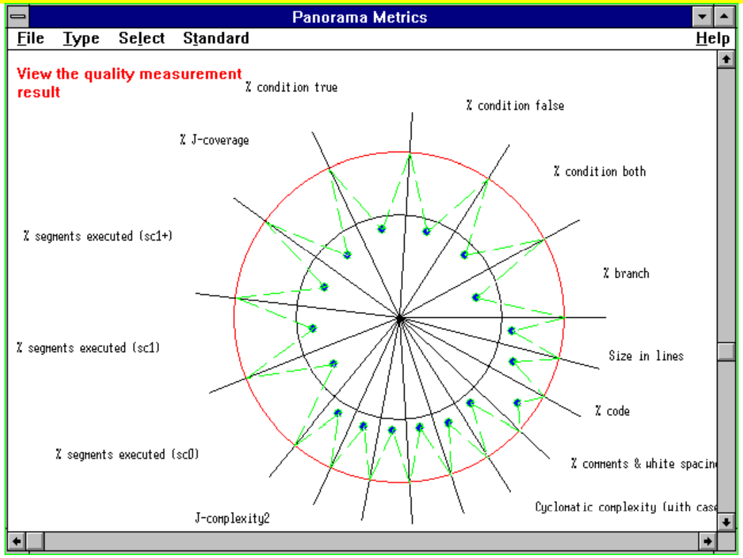
Test develop

Test genera
Test driver
Capture/
playback
Coverage analyzer

Test exe. & ev

Test support

Automate eg



Panorama Metrics
File Type Select Standard Help

View the quality measurement result

% condition true
% condition false
% J-coverage
% condition both
% segments executed (sc1+)
% branch
% segments executed (sc1)
Size in lines
% code
% segments executed (sc0)
% comments & white spacin
Cyclomatic complexity (with case
J-complexity2

Example Coverage

Introduction

Tools for
Review & insp

Test planning

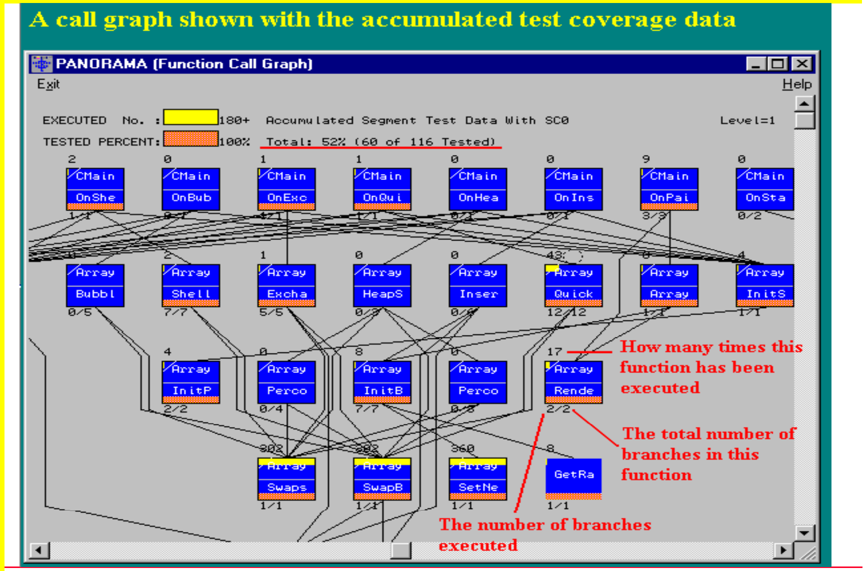
Test develop

Test genera
Test driver
Capture/
playback
Coverage analyzer

Test exe. & ev

Test support

Automate eg



A call graph shown with the accumulated test coverage data

PANORAMA (Function Call Graph)

```
EXECUTED  No. :      180+  Accumulated Segment Test Data With SC0        Level=1
TESTED PERCENT:      100%  Total: 52% (60 of 116 Tested)
```

How many times this function has been executed

The total number of branches in this function

The number of branches executed

Coverage Analyser

Introduction

Tools for
Review & insp

Test planning

Test develop

Test genera
Test driver
Capture/
playback
**Coverage
analyzer**

Test exe. & ev

Test support

Automate eg

Advantage: metrics provide an assessment of the extent and/or effectiveness of testing

Disadvantage: 100% coverage of logic does not correlate with a defect-free system.

☹ Testing performance is not good with an instrumented code, as it increases runtime and memory requirements.

☹ Some study indicates the use of code coverage tool may require 2% of the total development effort.

Example Tools:

- PurifyPlus from IBM/Rational http://www-306.ibm.com/software/awdtools/purifyplus/
- Junit
- HTMLunit

Limitation of Coverage Analyser

Introduction

Tools for
Review & insp

Test planning

Test develop

Test genera
Test driver
Capture/
playback
**Coverage
analyzer**

Test exe. & ev

Test support

Automate eg

⌘ **Different compilers**

Test tool's compiler and the working compiler are often built by different organizations. For safety, tests run under a test tool should be repeated with the real compiler to confirm the results

⌘ **May not test all conditions**

Compilers use lazy evaluation in which predicate evaluation stops when the truth value of the predicate is determined. It is possible to attempt to cover all predicate conditions (e.g., (A>3) AND (B<C)) but not actually do so!

⌘ **Artificial test environment**

Proper behavior of interrupts, cache memories, other OS features do not work as they do in reality or work differently with the instrumented version of the code.

Tools for Test Execution & Evaluation

Introduction

Tools for
Review & insp

Test planning

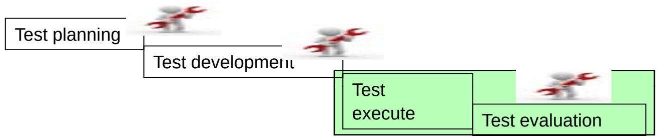
Test develop

Test exe. & ev

Test support

Automate eg

1. Capture/playback tool
2. Coverage analyser
3. **Memory testing tool (leak detector)**
4. **Performance tool**



Memory Leak

lead to running out of memory

Introduction

Tools for
Review & insp

Test planning

Test develop

Test exe. & ev

Memory test
Performance
test tool

Test support

Automate eg

The root cause for a memory leak is **failure to reclaim virtual memory** that is dynamically requested by the application.

Typically, the memory leak is found in a small function that is written correctly for its *normal path*, but incorrectly for its *abnormal path*.

Example: If virtual memory allocation is done at the start of the function, and there is a program interrupt, the error handler for the interrupt may not have been programmed to release the virtual memory.

For applications that run 24/7, even the smallest leak ultimately will eventually cause a system crash.

Memory Problems (1)

Introduction

Tools for
Review & insp

Test planning

Test develop

Test exe. & ev

Memory test
Performance
test tool

Test support

Automate eg

1 **memory blowout**: OS allocates pages of memory to the program and never release them. Other programs running on this machine will starve for memory and then crash.

2 **memory overuse (Hogging):** memory is allocated by a program and never freed.

3 **bottleneck:** when OS spends more time paging memory than running the program.

Memory Problems (2)

Introduction

Tools for
Review & insp

Test planning

Test develop

Test exe. & ev

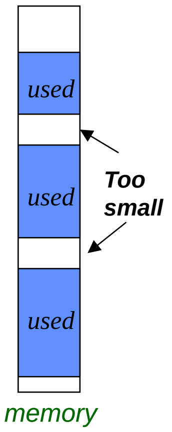
Memory test
Performance
test tool

Test support

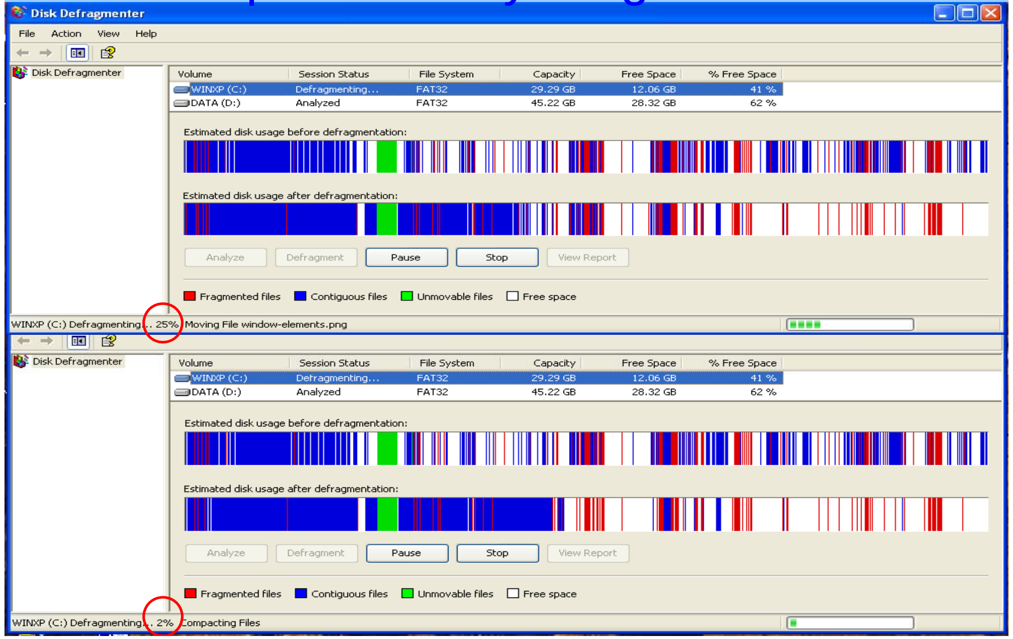
Automate eg

4 **memory fragmentation:** caused by overuse of memory.

- Fragmentation is a process that occurs when various-sized blocks of data storage are allocated and freed in the course of a program's operation.
- It results in the accumulation of small regions of free storage that are too small to be useful for allocation, even though in sum there may be more than sufficient free space.
- It slows down memory allocation.



Example: Memory Fragmentation



Volume	Session Status	File System	Capacity	Free Space	% Free Space
WINXP (C:)	Defragmenting...	FAT32	29.29 GB	12.06 GB	41 %
DATA (D:)	Analyzed	FAT32	45.22 GB	28.32 GB	62 %

Estimated disk usage before defragmentation:

Estimated disk usage after defragmentation:

🟥 Fragmented files 🟦 Contiguous files 🟩 Unmovable files □ Free space

WINXP (C:) Defragmenting... 25% Moving File window-elements.png

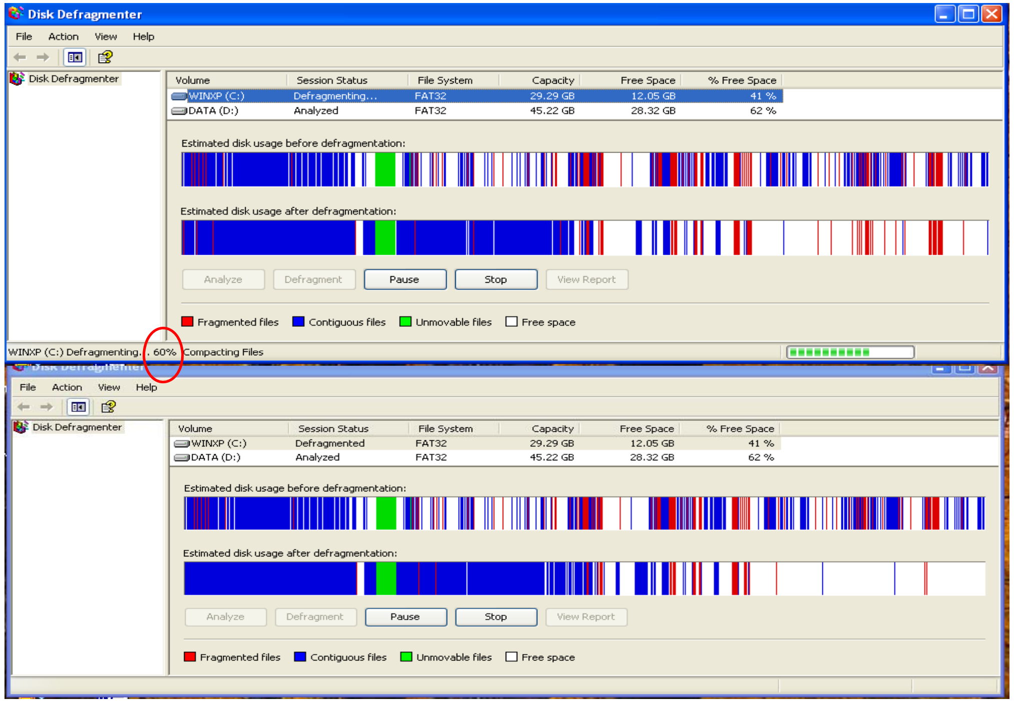
Volume	Session Status	File System	Capacity	Free Space	% Free Space
WINXP (C:)	Defragmenting...	FAT32	29.29 GB	12.06 GB	41 %
DATA (D:)	Analyzed	FAT32	45.22 GB	28.32 GB	62 %

Estimated disk usage before defragmentation:

Estimated disk usage after defragmentation:

🟥 Fragmented files 🟦 Contiguous files 🟩 Unmovable files □ Free space

WINXP (C:) Defragmenting... 2% Compacting Files



Volume	Session Status	File System	Capacity	Free Space	% Free Space
WINXP (C:)	Defragmenting...	FAT32	29.29 GB	12.05 GB	41 %
DATA (D:)	Analyzed	FAT32	45.22 GB	28.32 GB	62 %

Estimated disk usage before defragmentation:

Estimated disk usage after defragmentation:

🟥 Fragmented files 🟦 Contiguous files 🟩 Unmovable files □ Free space

WINXP (C:) Defragmenting... 60% Compacting Files

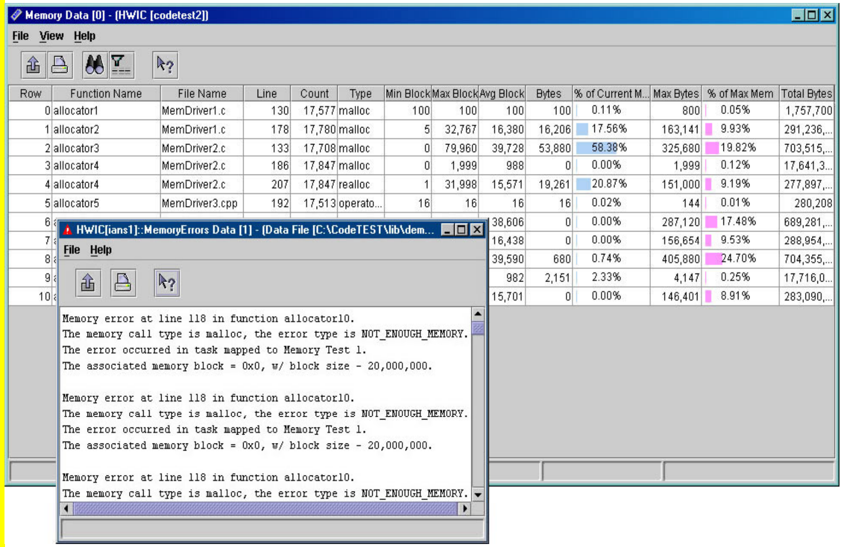
Volume	Session Status	File System	Capacity	Free Space	% Free Space
WINXP (C:)	Defragmented	FAT32	29.29 GB	12.05 GB	41 %
DATA (D:)	Analyzed	FAT32	45.22 GB	28.32 GB	62 %

Estimated disk usage before defragmentation:

Estimated disk usage after defragmentation:

🟥 Fragmented files 🟦 Contiguous files 🟩 Unmovable files □ Free space

Example Memory Usage Report



Row	Function Name	File Name	Line	Count	Type	Min Block	Max Block	Avg Block	Bytes	% of Current M...	Max Bytes	% of Max Mem	Total Bytes
0	allocator1	MemDriver1.c	130	17,577	malloc	100	100	100	100	0.11%	800	0.05%	1,757,700
1	allocator2	MemDriver1.c	178	17,780	malloc	5	32,767	16,380	16,206	17.56%	163,141	9.93%	291,236,...
2	allocator3	MemDriver2.c	133	17,708	malloc	0	79,960	39,728	53,880	58.38%	325,680	19.82%	703,515,...
3	allocator4	MemDriver2.c	186	17,847	malloc	0	1,999	988	0	0.00%	1,999	0.12%	17,641,3...
4	allocator4	MemDriver2.c	207	17,847	realloc	1	31,998	15,571	19,261	20.87%	151,000	9.19%	277,897,...
5	allocator5	MemDriver3.cpp	192	17,513	operato...	16	16	16	16	0.02%	144	0.01%	280,208
6	a							38,606	0	0.00%	287,120	17.48%	689,281,...
7	a							16,438	0	0.00%	156,654	9.53%	288,954,...
8	a							39,590	680	0.74%	405,880	24.70%	704,355,...
9	a							982	2,151	2.33%	4,147	0.25%	17,716,0...
10	a							15,701	0	0.00%	146,401	8.91%	283,090,...

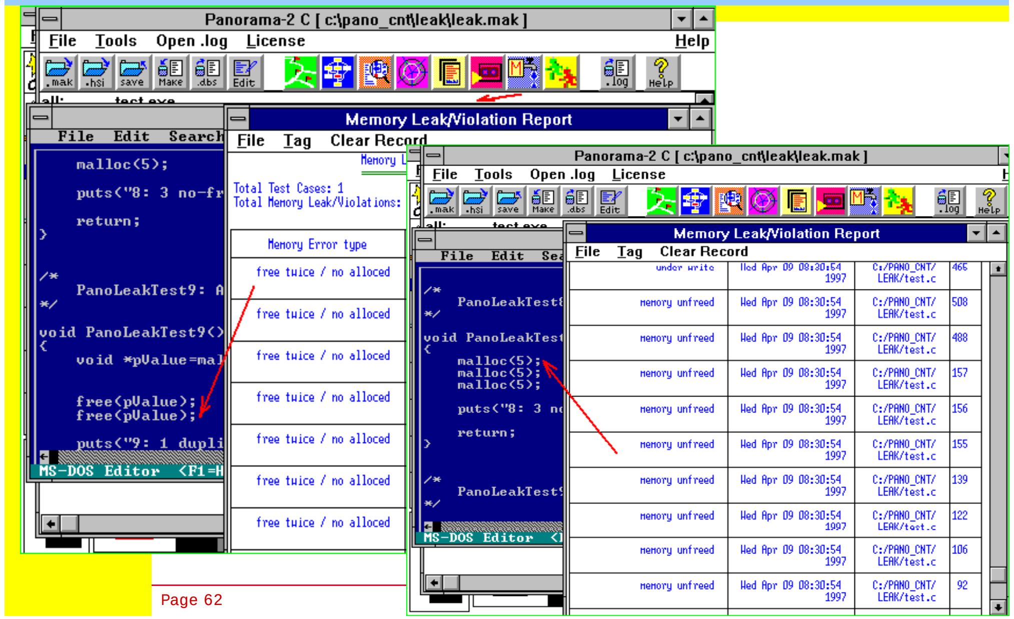
HWIC[ians1]::MemoryErrors Data [1] - (Data File [C:\CodeTEST\lib\dem...
File Help

```
Memory error at line 118 in function allocator10.
The memory call type is malloc, the error type is NOT_ENOUGH_MEMORY.
The error occurred in task mapped to Memory Test 1.
The associated memory block = 0x0, w/ block size - 20,000,000.

Memory error at line 118 in function allocator10.
The memory call type is malloc, the error type is NOT_ENOUGH_MEMORY.
The error occurred in task mapped to Memory Test 1.
The associated memory block = 0x0, w/ block size - 20,000,000.

Memory error at line 118 in function allocator10.
The memory call type is malloc, the error type is NOT_ENOUGH_MEMORY.
```

Example Memory Leak Report


Panorama-2 C [c:\pano_cnt\leak\leak.mak]
File Tools Open .log License Help
.mak .hsi save Make .dbs Edit .log Help
all: test.exe
File Edit Search
MS-DOS Editor <F1=H
Memory Leak/Violation Report
File Tag Clear Record
Memory L

```
    malloc(5);

    puts("8: 3 no-fr

    return;
}


/*
    PanoLeakTest9: A
*/

void PanoLeakTest9()
{
    void *pValue=mal

    free(pValue);
    free(pValue);

    puts("9: 1 dupli
```

Total Test Cases: 1
Total Memory Leak/Violations:

Memory Error type
free twice / no alloced
free twice / no alloced
free twice / no alloced
free twice / no alloced
free twice / no alloced
free twice / no alloced
free twice / no alloced

Panorama-2 C [c:\pano_cnt\leak\leak.mak]
File Tools Open .log License
.mak .hsi save Make .dbs Edit .log Help
all: test.exe
File Edit Sea
MS-DOS Editor <
Memory Leak/Violation Report
File Tag Clear Record

```
/*
    PanoLeakTest8
*/

void PanoLeakTest
{
    malloc(5);
    malloc(5);
    malloc(5);

    puts("8: 3 no

    return;
}


/*
    PanoLeakTest9
*/
```

under write	Wed Apr 09 08:30:54 1997	C:/PANO_CNT/ LEAK/test.c	465
memory unfreed	Wed Apr 09 08:30:54 1997	C:/PANO_CNT/ LEAK/test.c	508
memory unfreed	Wed Apr 09 08:30:54 1997	C:/PANO_CNT/ LEAK/test.c	488
memory unfreed	Wed Apr 09 08:30:54 1997	C:/PANO_CNT/ LEAK/test.c	157
memory unfreed	Wed Apr 09 08:30:54 1997	C:/PANO_CNT/ LEAK/test.c	156
memory unfreed	Wed Apr 09 08:30:54 1997	C:/PANO_CNT/ LEAK/test.c	155
memory unfreed	Wed Apr 09 08:30:54 1997	C:/PANO_CNT/ LEAK/test.c	139
memory unfreed	Wed Apr 09 08:30:54 1997	C:/PANO_CNT/ LEAK/test.c	122
memory unfreed	Wed Apr 09 08:30:54 1997	C:/PANO_CNT/ LEAK/test.c	106
memory unfreed	Wed Apr 09 08:30:54 1997	C:/PANO_CNT/ LEAK/test.c	92

Memory Testing Tool

Advantage

- Helps to identify memory problems.

Disadvantage

- May prevent reproduction of defects under certain situation.

Example Tools:

DevPartner

http://www.microfocus.com/products/micro-focus-developer/devpartner/index.aspx

Where is the memory leak?

```
#include <stdio.h&qt;
#include <malloc.h&qt;
static char   *p, buf[1024];
main()
{ int cnt;
 FILE  *fp;
 char  * oldp = malloc(2048);
fp = fopen(“/etc/termcap”,”r”);
 while (fgets(buf,1024,fp) != NULL)
 {  cnt = strlen(buf);
    p = (char *)malloc(cnt);
    memcpy(p,buf,cnt);
    write(1,p,cnt);
 }
 free(oldp);
}
```

Performance Testing

Introduction

Tools for
Review & insp

Test planning

Test develop

Test exe. & ev

Memory test
Performance test tool

Test support

Automate eg

Key **goal** of performance testing: Measuring the software product under a **real or simulated load**.

- Performance testing cannot be performed earlier in the life cycle because a fully or nearly developed software product is needed.

- For most applications, most of the execution time is spent in a relatively small amount of the code.
- Performance tool provides specific data about how and when execution time was spent, and how much time is attributable to which routines (code).
- Help to identify routines/procedures/functions where the most time is spent.

Change of Performance with User Number

Introduction

Tools for
Review & insp

Test planning

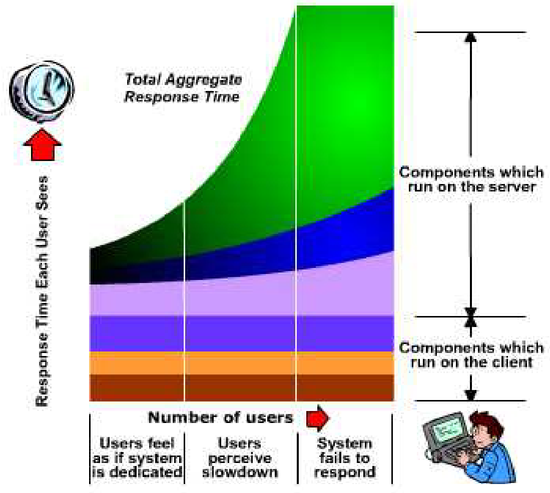
Test develop

Test exe. & ev

Memory test
Performance test tool

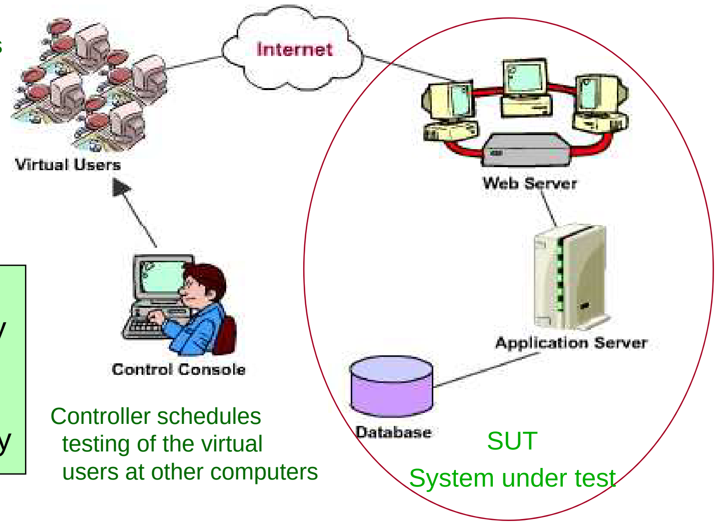
Test support

Automate eg



Performance Tool (1)

Each computer runs test scripts (some may be the same script)



Objectives:
simulate many users using the system simultaneously

Controller schedules testing of the virtual users at other computers

Controller

Introduction

Tools for
Review & insp

Test planning

Test develop

Test exe. & ev

Memory test
Performance test tool

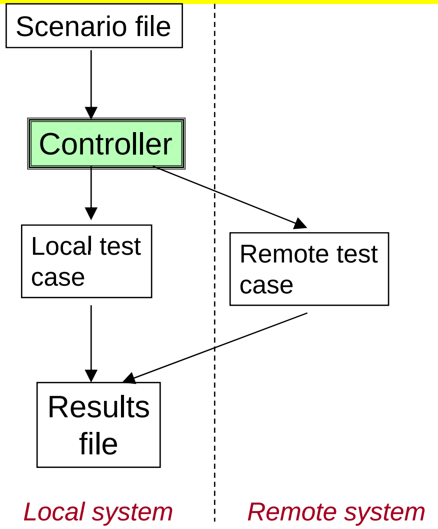
Test support

Automate eg

A controller allocates different parts of the test to separate computers (local and remote)

The scenarios tell the test case controller which test cases to process and how to process them.

At the start of a test run, the test controller reads the scenario file. It then executes the test cases described in the scenario file, and the results are posted into a result file.



Scenario file

Controller

Local test case

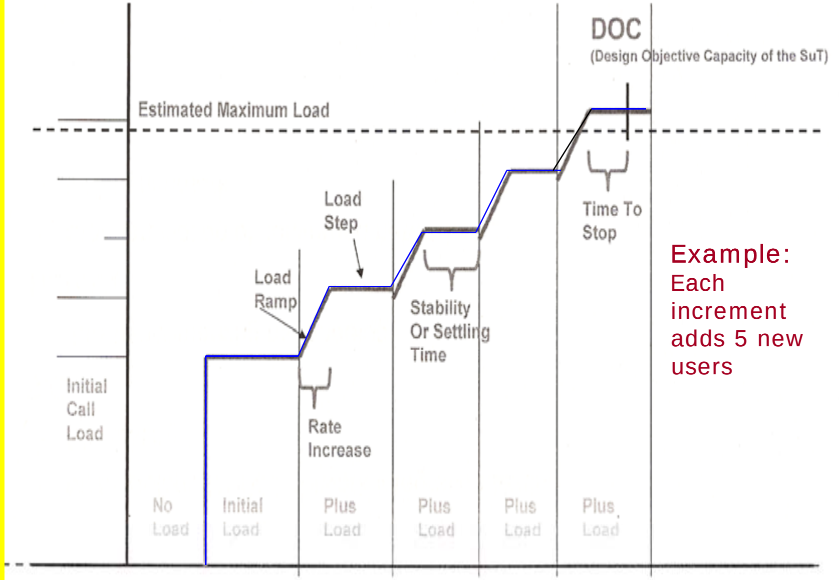
Remote test case

Results file

Local system

Remote system

Incrementally Increase the Load



Example: Each increment adds 5 new users

Performance Tool (2)

Introduction

Tools for
Review & insp

Test planning

Test develop

Test exe. & ev

Memory test
Performance test tool

Test support

Automate eg

- ⌘ Important for internet applications
- ⌘ Apply heavy users loads to database servers and application servers
- ⌘ Emulate network traffic
- ⌘ Record network traffic, server requests and responses, and user think times
- ⌘ Automate the scaling of workloads from one user to tens of thousands of users.

Failure of load test

Example

Introduction

Tools for
Review & insp

Test planning

Test develop

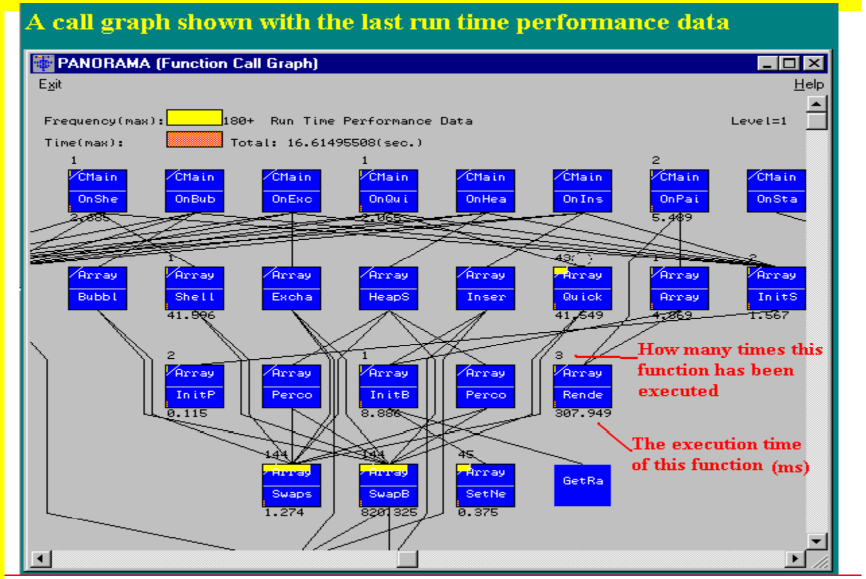
Test exe. & ev

Memory test
Performance test tool

Test support

Automate eg

A call graph shown with the last run time performance data



```
Frequency(max):      180+   Run Time Performance Data                Level=1
Time(max):                  Total: 16.61495508(sec.)
```

How many times this function has been executed

The execution time of this function (ms)

Performance Tool (3)

Introduction

Tools for
Review & insp

Test planning

Test develop

Test exe. & ev

Memory test
Performance test tool

Test support

Automate eg

Advantages

- ⌘ locates where the code spends most time
- ⌘ generates loads which cannot be done manually

Disadvantages

- ⌘ some programming with the test scripting language may be needed.
- ⌘ requires extensive training on the use of the tool.

Example Tools:

IBM Rational Performance Test http://www-01.ibm.com/software/awdtools/tester/performance/

Tools for Test Support

Introduction

Tools for

Review & insp

Test planning

Test develop

Test exe. & ev

Test support

Automate eg

1. **Problem management tool** (or **defect tracker, incident control system)**
2. Configuration management tool
3. **Test case management tool**

Problem Management Tool

Introduction

Tools for

Review & insp

Test planning

Test develop

Test exe. & ev

Test support

Problem managmt

Test case managmt

Automate eg

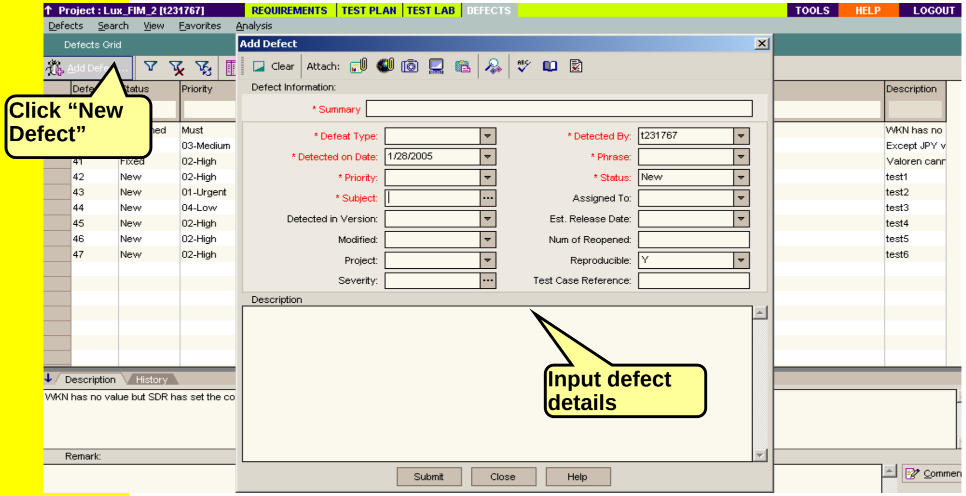
⌘ Used to record, track, and assist with the management of defects and enhancements throughout the life cycle of software products

⌘ **Typical features:**

- ➔ submit and update defect reports
- ➔ generate pre-defined or user-defined management reports, chart defect trends
- ➔ route defect reports to the responsible person
- ➔ selectively notify users automatically of changes in defect status
- ➔ provide access to all data via user-defined queries

Example Problem Management Tool: Test Director

Create a new defect and input details



Click "New Defect"

Add Defect

Defect Information:

* Summary			
* Defeat Type:		* Detected By:	t231767
* Detected on Date:	1/28/2005	* Phrase:	
* Priority:		* Status:	New
* Subject:		Assigned To:	
Detected in Version:		Est. Release Date:	
Modified:		Num of Reopened:	
Project:		Reproducible:	Y
Severity:		Test Case Reference:	

Description

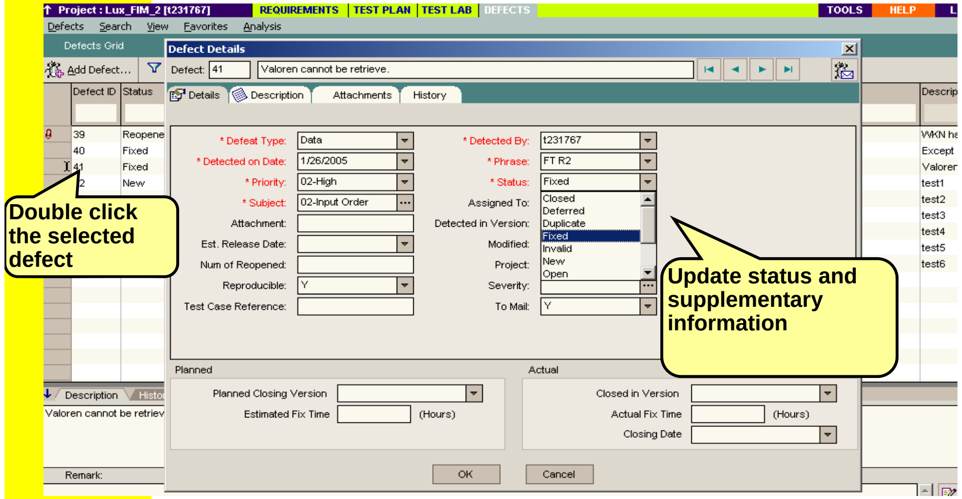
Input defect details

Defect ID	Status	Priority	Description
	…ned	Must	WKN has no
		03-Medium	Except JPY v
41	Fixed	02-High	Valoren cann
42	New	02-High	test1
43	New	01-Urgent	test2
44	New	04-Low	test3
45	New	02-High	test4
46	New	02-High	test5
47	New	02-High	test6

WKN has no value but SDR has set the co

Amend Defect

Update defect supplementary information



Defect Details

Defect: 41 Valoren cannot be retrieve.

* Defeat Type:	Data	* Detected By:	t231767
* Detected on Date:	1/26/2005	* Phrase:	FT R2
* Priority:	02-High	* Status:	Fixed
* Subject:	02-Input Order	Assigned To:	
Attachment:		Detected in Version:	
Est. Release Date:		Modified:	
Num of Reopened:		Project:	
Reproducible:	Y	Severity:	
Test Case Reference:		To Mail:	Y

Closed
Deferred
Duplicate
Fixed
Invalid
New
Open

Planned

Planned Closing Version

Estimated Fix Time (Hours)

Actual

Closed in Version

Actual Fix Time (Hours)

Closing Date

Defect ID	Status	Descrip
39	Reopene	WKN ha
40	Fixed	Except
41	Fixed	Valoren
42	New	test1
		test2
		test3
		test4
		test5
		test6

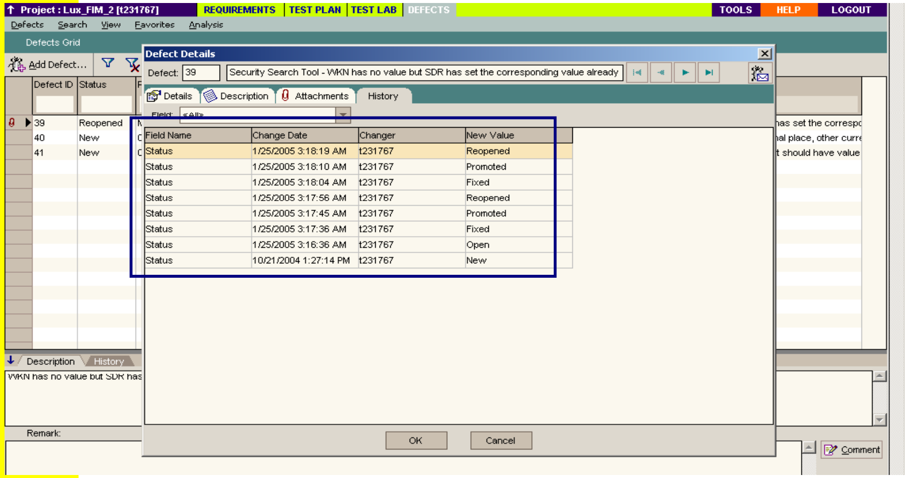
Valoren cannot be retriev

Double click the selected defect

Update status and supplementary information

Status History

Status history can be traced easily



Defect Details

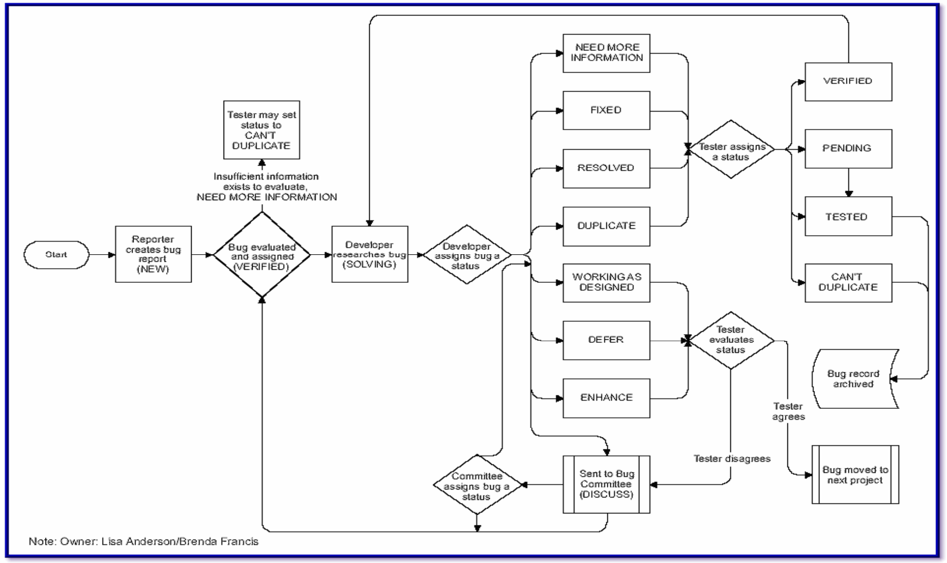
Defect: 39 Security Search Tool - WKN has no value but SDR has set the corresponding value already

Field Name	Change Date	Changer	New Value
Status	1/25/2005 3:18:19 AM	t231767	Reopened
Status	1/25/2005 3:18:10 AM	t231767	Promoted
Status	1/25/2005 3:18:04 AM	t231767	Fixed
Status	1/25/2005 3:17:56 AM	t231767	Reopened
Status	1/25/2005 3:17:45 AM	t231767	Promoted
Status	1/25/2005 3:17:36 AM	t231767	Fixed
Status	1/25/2005 3:16:36 AM	t231767	Open
Status	10/21/2004 1:27:14 PM	t231767	New

Defect ID	Status
39	Reopened
40	New
41	New

WKN has no value but SDR has

A Defect Life Cycle

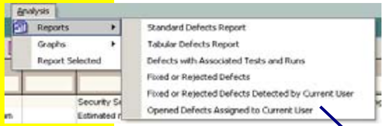


Start
Reporter creates bug report (NEW)
Bug evaluated and assigned (VERIFIED)
Insufficient information exists to evaluate, NEED MORE INFORMATION
Tester may set status to CAN'T DUPLICATE
Developer researches bug (SOLVING)
Developer assigns bug a status
NEED MORE INFORMATION
FIXED
RESOLVED
DUPLICATE
WORKING AS DESIGNED
DEFER
ENHANCE
Tester assigns a status
VERIFIED
PENDING
TESTED
CAN'T DUPLICATE
Tester evaluates status
Bug record archived
Tester agrees
Tester disagrees
Bug moved to next project
Sent to Bug Committee (DISCUSS)
Committee assigns bug a status

Note: Owner: Lisa Anderson/Brenda Francis

Reference: Crosstalk, Sept. 2003

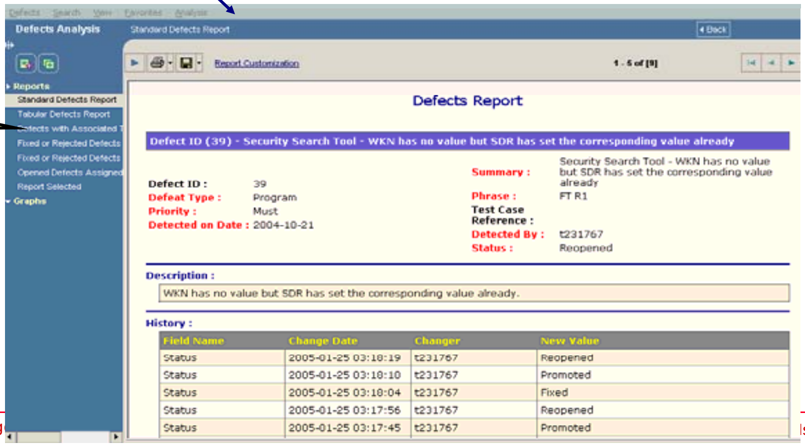
Report



Analysis
Reports
Graphs
Report Selected
Standard Defects Report
Tabular Defects Report
Defects with Associated Tests and Runs
Fixed or Rejected Defects
Fixed or Rejected Defects Detected by Current User
Opened Defects Assigned to Current User

Different reports can be generated

Select report



Defects Report

Defect ID (39) - Security Search Tool - WKN has no value but SDR has set the corresponding value already

Defect ID : 39
Defeat Type : Program
Priority : Must
Detected on Date : 2004-10-21

Summary : Security Search Tool - WKN has no value but SDR has set the corresponding value already
Phrase : FT R1
Test Case Reference :
Detected By : t231767
Status : Reopened

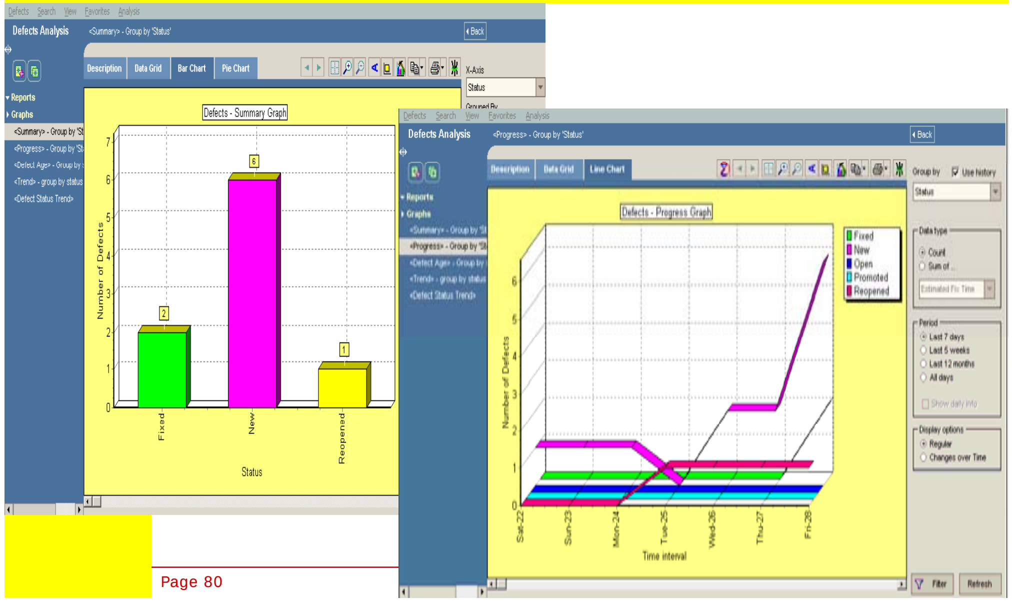
Description :

WKN has no value but SDR has set the corresponding value already.

History :

Field Name	Change Date	Changer	New Value
Status	2005-01-25 03:18:19	t231767	Reopened
Status	2005-01-25 03:18:10	t231767	Promoted
Status	2005-01-25 03:18:04	t231767	Fixed
Status	2005-01-25 03:17:56	t231767	Reopened
Status	2005-01-25 03:17:45	t231767	Promoted

Graph Reports: Bar and Line Charts



Example functions (from *QuickBugs*)

Introduction
Tools for
Review & insp
Test planning
Test develop
Test exe. & ev
Test support
Problem managmt
Test case managmt
Automate eg

- ☺ Record bugs in a shared XML repository accessible to developers, testers and managers.
- ☺ Support concurrent access to bug reports by multiple users.
- ☺ Assign responsibilities, access and privileges to users for each project.
- ☺ Customize the data collection and workflow for each project.
- ☺ Estimate, assign and track the work required to resolve and test bugs.
- ☺ Classify, sort, view, print and generate reports for bugs and new feature requests according to priority, type and other properties.
- ☺ Estimate personnel workloads and time required for project.
- ☺ Determine release dates based on bug detection and resolution statistics.
- ☺ Automatically maintain the history of each bug.
- ☺ Generate release notes, progress reports and statistics.
- ☺ Evaluate the efficiency of detection methods for locating bugs.

Problem Management Tool

Introduction
Tools for
Review & insp
Test planning
Test develop
Test exe. & ev
Test support
Problem managmt
Test case managmt
Automate eg

Advantages:

- ☺ Help to ensure all defects are corrected before a system or modification goes into production.
- ☺ Most tools cost about the same as a typical spreadsheet or word processing package
- ☺ Repository of defects can help to identify defect trends; aid in process improvement

Example Tools:

TestDirector from HP

http://www-heva.mercuryinteractive.com/products/testdirector/

Hints for Using Problem Management Tool

Introduction
Tools for
Review & insp
Test planning
Test develop
Test exe. & ev
Test support
Problem managmt
Test case managmt
Automate eg

- The correct input of each defect is very important. The programmer must be able to reproduce the defect that the tester or user has reported.
- The staff should provide effective defect descriptions so that defects can be debugged successfully.
- Use the Tool often to check for new issue reports; otherwise, your customers won’t use it. Reply quickly to customers so that they continue to use the system instead of making a phone call.
- Use the Tool to get information, not to keep tabs on your staff’ behavior; otherwise, they won’t input the true data.

Automating Problem Reporting

Introduction
Tools for
Review & insp
Test planning
Test develop
Test exe. & ev
Test support
Problem managmt
Test case managmt
Automate eg

Microsoft is doing automated problem reporting.

Statistics from Microsoft:

- MS Office has 35 MLOC
- 92% customer experienced 0 or 1 crash/month
- 0.75% customer experienced 10+ crash/month

Many problems!!

Automating Problem Reporting

Introduction

Tools for

Review & insp

Test planning

Test develop

Test exe. & ev

Test support

Problem managmt

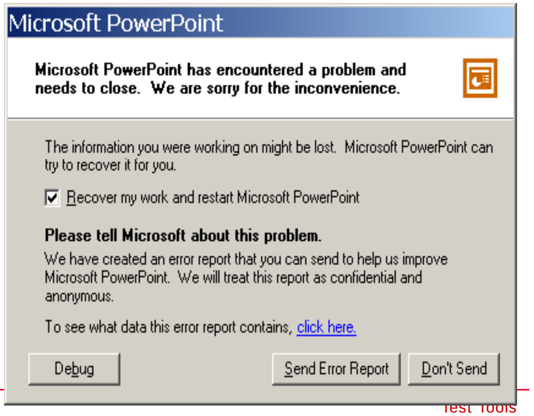
Test case managmt

Automate eg

Microsoft first rolled out automated problem reporting late in the development cycle for its Office XP (in 2003).

Reference: Steven Sinofsky, Senior VP, Microsoft, Dec. 8, 2004, HKICC

- Whenever Office crashes, MS ask whether you want to send the report to them!
- They then applies statistical reporting methods to see the scope and severity of problems.
- By Dec. 2004, MS received >2 Billion this kind of report (Many defects!!)



Microsoft PowerPoint

Microsoft PowerPoint has encountered a problem and needs to close. We are sorry for the inconvenience.

The information you were working on might be lost. Microsoft PowerPoint can try to recover it for you.

☑ Recover my work and restart Microsoft PowerPoint

Please tell Microsoft about this problem.

We have created an error report that you can send to help us improve Microsoft PowerPoint. We will treat this report as confidential and anonymous.

To see what data this error report contains, click here.

Test Case Management Tool (1)

Introduction

Tools for

Review & insp

Test planning

Test develop

Test exe. & ev

Test support

Problem managmt

Test case managmt

Automate eg

- Organize tests for ease of use and maintenance;
- Provide seamless integration with capture/playback and coverage analysis tools,
- Provide automated and extensive test reporting and documentation, e.g. test plans, specifications, results.

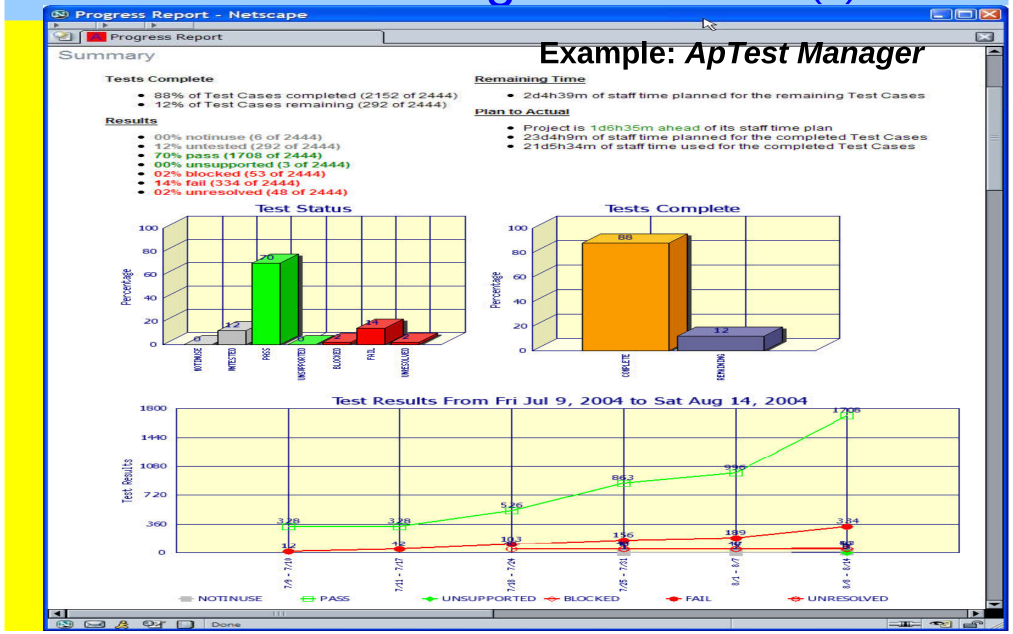
Advantage

⌘ supports the project management aspects of testing, for example, the scheduling of tests, the logging of results and the management of incidents raised during testing.

Example Tools:

- HP Quality Center (TestDirector)
- Seapine TestTrack
- QMetry
- TestRail
- XStudio

Test Case Management Tool (2)



Example: *ApTest Manager*

Progress Report - Netscape

Progress Report

Summary

Tests Complete

- 88% of Test Cases completed (2152 of 2444)
- 12% of Test Cases remaining (292 of 2444)

Results

- 00% notinuse (6 of 2444)
- 12% untested (292 of 2444)
- 70% pass (1708 of 2444)
- 00% unsupported (3 of 2444)
- 02% blocked (53 of 2444)
- 14% fail (334 of 2444)
- 02% unresolved (48 of 2444)

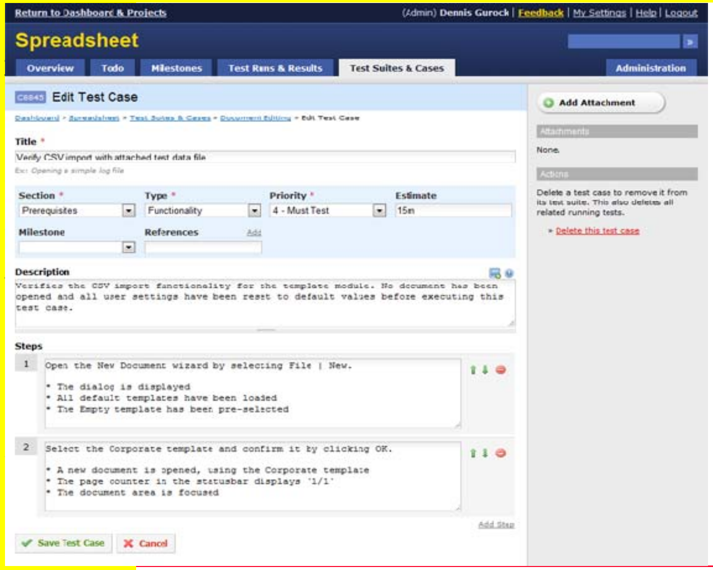
Remaining Time

- 2d4h39m of staff time planned for the remaining Test Cases

Plan to Actual

- Project is 1d6h35m ahead of its staff time plan
- 23d4h9m of staff time planned for the completed Test Cases
- 21d5h34m of staff time used for the completed Test Cases

Test Case Management Tool (3)



C6845 Edit Test Case

Dashboard » Spreadsheet » Test Suites & Cases » Document Editing » Edit Test Case

Title *

Verify CSV import with attached test data file

Ex: Opening a simple log file

Section *: Prerequisites

Type *: Functionality

Priority *: 4 - Must Test

Estimate: 15m

Milestone:

References:

Description

Verifies the CSV import functionality for the template module. No document has been opened and all user settings have been reset to default values before executing this test case.

Steps

1 Open the New Document wizard by selecting File | New.

- The dialog is displayed
- All default templates have been loaded
- The Empty template has been pre-selected

2 Select the Corporate template and confirm it by clicking OK.

- A new document is opened, using the Corporate template
- The page counter in the statusbar displays '1/1'
- The document area is focused

Attachments: None.

Actions: Delete a test case to remove it from its test suite. This also deletes all related running tests.

» Delete this test case

Edit test cases in TestRail.

Summary

Introduction

Tools for
Review & insp

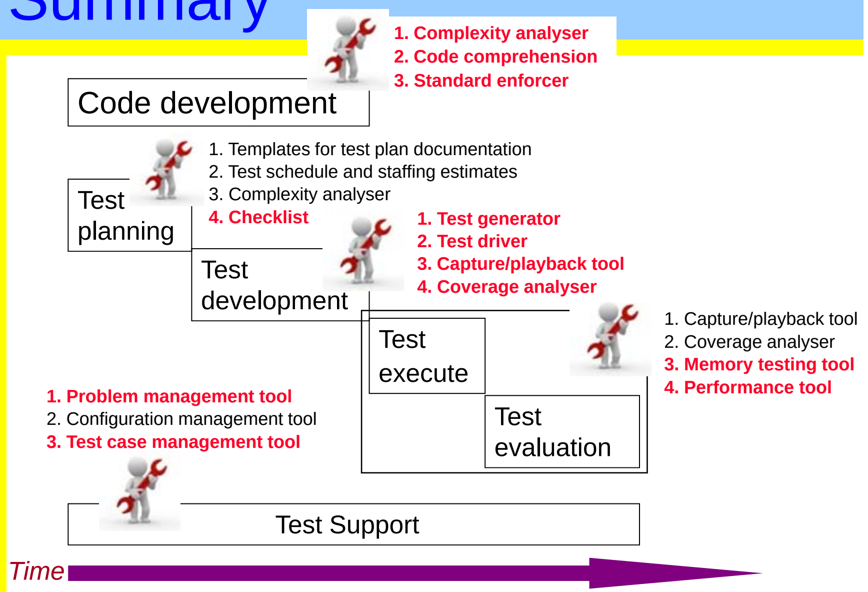
Test planning

Test develop

Test exe. & ev

Test support

Automate eg



1. Complexity analyser
2. Code comprehension
3. Standard enforcer

Code development

1. Templates for test plan documentation
2. Test schedule and staffing estimates
3. Complexity analyser
4. Checklist

Test planning

1. Test generator
2. Test driver
3. Capture/playback tool
4. Coverage analyser

Test development

1. Capture/playback tool
2. Coverage analyser
3. Memory testing tool
4. Performance tool

Test execute

Test evaluation

1. Problem management tool
2. Configuration management tool
3. Test case management tool

Test Support

Time

Summary

Introduction

Tools for
Review & insp

Test planning

Test develop

Test exe. & ev

Test support

Automate eg

Test Tool	Review & Inspection **	Test Planning	Test Design	Execution, Result Evaluation
Complexity analyser	×	×		
Code comprehension	×			
Syntax and semantic analyser	×			
Test documentation templates		×		
Schedule and staffing estimate		×		
Checklist		×		
Test data generator			×	
Test driver			×	
Coverage analyser			×	×
Capture/ playback			×	×
Memory test				×
Snapshot				×
Simulators and performance				×
Test harness generator				×
Problem management	×			×
Configuration management	×	×	×	×

Test tools Used by the Microsoft Solution for Management (MSM) test team

Introduction

Tools for
Review & insp

Test planning

Test develop

Test exe. & ev

Test support

Automate eg

- **Product Studio:** A bug tracking tool.
- **Test Management System:** A tool used to coordinate, record, and track all the test activities.
- **ADTest:** A tool used to generate load on Active Directory.
- **Print Stress:** Used to generate load on the print server.
- **NtBench.** A tool for characterizing disk data transfer performance.
- **WAN Simulator.** A hardware device or software tool that enables simulation of a variety of network speeds, bandwidths, latency, and conductivity.

Some Open Source Tools

Introduction

Tools for
Review & insp

Test planning

Test develop

Test exe. & ev

Test support

Automate eg

- Open sourced testing tools: www.opensourcetesting.org
- Testing .NET: NUnit, csUnit, dotUnit
- Coverage tool for Java: Cobertura, EMMA
- Performance test tool: JMeter, TestMaker, OpenSTA
- C++ unit testing: TUT
- Delphi coverage tool: Discover
- Source instrumenting profiling tool for checking performance and identify bottleneck: GpProfile
- Stress Test Tools: Microsoft Web Application Stress Test Tools
- Performance Test Tools: Microsoft Performance Monitor Tools

Some Open Source Tools

Introduction
Tools for Review & insp
Test planning
Test develop
Test exe. & ev
Test support
Automate eg

Two websites that list various test tools:

http://www.testingfaqs.org/
http://www.fileguru.com/downloads/software_testing

With tutorial for beginner:

1. CodeCover - for JAVA programs, whiteboard testing.
   http://codecover.org/index.html

2. Eclipse with JUnit - for JAVA programs, automated unit testing.

   Eclipse is a JAVA IDE environment. Any JAVA program written or loaded in Eclipse can be tested by JUnit (the testing module).
   http://eclipsetutorial.sourceforge.net/totalbeginnerlessons.html

A Test Automation Example: Company HKX

Introduction
Tools for Review & insp
Test planning
Test develop
Test exe. & ev
Test support
Automate eg

Organization

- 20 users
- 30 IT staff

IMS (Information Management System)

- 100 person-year effort
- 2 years development
- Outsourced to PcW
- Release in 2 major phases (1st phase, Dec. 2000, 2nd phase, Dec. 2001)
- Many minor releases.

Automate Regression Test

Introduction
Tools for Review & insp
Test planning
Test develop
Test exe. & ev
Test support
Automate eg

- For UAT, 20 users run tests for 4-5 weeks
- **Objective**: reduce this testing effort for future minor and major releases

Solution

- Automate the running of a subset of UAT test cases.

How?

- Use test case management tool, and capture/playback tool

Project Plan

Introduction
Tools for Review & insp
Test planning
Test develop
Test exe. & ev
Test support
Automate eg

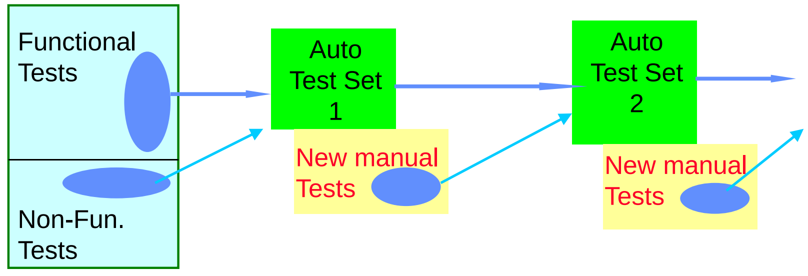
1. Consultant recommends test tools
   (study WinRunner, QArun; recommend WinRunner)
2. Training sessions
   (Organized 3 training sessions: 2 for users (1/2 day training), 1 for IT staff and manager (1 day training))
3. On-site assistant on using the tools
   (consultant shows users how to use the tool)
4. Run a pilot study with assistant of the consultant
   (Help users to debug and write test scripts)
5. Evaluate pilot results
   (interview users, collect feedback, develop a best practice guide)
6. Fully implement the test tools

What Test Cases to Automated?

- Critical functional tests (those unlikely to change)
- Selected non-functional tests (some load tests, some performance tests)

As it takes a substantial effort to automate a test case, we are likely not able to automate more than 10-15% of all test cases.

Automated Test Process



Tests used to test the **original release** (1st release)

Tests used to test the **2nd release** (with some new functions)

Tests used to test the **3rd release** (with some new functions)

Pilot Results

- Users run about 800 test cases
- Only 35% of these can be automated

Why the other test cases cannot be automated?

- Impossible cases (60%)

  e.g, testing recovery,

  running tests at different places => test case cannot be repeated!!
- Limitation of the tools (40%)

  E.g., test cases involving multiple users not supported

Lesson Learned

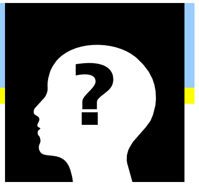
- It takes time to learn and become familiar with the feature of tools estimate: need 1 day training, and 4-5 days of using the tools before becoming productive.
- Be prepare to spend effort to maintain the test scripts. When I/O changes, the scripts have to be updated.
- Automate only after the I/O have been finalized/stable.

- Make sure the organization develops at least one staff to become an expert on the tool – external helps are expensive!
- Tools are not easy to use by 'end users'. Need to understand programming to modify the test scripts

Recommendations

- Organize the test cases into **related groups**. For test cases requiring the same setup, run them together. Then, maintaining the test cases will be easier.
- Assign < 5 staff to learn and do the automation. This reduces overhead for learning and limit the productivity impact.
- Tools are releasing new features every 6 months. The staff will need to upgrade their knowledge.

Exercise



Your Suggestion?

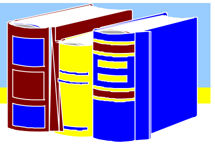
Your friend has just started a software development company that focuses on internet applications. He is planning to set up a test team of 5.

Being a start-up, the company can only afford to buy 2 test tools.

What would you recommend to your friend?

Clearly explain the rationale for your recommendation.

References



Fewster, M., and Graham, D., *Software Test Automation*, Addison-Wesley, 1999

Smale, A., *Test automation experience at Microsoft*

Web Resources

1. Links to many test tools, http://www.sqa-test.com/toolpage.html
2. Microsoft’s Web Application Stress tool
3. http://www.aptest.com/resources.html
4. www.softwareqatest.com
5. Neoload, http://www.neotys.com/load-testing-tool/demo.html#