



CUI Abbottabad

Department of Computer Science

SOFTWARE TESTING

Lecture 17

Debugging

1

AGENDA

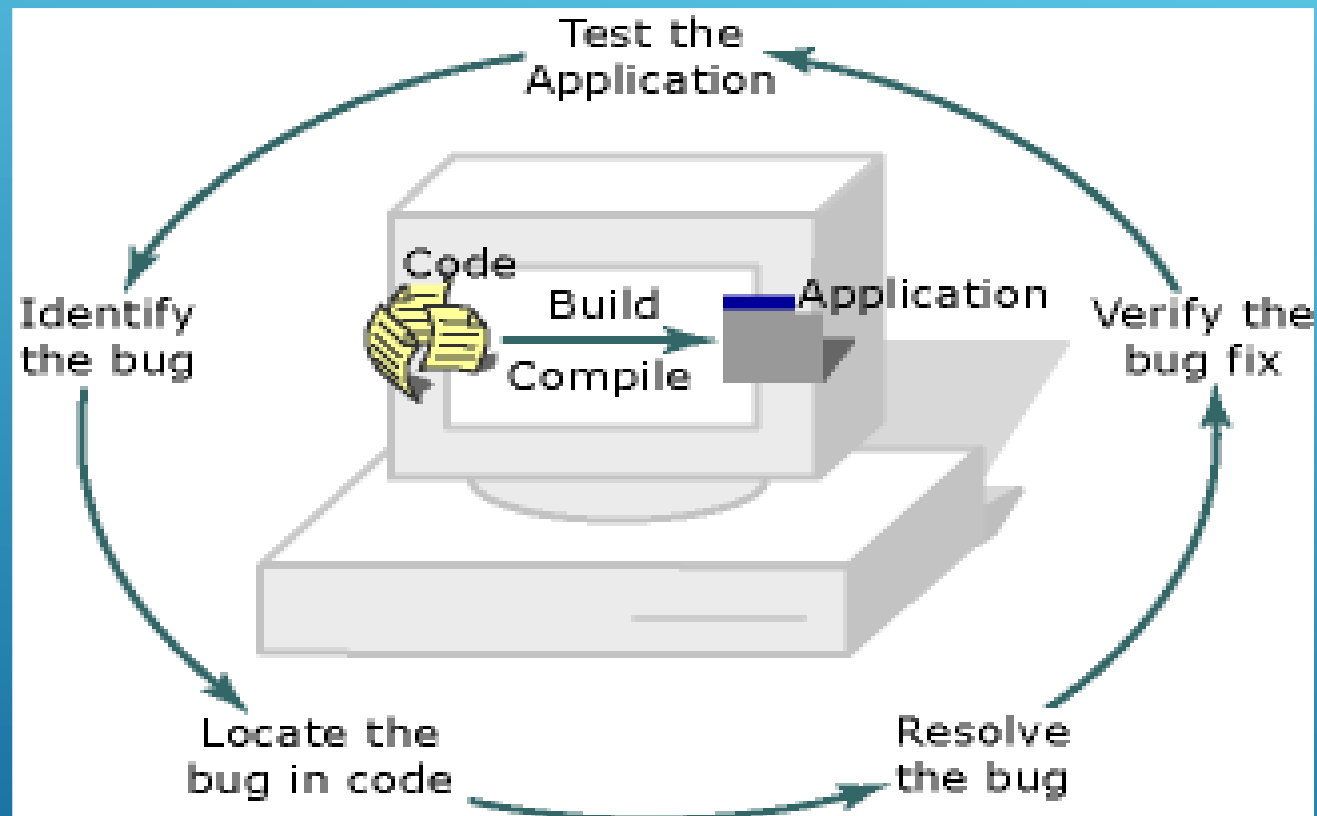


- ▶ Debugging
- ▶ Debugging Life Cycle
- ▶ The Debugging Process
- ▶ Difference between Testing & Debugging
- ▶ Bug and Bug Life Cycle
- ▶ Bug Management
- ▶ Reporting
- ▶ Bug/Defect Types

DEBUGGING

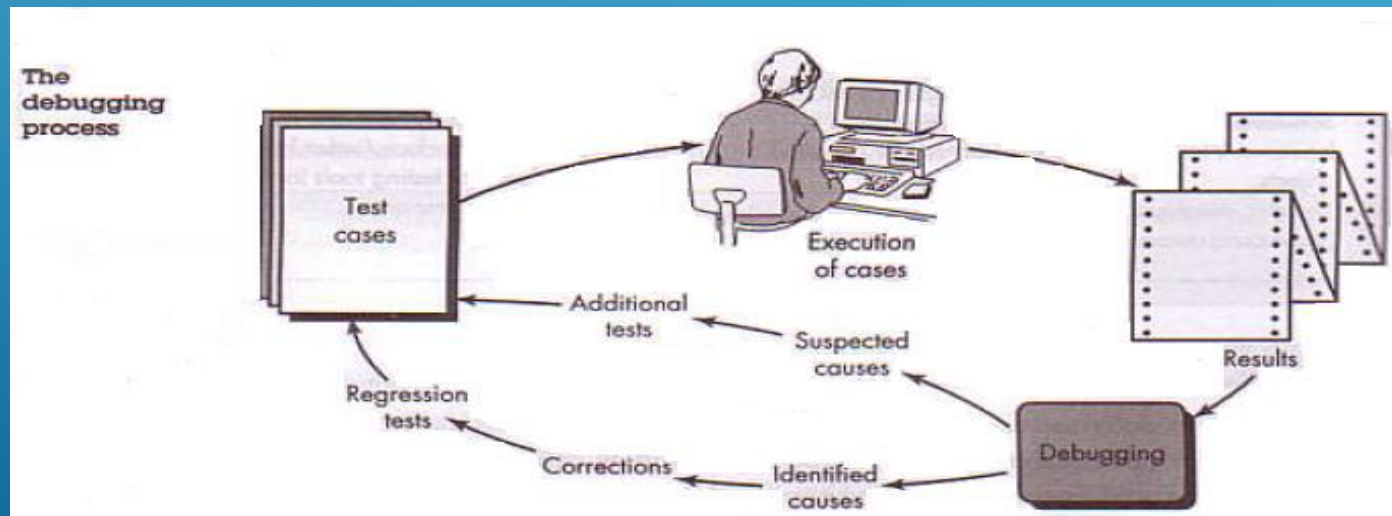
- ▶ Debugging is that activity which is performed after executing a successful test case.
- ▶ Remember that a successful test case is one that shows that a program does not do what it was designed to do.
- ▶ Debugging is a two-step process that begins when you find an error as a result of a successful test case.
 - ▶ Step 1: is the determination of the exact nature and location of the suspected error within the program.
 - ▶ Step 2: consists of fixing the error.
- ▶ *Locating the error represents about 95% of the activity*

DEBUGGING LIFE CYCLE



DEBUGGING PROCESS

- ▶ Debugging will always have one of two outcomes:
 - ▶ The cause will be found and corrected or
 - ▶ The cause will not be found.
- ▶ In the latter case, the person performing debugging may suspect a cause, design one or more test cases to help validate that suspicion, and work toward error correction in an iterative fashion.



TESTING VS DEBUGGING

TESTING

Testing is the process of executing a program or system with the aim of finding errors or bugs.

DEBUGGING

Correcting these errors or bugs (found during testing) is debugging.



BUG

What is a BUG

- ▶ A fault in a program, which causes the program to perform in an unintended or unanticipated manner.
- ▶ Reports detailing bugs in a program are commonly known as bug reports, fault reports, problem reports, trouble reports, defect reports etc.

Why BUG Occurs

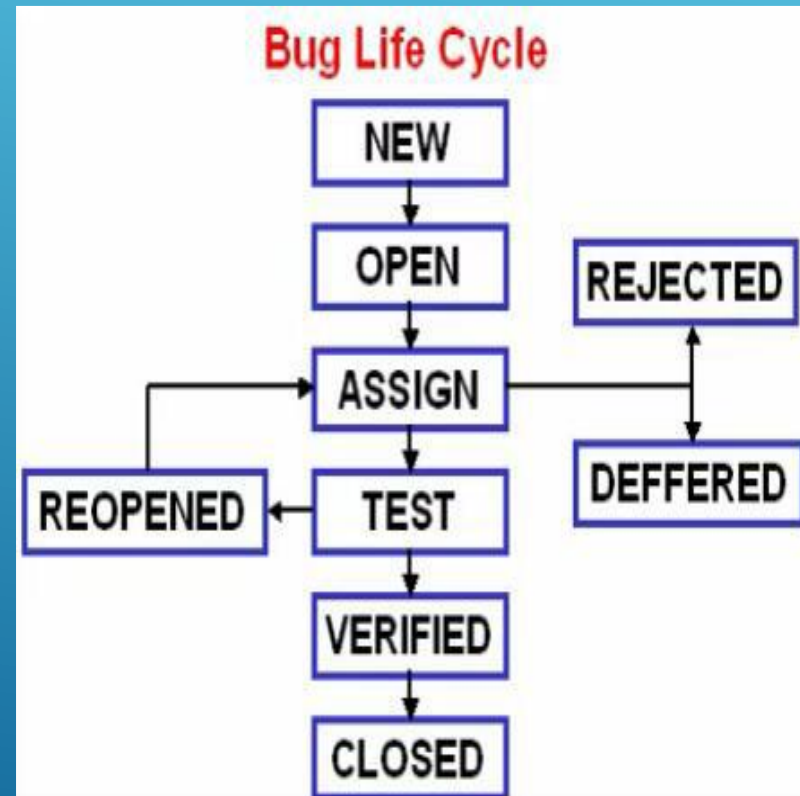
- ▶ There are so many reasons that can cause a bug some of them can be:
 - ▶ Syntax errors in the Codes
 - ▶ logical errors in the Codes
 - ▶ Unfinished requirements
 - ▶ Misunderstanding of user needs
 - ▶ Errors in the design documents
 - ▶ Lack of documentation

BUG LIFECYCLE

- ▶ In software development process, the bug has a life cycle. The bug should go through the life cycle to be closed. A specific life cycle ensures that the process is standardized. The bug attains different states in the life cycle. The life cycle of the bug can be shown diagrammatically as follows:

- ▶ **States of a bug:**

- 1) *New*
- 2) *Open*
- 3) *Assign*
- 4) *Test*
- 5) *Verified*
- 6) *Deferred*
- 7) *Reopened*
- 8) *Rejected*
- 9) *Closed*



BUG STATES

- ▶ **New:** When the bug is posted for the first time, its state will be NEW. This means that the bug is not yet approved.
- ▶ **Open:** After a tester has posted a bug, the lead of the tester approves that the bug is genuine and he changes the state as OPEN.
- ▶ **Assign:** Once the lead changes the state as OPEN, he assigns the bug to corresponding developer or developer team. The state of the bug now is changed to “ASSIGN”.
- ▶ **Resolved/Fixed/Test:** When developer makes necessary code changes and verifies the changes then he/she can make bug status as „Fixed“ and the bug is passed to testing team.
- ▶ **Deferred:** The bug, changed to deferred state means the bug is expected to be fixed in next releases. The reasons for changing the bug to this state have many factors. Some of them are priority of the bug may be low, lack of time for the release or the bug may not have major effect on the software.

BUG STATES

- ▶ **Rejected/Invalid:** Some times developer or team lead can mark the bug as Rejected or invalid if the system is working according to specifications and bug is just due to some misinterpretation.
- ▶ **Pending Retest:** After the bug is fixed, it is passed back to the testing team to get retested and the status of „Pending Retest“ is assigned to it.
- ▶ **Retest:** The testing team leader changes the status of the bug, which is previously marked with „Pending Retest“ to „Retest“ and assigns it to a tester for retesting.
- ▶ **Duplicate:** If the bug is repeated twice, then one bug status is changed to “DUPLICATE”.
- ▶ **Verified:** Once the bug is fixed and the status is changed to TEST, the tester tests the bug. If the bug is not present in the software, he approves that the bug is fixed and changes the status to VERIFIED.

BUG STATES

- ▶ **Could not reproduce:** A 'non-reproducible bug' is one such bug that occurs at a certain point of time or is triggered by a particular action, which are not easy to trace or recreate or reproduce by the testers. Such bugs are hard to replicate later on can become a challenge for a tester to deal with. If developer is not able to reproduce the bug by the steps given in bug report by QA then developer can mark the bug as "CNR". QA needs action to check if bug is reproduced and can assign to developer with detailed reproducing steps.
- ▶ **Reopened:** If the bug still exists even after the bug is fixed by the developer, the tester changes the status to REOPENED. The bug traverses the life cycle once again.
- ▶ **Closed:** Once the bug is fixed, it is tested by the tester. If the tester feels that the bug no longer exists in the software, he changes the status of the bug to CLOSED. This state means that the bug is fixed, tested and approved.
- ▶ **Postponed:** Sometimes, testing of a particular bug has to be postponed for an indefinite period. This situation may occur because of many reasons, such as unavailability of Test data, unavailability of particular functionality etc. That time, the bug is marked with "Postponed" status.

BUG MANAGEMENT

- ▶ It is common practice for software to be released with known bugs that are considered non-critical. While software products may, by definition, contain any number of unknown bugs.
- ▶ Most big software projects maintain two lists of "known bugs"— those known to the software team, and those to be told to users. This is not dishonesty, but users are not concerned with the internal workings of the product. The second list informs users about bugs that are not fixed in the current release, or not fixed at all, and a workaround may be offered.
- ▶ There are various reasons for not fixing bugs:
 - ▶ The developers often don't have time or it is not economical to fix all non-severe bugs.
 - ▶ The bug could be fixed in a new version or patch that is not yet released.
 - ▶ The changes to the code required to fix the bug would be large, and would bring with them the chance of introducing other bugs into the system.
 - ▶ It's "not a bug". A misunderstanding has arisen between expected and provided behaviour.

COMMON TYPES OF BUGS

- ▶ **Math bugs**
 - ▶ Division by zero
 - ▶ Arithmetic overflow or underflow
- ▶ **Logic bugs**
 - ▶ Infinite loops and infinite recursion
- ▶ **Syntax bugs**
 - ▶ Use of the wrong operator, such as performing assignment instead of equality
- ▶ **Resource bugs**
 - ▶ Using an un-initialized variable
 - ▶ Resource leaks, where a finite system resource such as memory or file handles are exhausted by repeated allocation without release.
 - ▶ Buffer overflow, in which a program tries to store data past the end of allocated storage.
- ▶ **Team working bugs**
 - ▶ Comments out of date or incorrect: many programmers assume the comments accurately describe the code
 - ▶ Differences between documentation and the actual product

REPORTING

- ▶ **Daily Summary Data** : In addition to entering new bugs and closing fixed bug in the bug tracking application,
 - ▶ Number of test cases completed
 - ▶ Number of new issues identified and reported
 - ▶ The current list of existing bugs by severity
- ▶ **Weekly Summary Report** : At the end of each week, creates a weekly report to provide some additional information to clients.
- ▶ **End of Cycle Reports**: QA engineers execute all test cases for a build, supply a complete set of test results for the cycle. provide the following information:
 - ▶ List of bugs found in this testing cycle
 - ▶ List of bugs fixed in this testing cycle
 - ▶ List of bugs deferred to a later release

BUG/DEFECT TYPES

- ▶ Defects that are detected by the tester are classified into categories by the nature of the defect. The following are the classification :
 1. **Showstopper (X):** The impact of the defect is severe, and the system cannot go into the production environment without resolving the defect since an interim solution may not be available.
 2. **Critical (C):** The impact of the defect is severe; however, an interim solution is available.
 3. **Noncritical (N):** These are also the defects that could potentially be resolved via documentation and user training. These can be GUI defects.



Bug/Defect Types



Defect Report



Methods of Debugging

Debugging by Brute Force
Attack

Debugging by Induction

Debugging by Deduction

Debugging by Backtracking

Debugging by Testing

AGENDA

DEFECT REPORT

Particulars that have to be filled by a tester are:

- ▶ **Defect Id:** Number associated with a particular defect, and henceforth referred by its ID.
- ▶ **Date of execution:** The date on which the test case which resulted in a defect was executed.
- ▶ **Severity:** As explained, it can be Critical, Non-Critical and Showstopper.
- ▶ **Module ID:** Module in which the defect occurred.
- ▶ **State:** New, Open, Assign, Test, Verified, Deferred, Reopened, Rejected, Close.
- ▶ **Defect description:** Description as to how the defect was found, the exact steps that should be taken to simulate the defect, other notes and attachments if any.
- ▶ **Test Case Reference No:** The number of the test case which resulted in the defect.

DEFECT REPORT

- ▶ **Owner:** The name of the tester who executed the test case.
- ▶ **Test case description:** The instructions in the test cases for the step in which the error occurred.
- ▶ **Expected Result:** The expected result after the execution of the instructions in the test case descriptions.
- ▶ **History of the defect:** Normally taken care of the automated tool used for defect tracking and reporting.
- ▶ **Attachments:** The screen shot showing the defect should be captured and attached.
- ▶ **Responsibility:** Identified team member of the development team for fixing the defect.

Bug Reporting in WORD

brought to you by usersnap.com

Bug Report: #Bug-ID

Bug ID	#Bug-ID (e.g. #test_ABC, #123, #home_123) —
Tester	<i>Han Solo</i>
Date (submitted)	<i>01.11.2018</i>
Title	<i>CONTACT FORM - No confirmation message is shown</i>
Bug Description	
URL	<i>https://milleniumfalcon.rebells/contact-us</i>
Summary	The page where the contact form is on does not show any confirmation message after submitting a contact request.
Screenshot	see attached screenshots
Platform	Mac OS 10.14
Browser	Chrome 69.0.3497.100 (Official Build) (64-bit)
Administrative	
Assigned To	<i>Chewbacca</i>
Assigned At	<i>01.11.2018</i>
Priority	High
Severity	Critical

Additional Notes:

- **Step-by-step Description:**
 - a. Filled out contact form
 - b. clicked on submit
 - c. form loaded a while
 - d. contact form showed "<empty>"
- **Screenshots:**
- **Result - Should:** "Thank you for contacting us, we will get in touch shortly"
- **Result - Is:** "<empty>"

Bug tracking is even easier with usersnap.com.
[Test us for free - no credit card required](#)

Bug Reporting in EXCEL

	A	B	C
1	Category	Label	Value
2	Bug ID	ID number	#123
3		Name	CART - Unable to add new item to my cart
4		Reporter	Mike A
5		Submit Date	03/04/16
6	Bug overview	Summary	When my cart contains one item, I am unable to add a second item via the add to cart button on a product page
7		URL	www.example.com/product/abc
8		Screenshot	www.example.com/screenshot123
9	Environment	Platform	Macintosh
10		Operating System	OS X 10.12.0
11		Browser	Chrome 53
12	Bug details	Steps to reproduce	add one item to cart > go to product abc via the search bar > add new item to cart via "add to cart" button (see screenshot) > go to cart
13		Expected result	The cart should contain 2 items
14		Actual result	The cart contains only 1 item
15		Description	/
16	Bug tracking	Severity	Major
17		Assigned to	/
18		Priority	High
19	Notes	Notes	/

Bug Reporting in JIRA

Header - wrong image shown

[Edit](#) [Comment](#) [Assign](#) [To Do](#) [In Progress](#) [Done](#)

[Share](#) [Download](#) [More](#)

Type: 🔴 Bug Status: TO DO [\(View workflow\)](#)

Priority: ↑ Medium Resolution: Unresolved

Labels: None

Environment: Chrome Version 70.0.3538.77 (Official Build) (64-bit) -
Screen Resolution: 1920 x 947

Sprint:

Assignee: 👤 Unassigned

[Assign to me](#)

Reporter: 👤 Usersnap Developer

Votes: 0

Watchers: 1 [Stop watching this issue](#)

Created: 14 minutes ago

Updated: Just now

Description

Components:
on homepage

Description:
Outdated header image on www.usersnap.com

Steps to reproduce:

- 1) Go to usersnap.com
- 2) check the header image

Javascript console errors:

no error messages

Result:

expected Result:

header_image_2.jpeg

actual Result:

header_image_1.jpeg

Development


[Create branch](#)

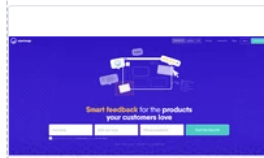
Agile

[View on Board](#)

Attachments

...

 Drop files to attach, or [browse](#).



METHODS OF DEBUGGING

The methods of debugging are listed below:

1. Debugging by Brute Force Attack
2. Debugging by Induction
3. Debugging by Deduction
4. Debugging by Backtracking
5. Debugging by Testing

1. DEBUGGING BY BRUTE FORCE

- ▶ The most common scheme for debugging a program is the “brute force” method. It is popular because it requires little thought and is the least mentally taxing of the methods, but it is inefficient and generally unsuccessful.
- ▶ Brute force methods can be partitioned into at least three categories:
 - 1) **Debugging with a storage dump.**
 - 2) **Debugging according to the common suggestion to “scatter print statements throughout your program.”**
 - 3) **Debugging with automated debugging tools.**

1.1. DEBUGGING BY STORAGE DUMP

- ▶ Storage dump is the display or printout of the contents of memory. Or display of all storage locations in hexadecimal or octal format.
- ▶ When a program abends, a memory dump can be taken in order to examine the status of the program at the time of the crash.
- ▶ The programmer looks into the buffers to see which data items were being worked on when it failed. Counters, variables, switches and flags are also inspected.
- ▶ Brute force debugging method is applied when all else fails
- ▶ It is the most inefficient of the brute force methods. Here's why;
 - ❑ *There's a massive amount of data, most of which is irrelevant.*
 - ❑ *A memory dump is a static picture of the program, showing the state of the program at only one instant in time; to find errors, you have to study the dynamics of a program (state changes over time).*

```
txdns.exe -fm smalllist.txt microsoft.com
TXDNS <http://www.txdns.net> 2.0.0 running STAND-ALONE Mode
accounting.microsoft.com 207.46.155.50
billing.microsoft.com 65.54.157.250
directory.microsoft.com 131.197.115.87
msn1.microsoft.com 207.46.232.180
example.microsoft.com 207.46.232.182
exchange.microsoft.com 131.197.236.102
ftp.microsoft.com 207.46.236.102
gallery.microsoft.com 131.197.236.102
mail.microsoft.com 131.197.236.102
mall.microsoft.com 131.197.236.102
public.microsoft.com 131.197.236.102
research.microsoft.com 131.197.236.102
services.microsoft.com 207.46.232.180
shop.microsoft.com 207.46.232.182
smn1.microsoft.com 131.197.115.214
windows.microsoft.com 207.46.157.134
Resolved names: 16
Failed queries: 349
Total queries: 366
```


1.2. DEBUGGING USING PRINT STATEMENTS

- ▶ Scattering statements throughout a failing program to display variable values is better than a dump as it is not static and shows the dynamics of a program, but this method, too, has many shortcomings:
- ❑ Again, you are not thinking.
- ❑ It requires you to change the program; such changes can mask the error or introduce new errors.
- ❑ It may work on small programs, but the cost of using it in large programs like operating systems is quite large.

```
#include <iostream>
#include <string>
using namespace std;

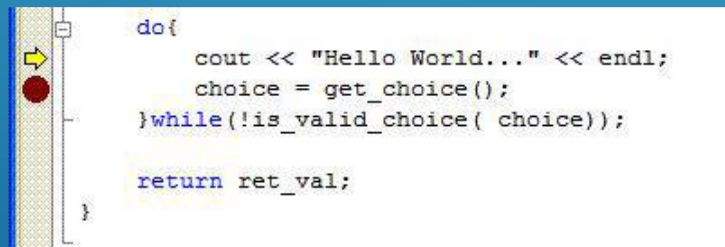
int main () {

    int num1, num2;

    cout << "Welcome to the Addition-only calculator. Please enter the first value: ";
    cin >> num1;
    cout << endl << "You have entered " << num1 << " as the first value. Please enter the second value: ";
    cin >> num2;
    cout << endl << num1 << " + " << num2 << " = " << num1 + num2;
    return 0;
}
```

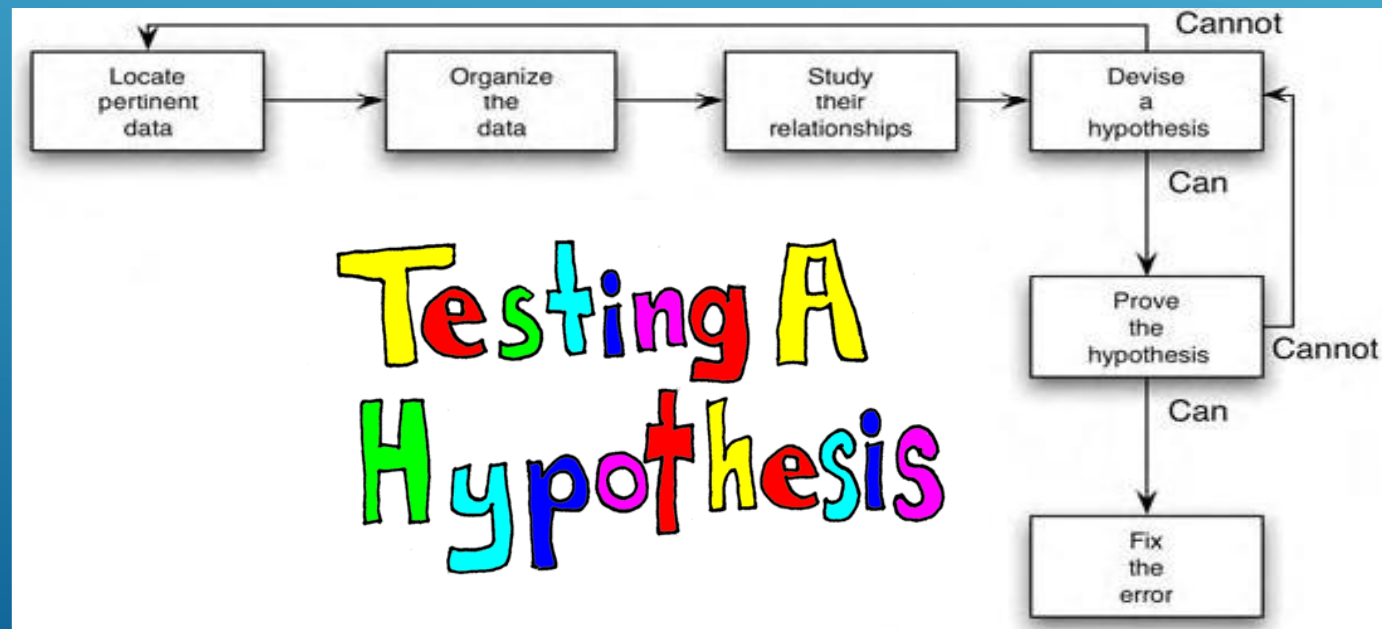
1.3. DEBUGGING USING AUTOMATED DEBUGGING TOOLS

- ▶ Automated debugging tools work similarly to inserting print statements within the program, but rather than making changes to the program, you analyse the dynamics of the program with the debugging features of the programming language.
- ▶ A common function of debugging tools is the ability to set breakpoints that cause the program to be suspended when a particular statement is executed or when a particular variable is altered, and then the programmer can examine the current state of the program.



2. DEBUGGING BY INDUCTION

- ▶ It should be obvious that careful thought will find most errors
- ▶ One particular thought process is induction, where you move from the particulars of a situation to the whole. That is, start with the clues (the symptoms of the error, possibly the results of one or more test cases) and look for relationships among the clues. The induction process is illustrated in Figure given below:



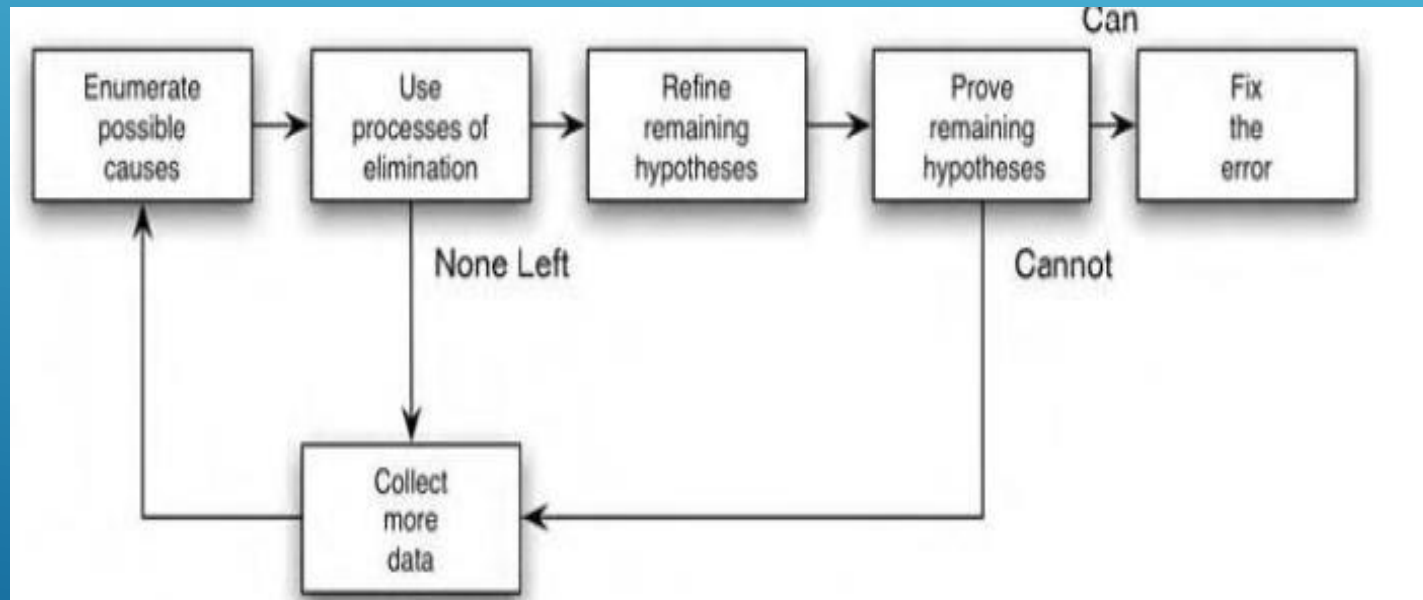
2. DEBUGGING BY INDUCTION

The steps are as follows:

- 1) **Locate the pertinent data.** A major mistake debuggers make is failing to take account of all available data or symptoms about the problem. The first step is the enumeration of all you know about what the program did correctly and what it did incorrectly
- 2) **Organize the data.** Remember that induction implies that you're processing from the particulars to the general, so the second step is to structure the pertinent data to let you observe the patterns.
- 3) **Devise a hypothesis.** Next, study the relationships among the clues and devise, using the patterns that might be visible in the structure of the clues, one or more hypotheses about the cause of the error.
- 4) **Prove the hypothesis.** A major mistake at this point, given the pressures under which debugging usually is performed, is skipping this step and jumping to conclusions to fix the problem. If you skip this step, you'll probably succeed in correcting only the problem symptom, not the problem itself.
- 5) **Fix the Error.** If hypothesis is proved successfully then go on to fix the error else repeat steps 3 and 4

3. DEBUGGING BY DEDUCTION

- The process of deduction proceeds from some general theories or premises, using the processes of elimination and refinement, to arrive at a conclusion (the location of the error).

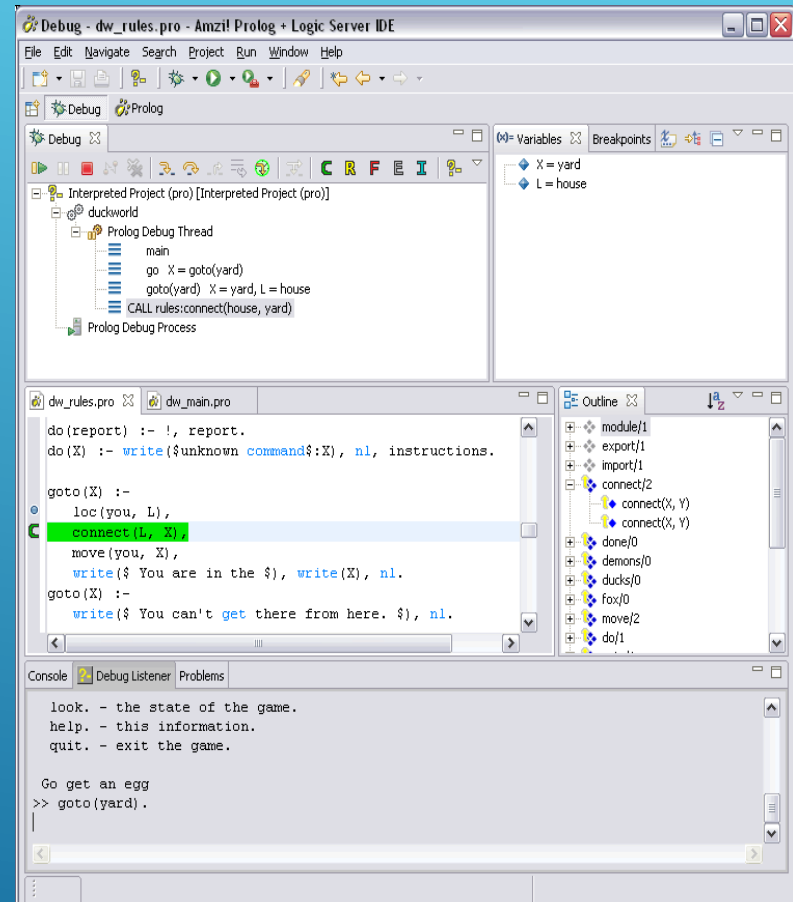


3. DEBUGGING BY DEDUCTION

- ▶ ***Enumerate the possible causes or hypotheses.*** The first step is to develop a list of all conceivable causes of the error. They don't have to be complete explanations; they are merely theories to help you structure and analyze the available data.
- ▶ ***Use the data to eliminate possible causes.*** Carefully examine all of the data, particularly by looking for contradictions, and try to eliminate all but one of the possible causes. If all are eliminated, you need more data through additional test cases to devise new theories.
- ▶ ***Refine the remaining hypothesis.*** The possible cause at this point might be correct, but it is unlikely to be specific enough to pinpoint the error. Hence, the next step is to use the available clues to refine the theory.
- ▶ ***Prove the remaining hypothesis.*** This vital step is identical to step 4 in the induction method.

4. DEBUGGING BY BACKTRACKING

- ▶ Backtracking is a fairly common debugging approach that can be used successfully in small programs
- ▶ Beginning at the site where a symptom has been uncovered, the source code is traced backwards till the error is found.
- ▶ Unfortunately, as the no of source lines increases, the no of potential backward paths may become unmanageably large



5. DEBUGGING BY TESTING

- ▶ The last “thinking type” debugging method is the use of test cases. consider two types of test cases:
- ▶ Test cases for testing, where the purpose of the test cases is to expose a previously undetected error, and
- ▶ Test cases for debugging, where the purpose is to provide information useful in locating a suspected error.
- ▶ The difference between the two is that test cases for testing tend to be “fat” because you are trying to cover many conditions in a small number of test cases. Test cases for debugging, on the other hand, are “slim” since you want to cover only a single condition or a few conditions in each test case.
- ▶ In other words, after a symptom of a suspected error is discovered, you write variants of the original test case to attempt to pinpoint the error. Actually, this method is not an entirely separate method; it often is used in conjunction with the induction and deduction method.