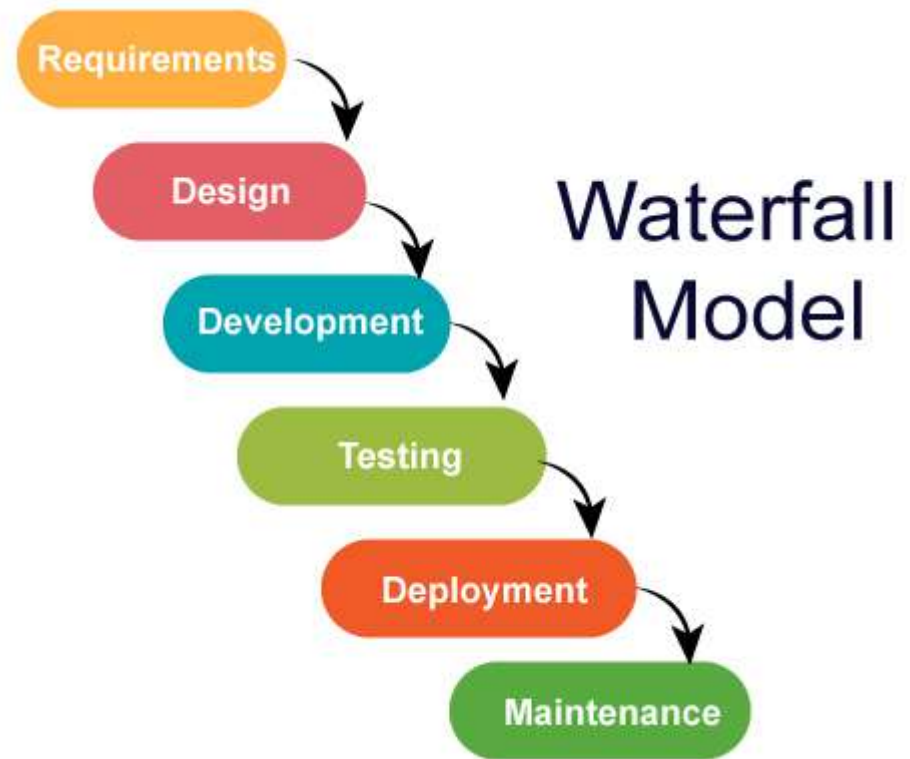


Debugging

Rupesh Nasre.
rupesh@cse.iitm.ac.in

Software Engineering Life Cycle

- For established products
- Different people work in different phases.
- Deliverables of a team for input to the next.
- Unsuitable for startups / uncertain projects.
- Also includes documentation.



Human Activity and Errors

- Humans are inherently not precise.
 - Their tasks are prone to errors.
- When programming is a human activity, errors are inevitable.
 - This does not mean programs generated by AI agents are error-free. Why?
- Finding and fixing errors in large programs can be complex and time-consuming.
 - Real-world code is written by several teams.

Debugging

- Art and science of removing bugs from programs.
- In 1944, Robert Oppenheimer (WWII) used the term in a letter, in the context of aeroplane engine testing.
- In 1947, engineers working on the Mark II Aiken Relay Calculator at Harvard found a moth trapped in a relay that was causing errors.
 - This popularised the term *debugging*.

9/9

0800 Antan started
 1000 " stopped - antan ✓
 13" sec (032) MP - MC ~~1.982647000~~
 (033) PRO 2 2.130476415 (3) 4.615925059 (-2)

concd 2.130676415

Relays 6-2 in 033 failed special speed test
 in relay " 10.000 test.

Relay
 2145
 Relay 3370

1100 Started Cosine Tape (Sine check)
 1525 Started Mult + Adder Test.

1545



Relay #70 Panel F
 (moth) in relay.

First actual case of bug being found.
 1630 Antan started.
 1700 closed down.

Identifying Bugs

- Testing
 - Unit testing
 - Integration testing
 - Inherently incomplete
- Verification
 - Proof that the program is correct.

If debugging is the process of removing software bugs,
then programming must be the process of putting them in.

Edsger Dijkstra

Testing

- Compile often.
 - Compiler helps us both with syntax and types.
 - Aim for zero warnings.
- Test incrementally.
 - For list functions print, insert, and remove, test the functionality of insert + print.
 - Add prints in empty function stubs to check flow.
 - Simplification first, then enhancement
 - Correctness first, then optimization

Unit Testing

- First, be a friend.
 - Check basic functionality
 - Simple inputs
- Then, be an adversarial friend.
 - Try to break the program
 - Check extreme cases (input values)
 - Design complex scenarios (combination of inputs)
 - Change environment (shell variables, install in non-default paths, another OS, another hardware, ...)

Unit Testing

- Work with focus
 - Typically, one function at a time.
 - Guard input arguments.
 - Guard return values from function calls.
 - Mention guarantees on return values in comments.
- Error handling
 - Out-of-bound values (e.g., student with 4-digit rollno)
 - Pointers
 - Assertions
 - Unique error ID

Documentation

- Documents
 - Requirements, Design, User Manual
 - Always separate from the code
- Comments *Debug only code, comments can lie.*
 - Closer to the code, but ...
 - Special comments can generate documentation (e.g., doxygen)
- Asserts and Errors
 - Part of the code
 - Low-level, useful for supporting the customer



How the customer explained it



How the Project Leader understood it



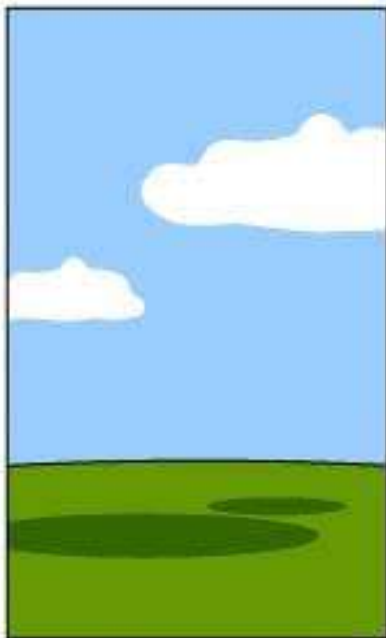
How the Analyst designed it



How the Programmer wrote it



How the Business Consultant described it



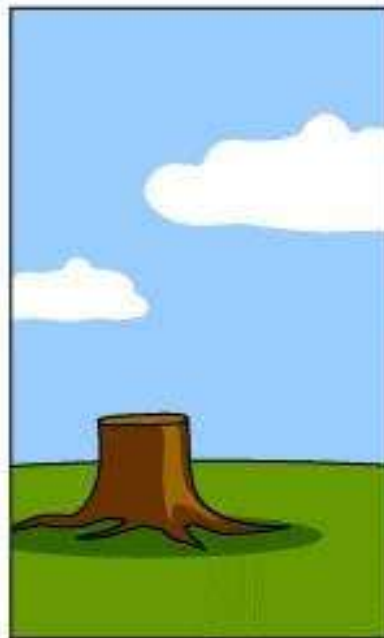
How the project was documented



What operations installed



How the customer was billed



How it was supported



What the customer really needed

`gdb`

- Command-line debugger
 - Runs an executable (e.g., `a.out`)
 - Provides control over the execution
- Supports various functionality
 - Mapping executable code back to the source
 - Stepping through the code
 - Setting breakpoints
 - Examine variable values
 - ... and even changing variable values
- Example: `1.c`

gdb: Compilation and Execution

- `gcc -g file.c`
 - Adds debug symbol information
 - Permits using source-code lines and symbols
- `gdb a.out`
 - Reads symbol information (variable names etc.)
 - Provides a prompt for user-commands: `(gdb)`
 - This is a good time to set breakpoints.
- A breakpoint is a program point where you want `a.out` to temporarily stop (that is, take a break).
 - `(gdb) b main`

gdb: Compilation and Execution

- **(gdb) b main** // a.out is not running yet
Breakpoint 1 at 0x1250: file file.c, line 34.
- **(gdb) run** // can provide command-line args
Starting program:
/home/ruresh/public_html/teaching/ssad/jan26/work/debugging/a.out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main () at file.c:34

34 **int N = 1;**

- **(gdb) quit**

**Without a
breakpoint,
the program
continues
execution.**

Example: **1.c**

Breakpoints

- We can set multiple breakpoints.
- Breakpoints can be set and deleted after the program is run.

- b main
- b function
- b 35 *// line in current file*
- break 1.c:35 *// file:line*
- break 1.c:35 if num < 100

Example: processes.c

- info breakpoints

Num	Type	Disp	Enb	Address	What
1	breakpoint	keep	y	0x00000000000001250	in main at file.c:34

- delete 1

(gdb) **b main**

Breakpoint 1 at 0x1250: file processes.c, line 34.

(gdb) **run**

...

Breakpoint 1, main () at processes.c:34

34 int N = 1;

(gdb) **s**

35 Task *first = createProcesses(N);

(gdb) **s**

createProcesses (N=1) at processes.c:22

22 if (N == 0) return NULL;

(gdb) // enter

23 Task *proc = createOneProcess();

(gdb) **n**

24 proc->next = createProcesses(N - 1);

(gdb) **s**

createProcesses (N=0) at processes.c:22

22 if (N == 0) return NULL;

(gdb) // enter

26 }

(gdb)

createProcesses (N=1) at processes.c:24

24 proc->next = createProcesses(N - 1);

(gdb)

Program received signal SIGSEGV, Segmentation fault.

0x00005555555520c in createProcesses (N=1) at processes.c:24

24 proc->next = createProcesses(N - 1);

(gdb)

Useful Commands

Command	Explanation	Example
r, run	Execute the binary	run <input>
b, break	Set breakpoint	b main
c, continue	Continue executing the program until the next breakpoint (or end of program)	c
n, next	Execute the next line fully. If it is a function call, execute the full function.	n
s, step	Execute the next line as a single step. If it is a function call, enter the function.	s
p, print	Print the value of a variable or expression	p (i + j)
info break	List current breakpoints	info breakpoints
q, quit	Quit gdb	q
watch	Pause execution as soon as a variable or expression changes.	watch ptr
bt, backtrace	Print the call-trace starting from main Useful for recursive calls	bt
<enter>	Repeat the last command Useful for n and s	<enter>

In this course...

Application	Programming	Tools
Puzzle of grass, goat, lion	Pointers, structures	Editor, visualization
Data maintenance	File I/O	I/O redirection, make
Item grouping	Linked Lists	Debugging
Gene sequencing	Recursion, strings	Memory profiling, ulimit
Closest match	Matrices	Compute profiling, plotting
Keep the recent	Arrays	Version control
Cryptography	Strings, bitwise operators	Library (ncurses)
Unit testing	Command-line	Testing
Web-page creation, analysis	HTML, Strings	wget, shell scripting
Game	Graphics	Library (graphics)
Basics	Python	--

Our goal: *to have fun with automation*