C++ Software Engineering

for engineers of other disciplines

RECAP 01

ALTEN

Autumn 2021

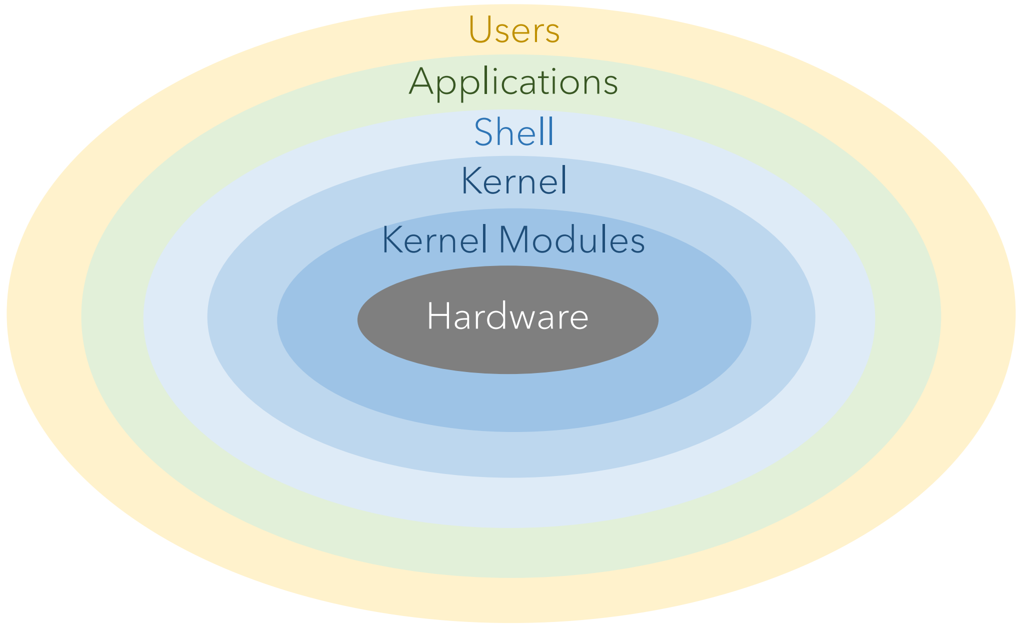
Gothenburg, Sweden

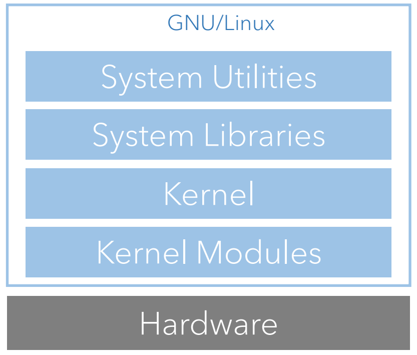
petter.lerenius@alten.se

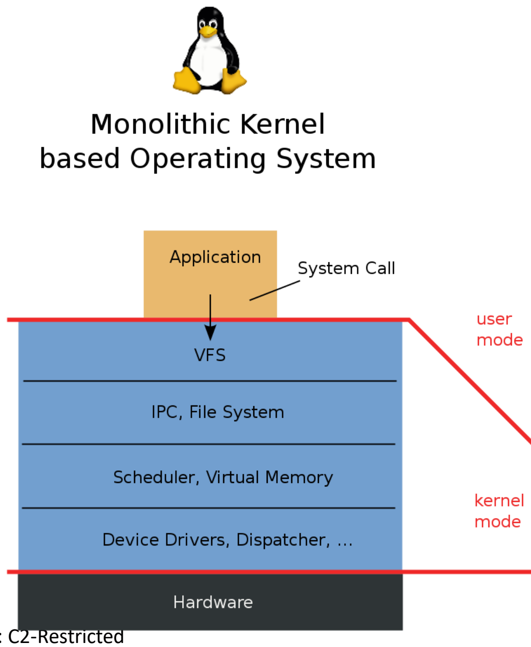
rashid.zamani@alten.se

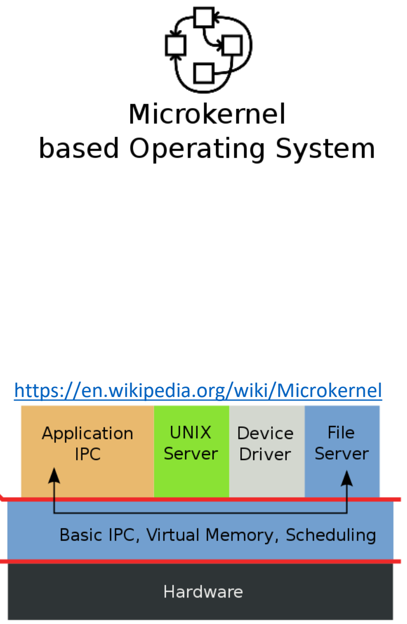
*nix

- ***Unix*** developed in 1970s, lead by the same people who invented C programming language.
- Its design is based on *Unix Philosophy* to implement *minimalist, and modular* software.






Monolithic Kernel based Operating System
Application
System Call
VFS
IPC, File System
Scheduler, Virtual Memory
Device Drivers, Dispatcher, ...
Hardware
user mode
kernel mode

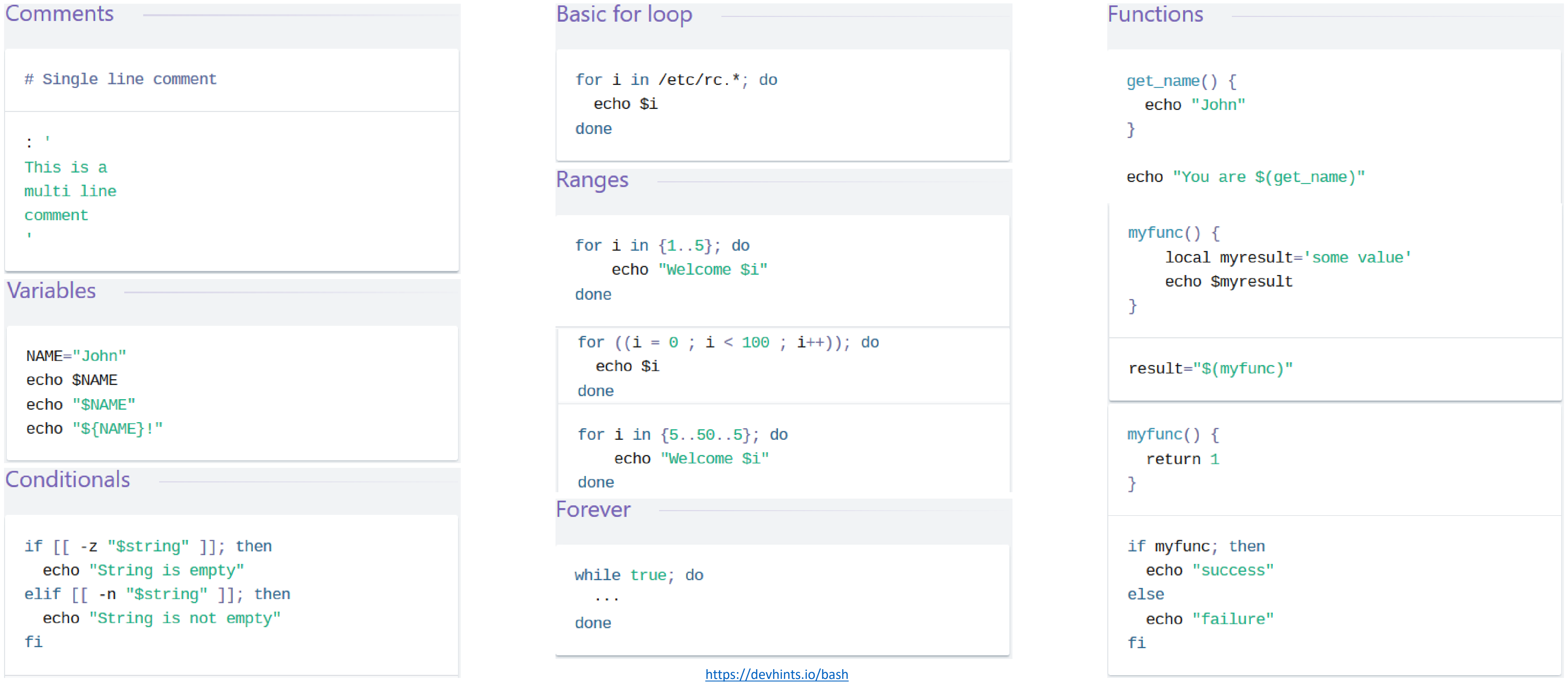

Microkernel based Operating System
Application IPC
UNIX Server
Device Driver
File Server
Basic IPC, Virtual Memory, Scheduling
Hardware

https://en.wikipedia.org/wiki/Microkernel

"The group coined the name Unics for Uniplexed Information and Computing Service (pronounced "eunuchs"), as a pun on Multics (an influential early operating system) […] "no one can remember" the origin of the final spelling Unix […] In 1983, Richard Stallman announced the GNU (short for "GNU's Not Unix") project, an ambitious effort to create a free software Unix-like system; "free" in the sense that everyone who received a copy would be free to use, study, modify, and redistribute it. The GNU project's own kernel development project, GNU Hurd, had not yet produced a working kernel, but in 1991 Linus Torvalds released the kernel Linux as free software." https://en.wikipedia.org/wiki/Unix

Bash Scripting

- ***Bash*** *"Bourne-Again SHell"* is the command language interpreter for GNU – it is also a programming language.



Comments

```
# Single line comment
```

```
: '
This is a
multi line
comment
'
```

Variables

```
NAME="John"
echo $NAME
echo "$NAME"
echo "${NAME}!"
```

Conditionals

```
if [[ -z "$string" ]]; then
  echo "String is empty"
elif [[ -n "$string" ]]; then
  echo "String is not empty"
fi
```

Basic for loop

```
for i in /etc/rc.*; do
  echo $i
done
```

Ranges

```
for i in {1..5}; do
    echo "Welcome $i"
done
```

```
for ((i = 0 ; i < 100 ; i++)); do
  echo $i
done
```

```
for i in {5..50..5}; do
    echo "Welcome $i"
done
```

Forever

```
while true; do
  ...
done
```

Functions

```
get_name() {
  echo "John"
}

echo "You are $(get_name)"
```

```
myfunc() {
    local myresult='some value'
    echo $myresult
}
```

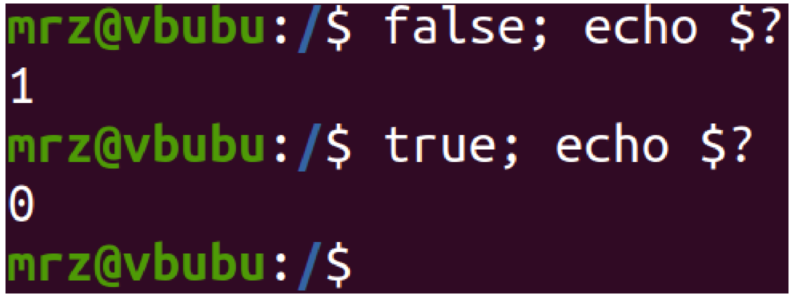
```
result="$(myfunc)"
```

```
myfunc() {
  return 1
}
```

```
if myfunc; then
  echo "success"
else
  echo "failure"
fi
```

https://devhints.io/bash

False is 1!



```
mrz@vbubu:/$ false; echo $?
1
mrz@vbubu:/$ true; echo $?
0
mrz@vbubu:/$
```

Git – cheat sheet

GIT BASICS

`git init <directory>`	Create empty Git repo in specified directory. Run with no arguments to initialize the current directory as a git repository.
`git clone <repo>`	Clone repo located at `<repo>` onto local machine. Original repo can be located on the local filesystem or on a remote machine via HTTP or SSH.
`git config user.name <name>`	Define author name to be used for all commits in current repo. Devs commonly use `--global` flag to set config options for current user.
`git add <directory>`	Stage all changes in `<directory>` for the next commit. Replace `<directory>` with a `<file>` to change a specific file.
`git commit -m "<message>"`	Commit the staged snapshot, but instead of launching a text editor, use `<message>` as the commit message.
`git status`	List which files are staged, unstaged, and untracked.
`git log`	Display the entire commit history using the default format. For customization see additional options.
`git diff`	Show unstaged changes between your index and working directory.

GIT BRANCHES

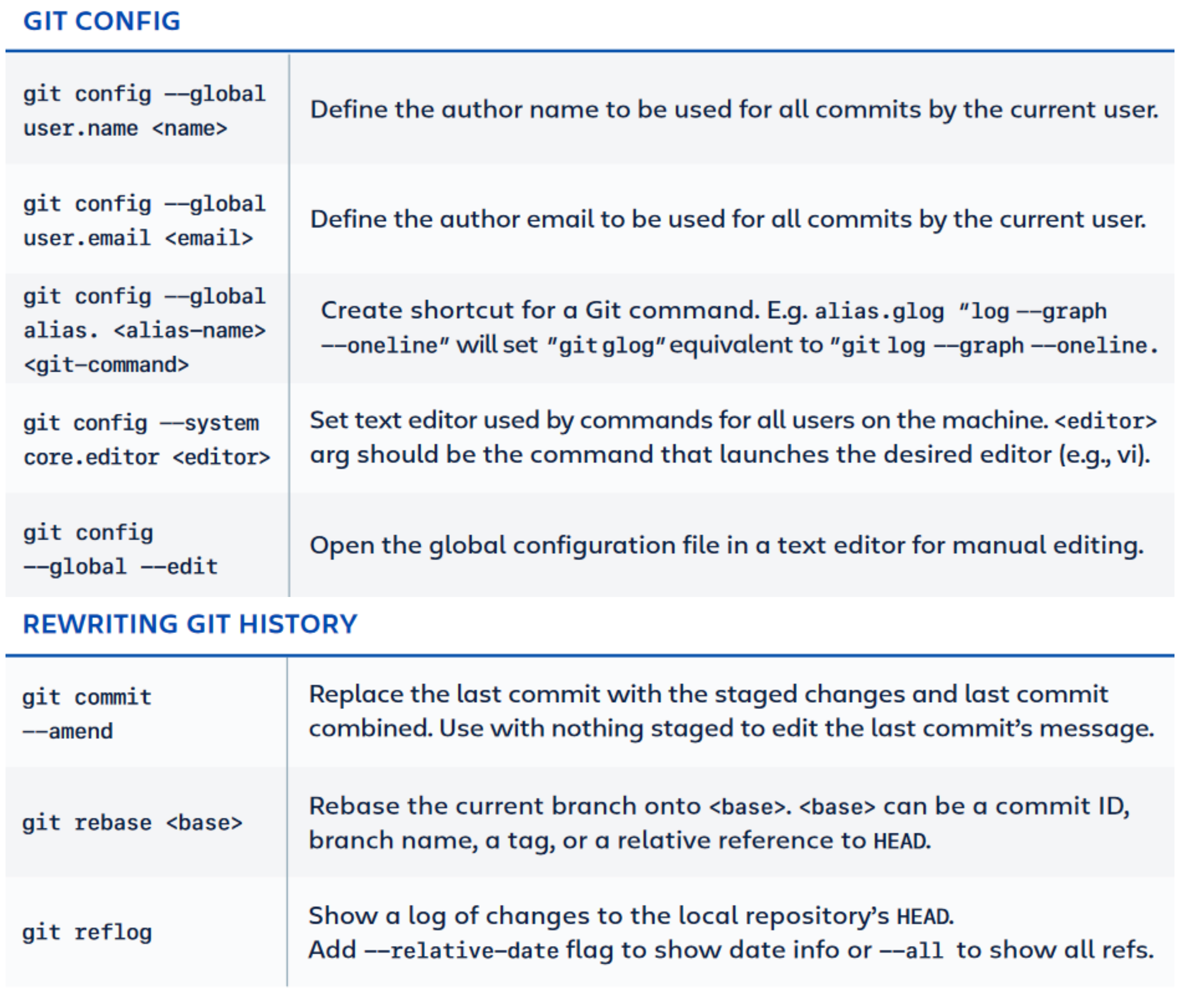
`git branch`	List all of the branches in your repo. Add a `<branch>` argument to create a new branch with the name `<branch>`.
`git checkout -b <branch>`	Create and check out a new branch named `<branch>`. Drop the `-b` flag to checkout an existing branch.
`git merge <branch>`	Merge `<branch>` into the current branch.

REMOTE REPOSITORIES

`git remote add <name> <url>`	Create a new connection to a remote repo. After adding a remote, you can use `<name>` as a shortcut for `<url>` in other commands.
`git fetch <remote> <branch>`	Fetches a specific `<branch>`, from the repo. Leave off `<branch>` to fetch all remote refs.
`git pull <remote>`	Fetch the specified remote's copy of current branch and immediately merge it into the local copy.
`git push <remote> <branch>`	Push the branch to `<remote>`, along with necessary commits and objects. Creates named branch in the remote repo if it doesn't exist.

https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet

Git – cheat sheet

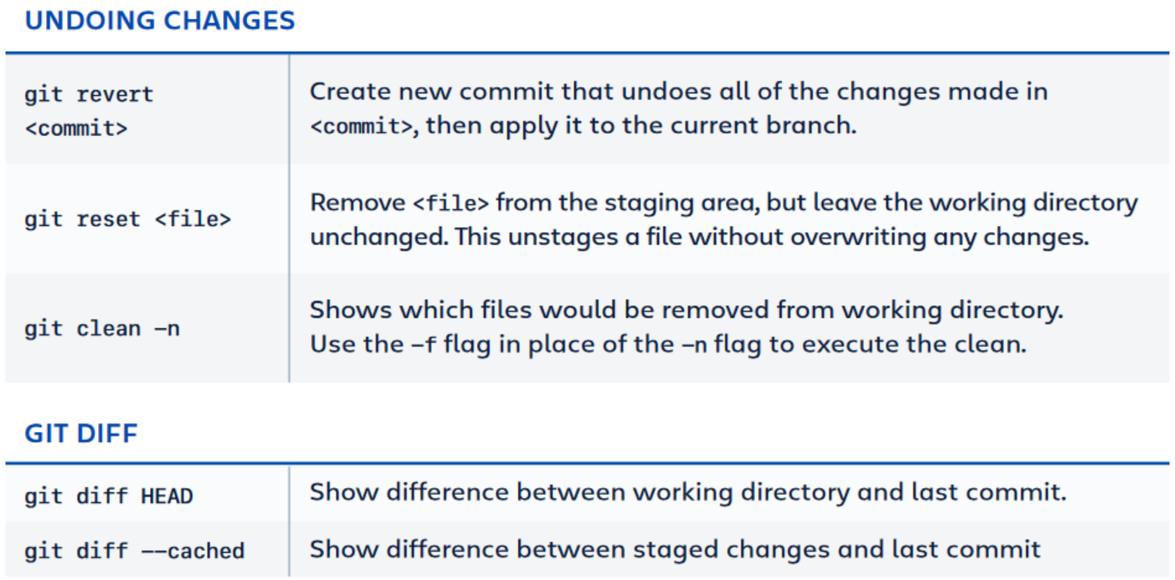


GIT CONFIG

`git config --global user.name <name>`	Define the author name to be used for all commits by the current user.
`git config --global user.email <email>`	Define the author email to be used for all commits by the current user.
`git config --global alias. <alias-name> <git-command>`	Create shortcut for a Git command. E.g. `alias.glog "log --graph --oneline"` will set `"git glog"` equivalent to `"git log --graph --oneline`.
`git config --system core.editor <editor>`	Set text editor used by commands for all users on the machine. `<editor>` arg should be the command that launches the desired editor (e.g., vi).
`git config --global --edit`	Open the global configuration file in a text editor for manual editing.

REWRITING GIT HISTORY

`git commit --amend`	Replace the last commit with the staged changes and last commit combined. Use with nothing staged to edit the last commit's message.
`git rebase <base>`	Rebase the current branch onto `<base>`. `<base>` can be a commit ID, branch name, a tag, or a relative reference to HEAD.
`git reflog`	Show a log of changes to the local repository's HEAD. Add `--relative-date` flag to show date info or `--all` to show all refs.



UNDOING CHANGES

`git revert <commit>`	Create new commit that undoes all of the changes made in `<commit>`, then apply it to the current branch.
`git reset <file>`	Remove `<file>` from the staging area, but leave the working directory unchanged. This unstages a file without overwriting any changes.
`git clean -n`	Shows which files would be removed from working directory. Use the `-f` flag in place of the `-n` flag to execute the clean.

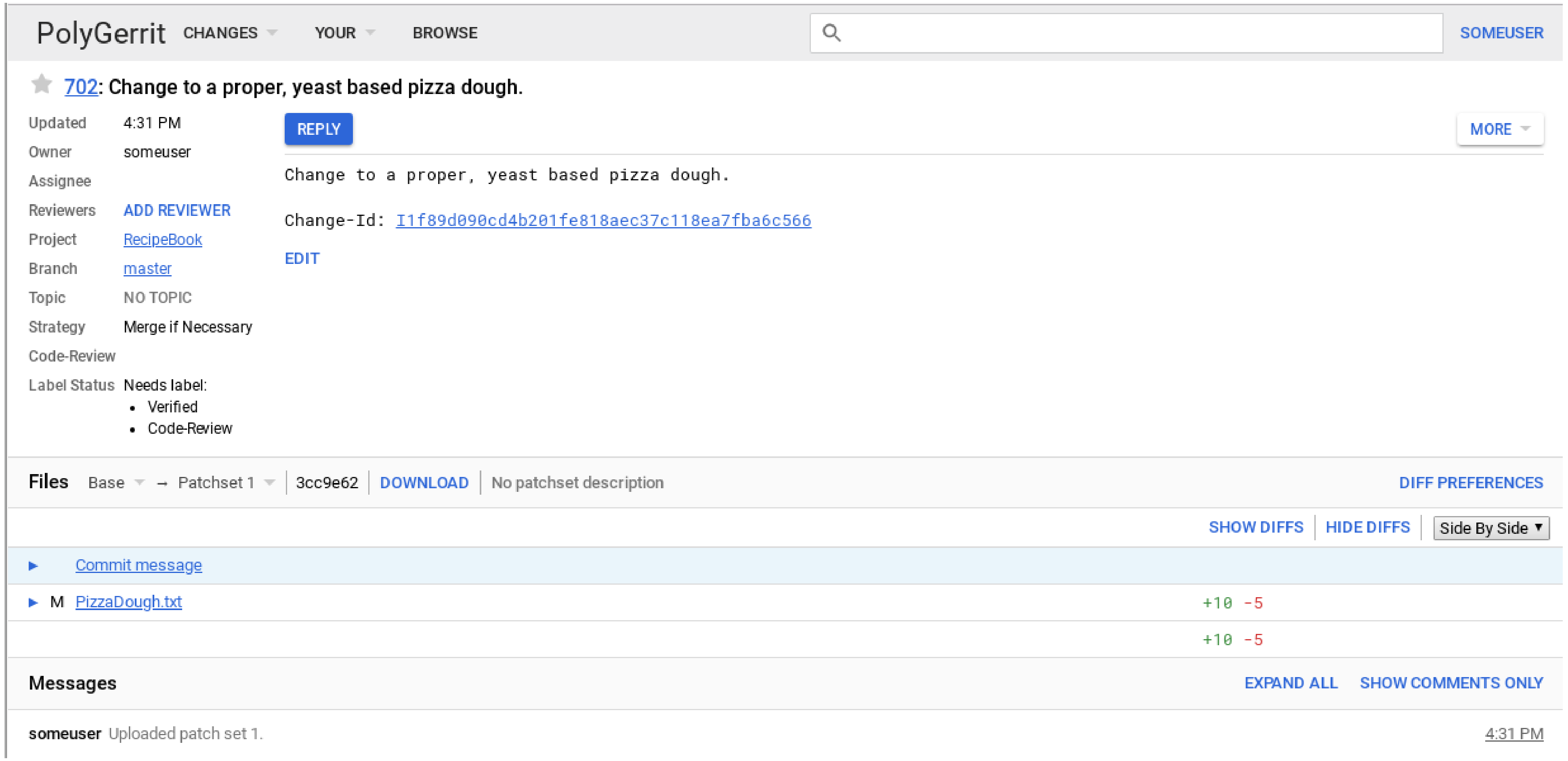
GIT DIFF

`git diff HEAD`	Show difference between working directory and last commit.
`git diff --cached`	Show difference between staged changes and last commit

https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet

Gerrit – request a review

- Once a commit happens to the **`refs/for/`**<*branchName*> of the remote repository, Gerrit creates a review.



PolyGerrit CHANGES YOUR BROWSE SOMEUSER

702: Change to a proper, yeast based pizza dough.

Updated	4:31 PM
Owner	someuser
Assignee	
Reviewers	ADD REVIEWER
Project	RecipeBook
Branch	master
Topic	NO TOPIC
Strategy	Merge if Necessary
Code-Review	
Label Status	Needs label: • Verified • Code-Review

REPLY MORE

```
Change to a proper, yeast based pizza dough.

Change-Id: I1f89d090cd4b201fe818aec37c118ea7fba6c566
```

EDIT

Files Base → Patchset 1 | 3cc9e62 | DOWNLOAD | No patchset description DIFF PREFERENCES

SHOW DIFFS | HIDE DIFFS | Side By Side

▶		Commit message	
▶	M	PizzaDough.txt	+10 -5
			+10 -5

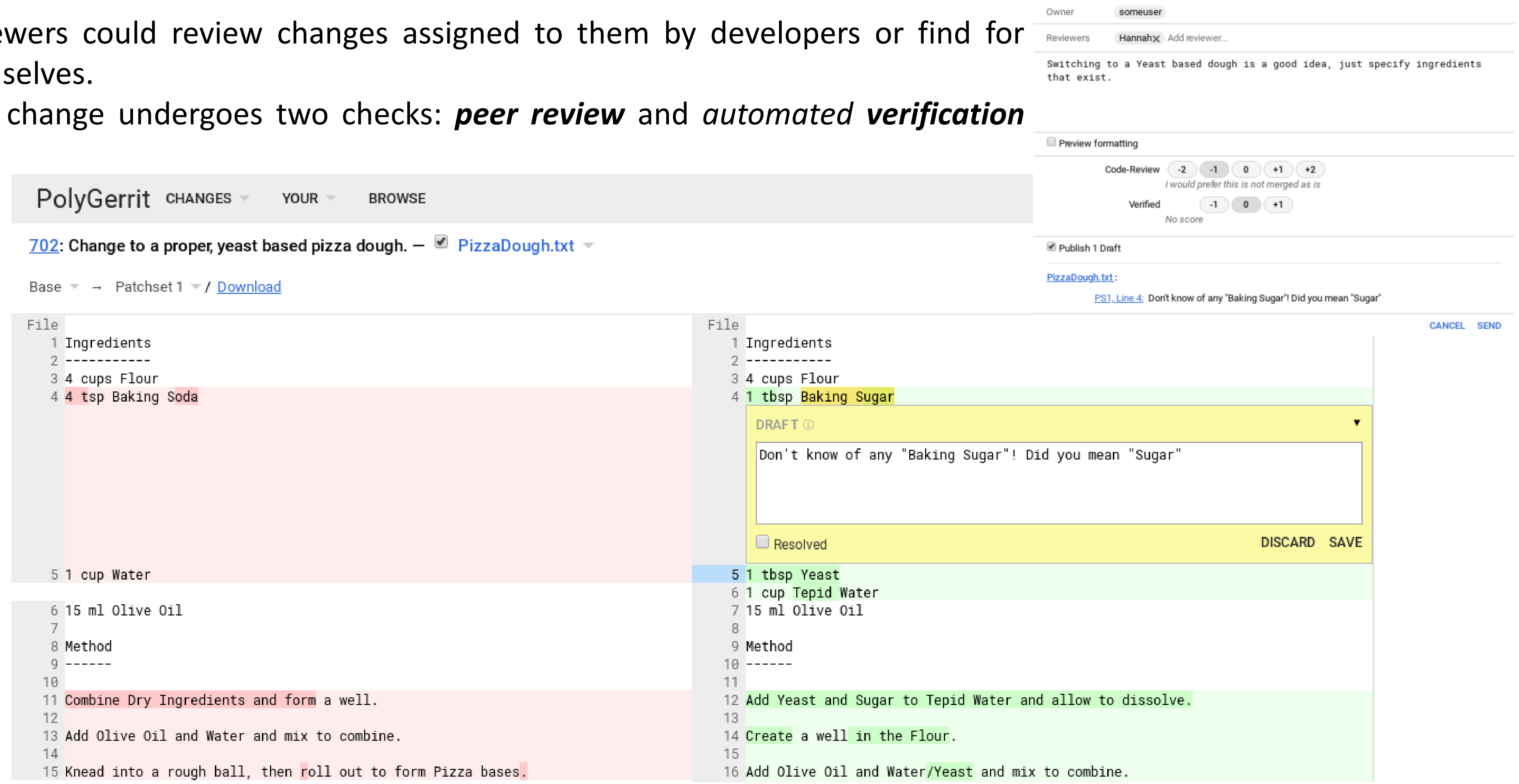
Messages EXPAND ALL SHOW COMMENTS ONLY

someuser Uploaded patch set 1. 4:31 PM

https://gerrit-review.googlesource.com/Documentation/intro-gerrit-walkthrough.html#_making_the_change

Gerrit – review a change

- Reviewers could review changes assigned to them by developers or find for themselves.
- Each change undergoes two checks: ***peer review*** and *automated* ***verification*** *step.*



702: Change to a proper, yeast based pizza dough. — PizzaDough.txt

```
Switching to a Yeast based dough is a good idea, just specify ingredients
that exist.
```

Code-Review: -1 — *I would prefer this is not merged as is*

PizzaDough.txt:

PS1, Line 4: Don't know of any "Baking Sugar"! Did you mean "Sugar"

```
File
 1 Ingredients
 2 -----------
 3 4 cups Flour
 4 4 tsp Baking Soda
 5 1 cup Water
 6 15 ml Olive Oil
 7
 8 Method
 9 ------
10
11 Combine Dry Ingredients and form a well.
12
13 Add Olive Oil and Water and mix to combine.
14
15 Knead into a rough ball, then roll out to form Pizza bases.
```

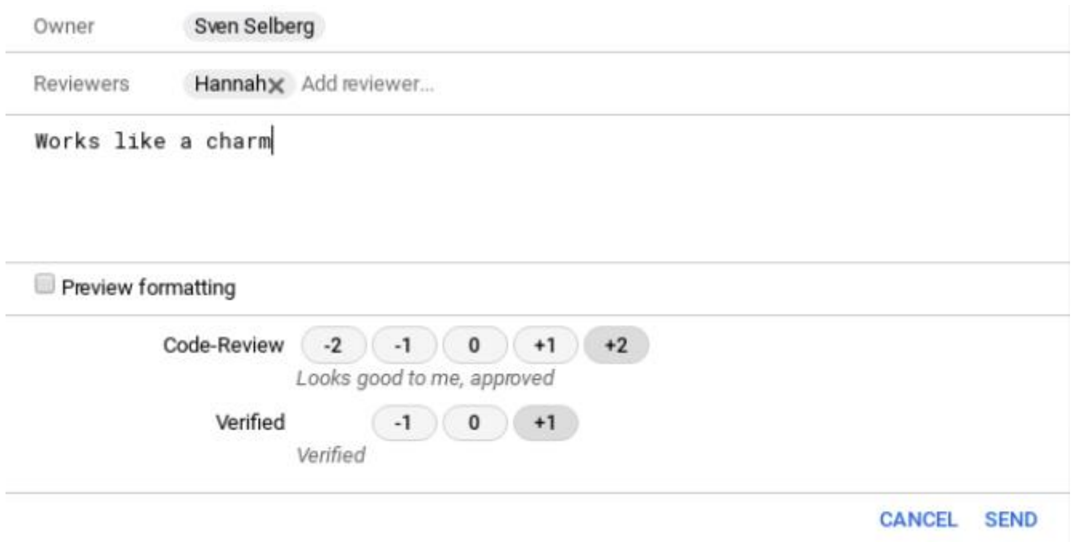
```
File
 1 Ingredients
 2 -----------
 3 4 cups Flour
 4 1 tbsp Baking Sugar
 5 1 tbsp Yeast
 6 1 cup Tepid Water
 7 15 ml Olive Oil
 8
 9 Method
10 ------
11
12 Add Yeast and Sugar to Tepid Water and allow to dissolve.
13
14 Create a well in the Flour.
15
16 Add Olive Oil and Water/Yeast and mix to combine.
```

```
Don't know of any "Baking Sugar"! Did you mean "Sugar"
```

https://gerrit-review.googlesource.com/Documentation/intro-gerrit-walkthrough.html#_creating_the_review

Gerrit – reworking & submitting

- If reviewers *reject* the changes, the developer shall:
  - Incorporate the comments
  - Checkout the commit
  - *Amend* the commit (*rebase* if necessary)
  - Push the commit to Gerrit
- Once the change are approved by the reviewer it needs to be *verified*.
- Verification is usually an automated step, reviewers with *verification permission* can perform manual verification if needed.



Owner Sven Selberg

Reviewers Hannah Add reviewer...

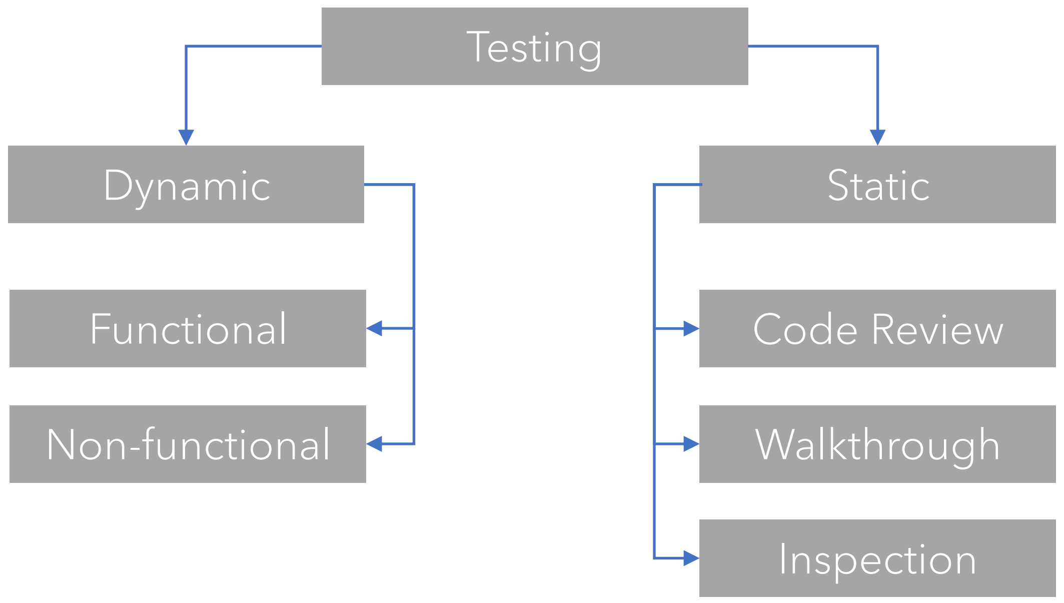
Works like a charm

https://gerrit-review.googlesource.com/Documentation/intro-gerrit-walkthrough.html#_creating_the_review

- The verification procedure is usually triggered automatically once a reviewer approves the change. There are *plug-ins* for Gerrit which triggers build automation tools like Jenkins.

Dynamic vs Static

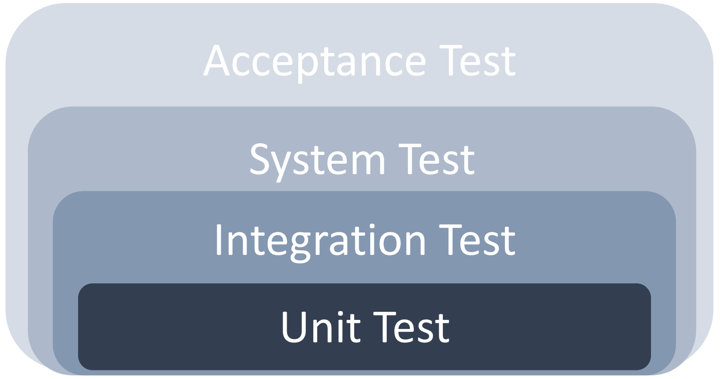
- Dynamic testing requires execution of the software.
- Static testing occurs on the source code.
- Static testing can only ***verify*** while dynamic testing can both ***verify*** & ***validate***:
    - Functional *verifies* the whether right product is made.
    - Non-functional *validates* if the product is the right one.



- Inspection covers a broad range of activities – code reviews and peer reviews are both inspections.
- Walkthroughs and inspections could target any artefacts not just source code.
- Verification is to figure whether the *correct* product is implemented, whereas validation's concerns is if the product is implemented *correctly*.

Testing Level

- Testing, conceptually, can occur on four levels:
  - ***Unit Testing***: testing the smallest working *units,*
  - ***Integration Testing***: testing the *interfaces* between *components,*
  - ***System Testing***: testing the whole *integrated* system, and
  - ***Acceptance Test***: testing the system for the stakeholders

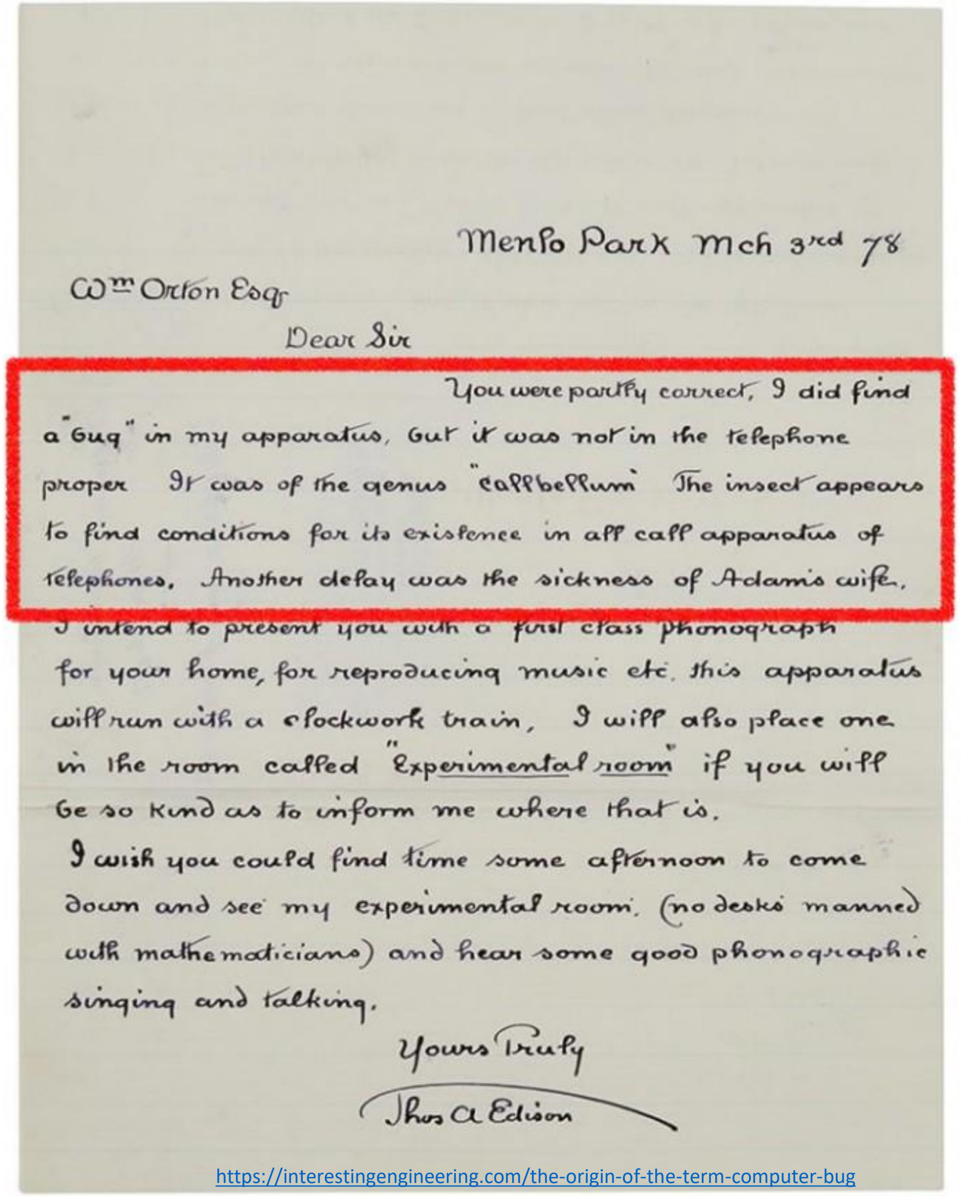


"Broadly speaking, there are at least three levels of testing: unit testing, integration testing, and system testing. However, a fourth level, acceptance testing, may be included by developers. This may be in the form of operational acceptance testing or be simple end-user (beta) testing, testing to ensure the software meets functional expectations. Based on the ISTQB Certified Test Foundation Level syllabus, test levels includes those four levels, and the fourth level is named acceptance testing. https://en.wikipedia.org/wiki/Software_testing#Testing_levels

- *Acceptance test* is usually a test performed to certify the system is satisfying the requirements -- this is a very common test which occurs upon delivery of a system.

Debugging

- Bugs are faults in the program which causes it to behave unexpectedly.
- Debugging is the procedure of resolving the bugs.
- *Debugger* is a tool used to *execute* program *step by step,* providing developer with a *real-time* inside look into software.
- Compiler can insert *special header* in the machine code for debuggers.
- Bugs which are not reproducible are very bad!
- **gdb** from GNU project is the main debugger used for C and C++.



Menlo Park Mch 3rd 78

Wm Orton Esqr

Dear Sir

You were partly correct, I did find a "bug" in my apparatus, but it was not in the telephone proper. It was of the genus "callbellum" The insect appears to find conditions for its existence in all call apparatus of telephones. Another delay was the sickness of Adams wife. I intend to present you with a first class phonograph for your home, for reproducing music etc. this apparatus will run with a clockwork train, I will also place one in the room called "Experimental room" if you will be so kind as to inform me where that is.

I wish you could find time some afternoon to come down and see my experimental room, (no desks manned with mathematicians) and hear some good phonographic singing and talking.

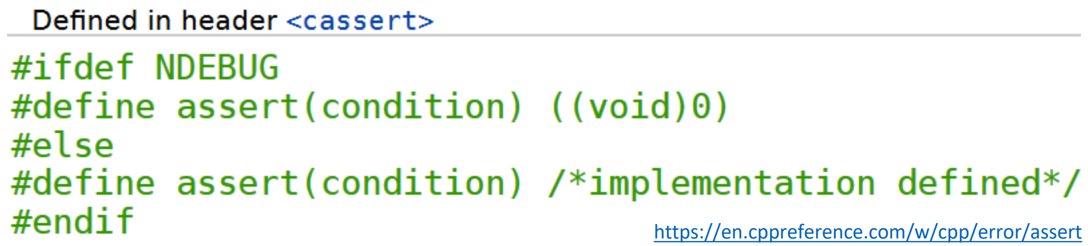
Yours Truly

Thos A Edison

https://interestingengineering.com/the-origin-of-the-term-computer-bug

assert

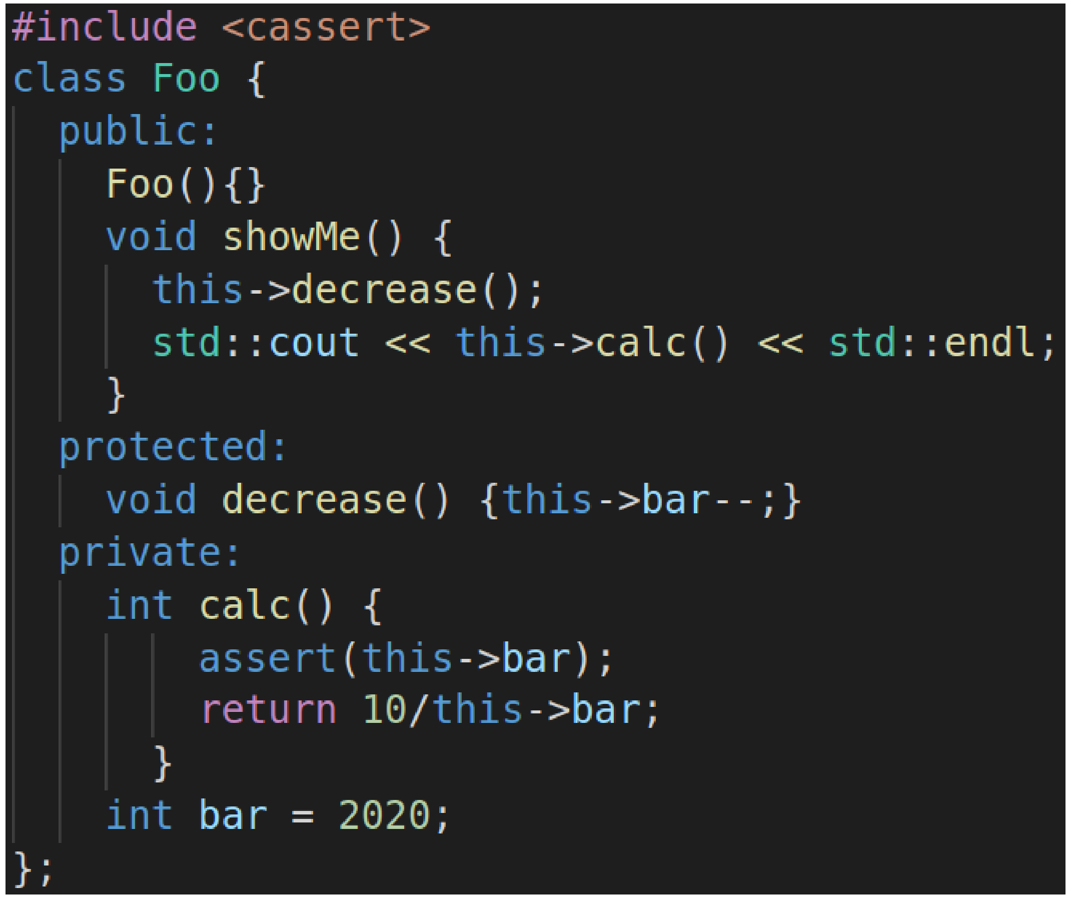
- Checks whether a condition holds or not (equals to 0).
- If the condition doesn't hold, information regarding where in the source code the *assertion* has failed is printed out and the program execution is *aborted.*
- Usually would go to *release builds* as the extra check hampers the performance.
- Gives developers the better possibility of bug tracing.



Defined in header <cassert>

```
#ifdef NDEBUG
#define assert(condition) ((void)0)
#else
#define assert(condition) /*implementation defined*/
#endif
```

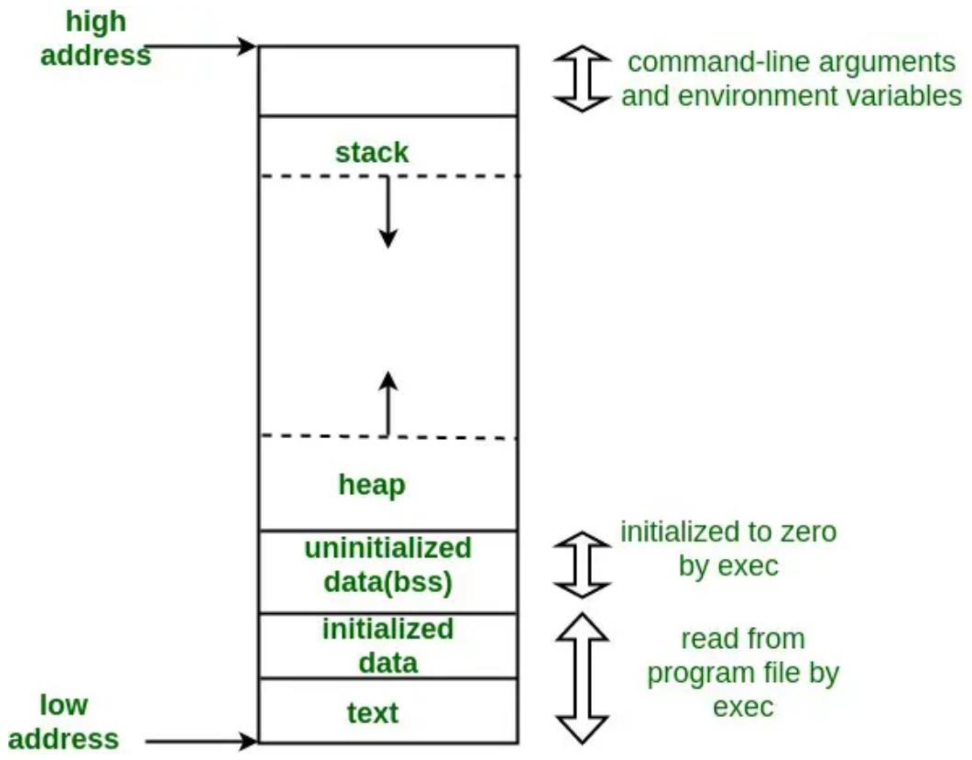
https://en.cppreference.com/w/cpp/error/assert



```cpp
#include <cassert>
class Foo {
  public:
    Foo(){}
    void showMe() {
      this->decrease();
      std::cout << this->calc() << std::endl;
    }
  protected:
    void decrease() {this->bar--;}
  private:
    int calc() {
        assert(this->bar);
        return 10/this->bar;
      }
    int bar = 2020;
};
```

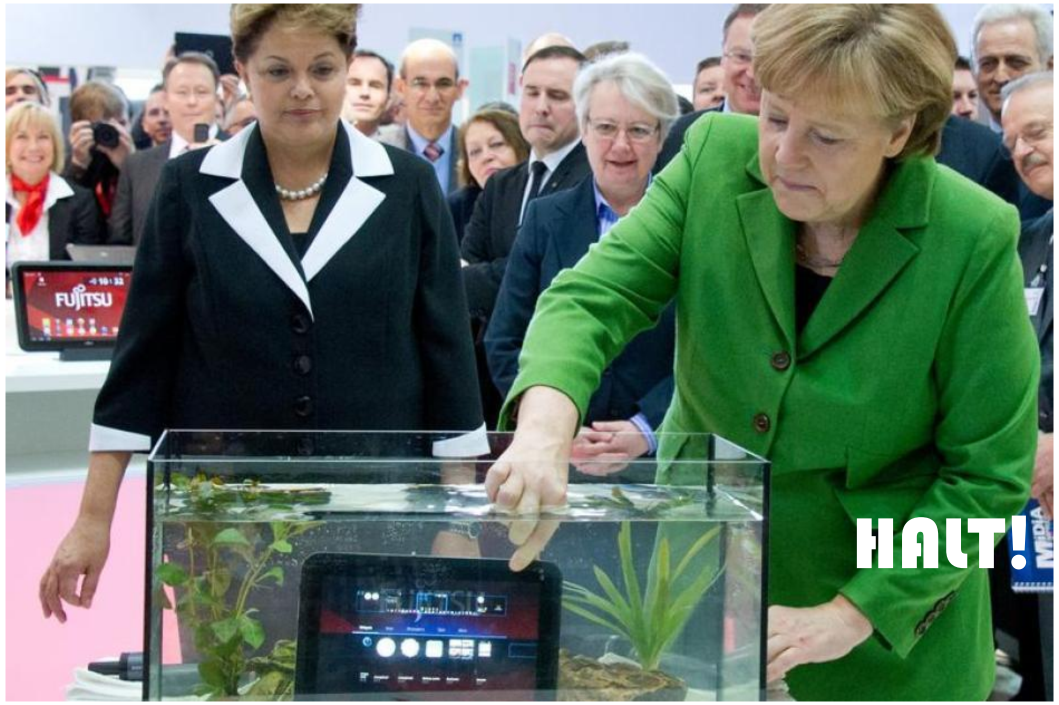
C++ Worst Bugs?

- C++ code can make memory faulty, even crashing the whole system.
- There are specific tools which look into memory allocation and de-allocation through dynamic and static testing.
- Users with malicious intentions will try to exploit memory bugs.
- Certain security techniques exist to ensure memory safety:
    - Stack canaries
    - Non-executable memory space
    - Address space layout randomization


high address
command-line arguments and environment variables
stack
heap
uninitialized data(bss)
initialized to zero by exec
initialized data
read from program file by exec
text
low address

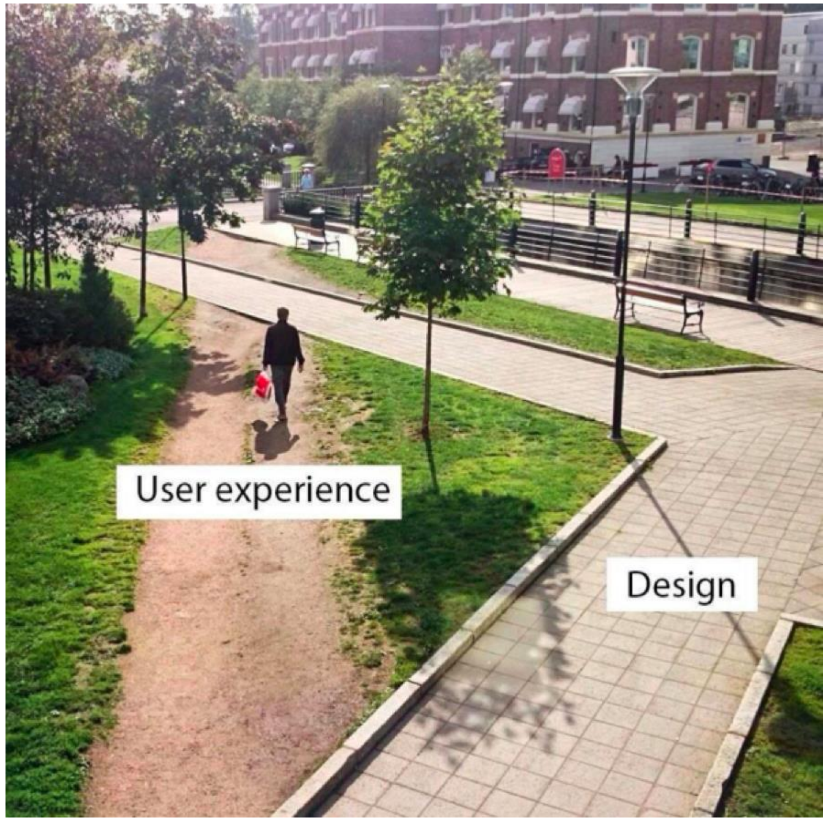
https://microcontrollerslab.com/difference-between-stack-and-heap/

BREAK!



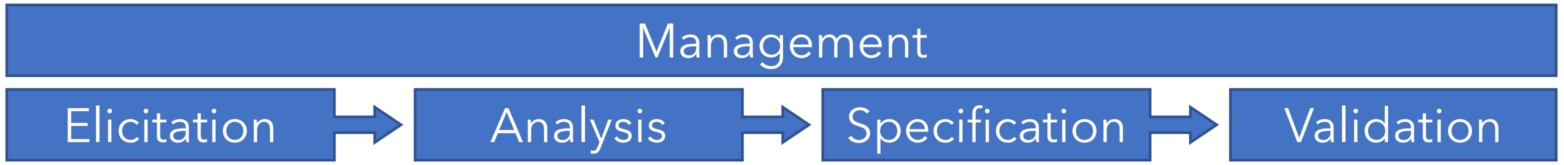
Software Requirements

- Requirement the is needs of *stakeholders* which the *software* supposed to satisfy.
- Requirements are both functional (capability) & non-functional (condition).
- Gaining knowledge about the problem from different sources through *requirement elicitation.* Requirements are the *refined* and *specified*.
- Through out the project, requirements are traced, prioritized, document, and might even be modified through a process called requirement management.
- At the end of the project (acceptance test) requirements are *validated & Verified.*

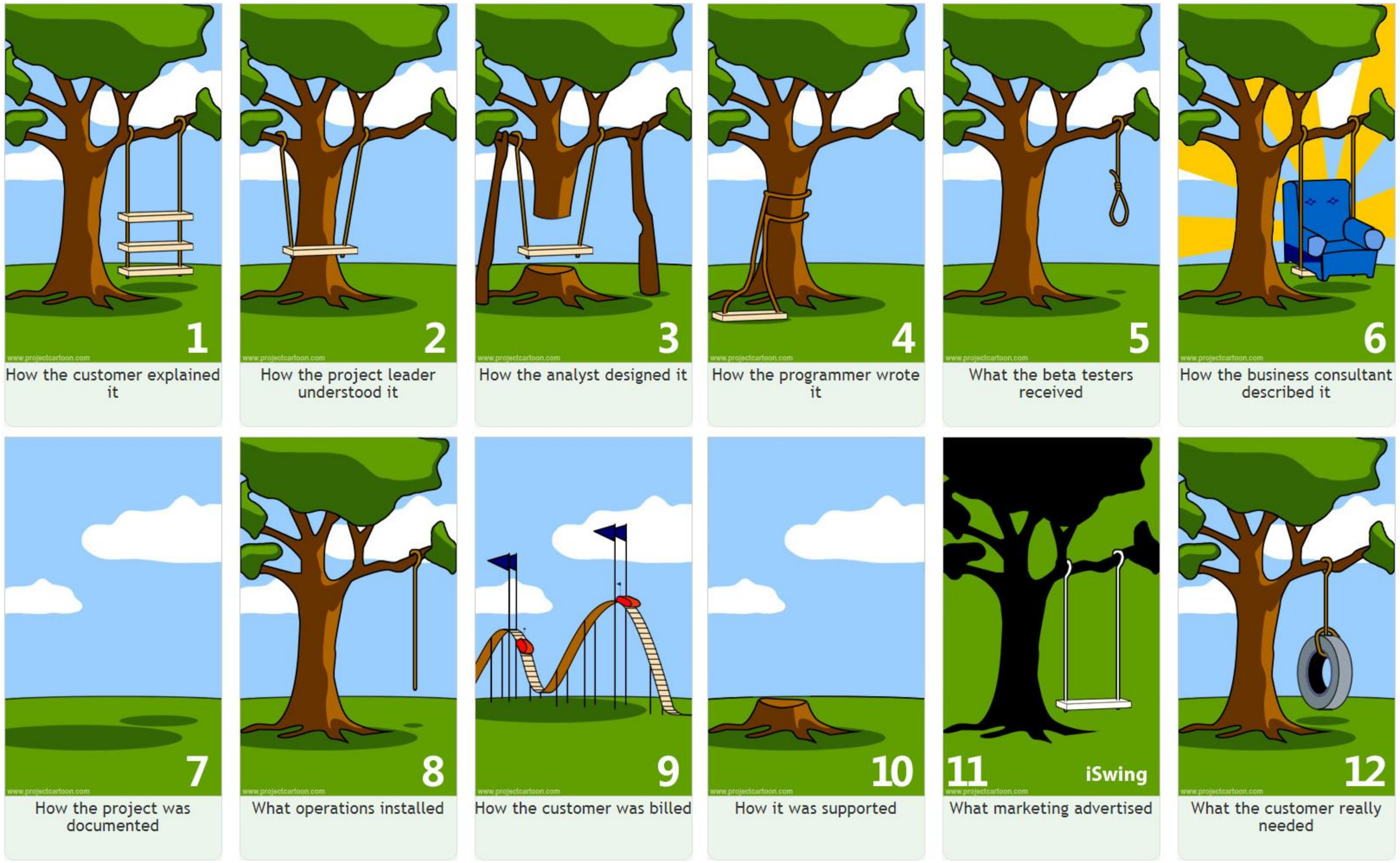


https://missunitwocents.tumblr.com/post/163749409715/desire-paths

- Requirements are the skeleton of the project and heavily affect architecture, design, and implementation of software. Massive requirement changes are costly, especially late in the development.



Software Requirements

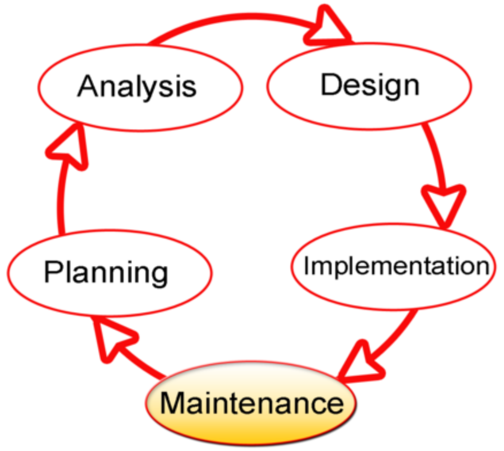

1
www.projectcartoon.com
How the customer explained it
2
How the project leader understood it
3
How the analyst designed it
4
How the programmer wrote it
5
What the beta testers received
6
How the business consultant described it
7
How the project was documented
8
What operations installed
9
How the customer was billed
10
How it was supported
11
iSwing
What marketing advertised
12
What the customer really needed

- Proper Documentation of the requirements ensure information integrity through out communications.

https://www.zentao.pm/share/trees-wing-project-management-cartoon-97.html

SDLC

- System Development Life Cycle is the complete process of an *information system.*
- There are different SDLC models & methodologies:
  - ***Sequential***: focuses on correct and complete planning as the guideline for the project (BDUF).
  - ***Iterative***: implement and enhance modules (limited scope) over many iterations.
  - ***Agile***: concentrates on *micro-processes* which welcome modifications through out the whole development cycle

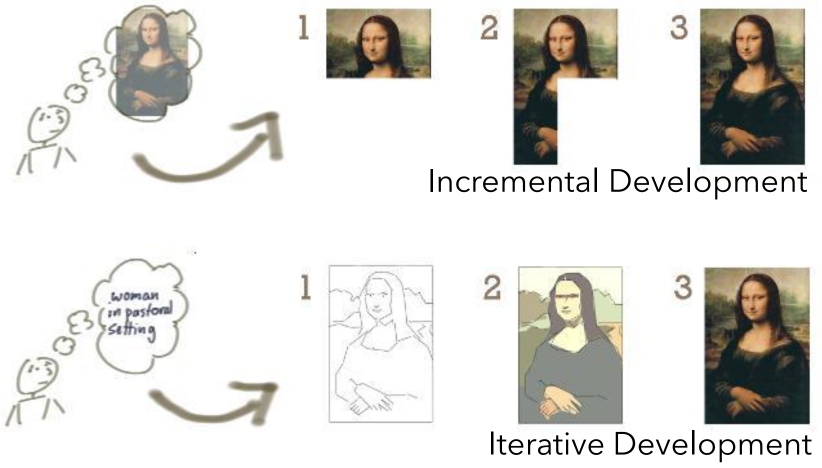


https://en.wikipedia.org/wiki/Systems_development_life_cycle#Design

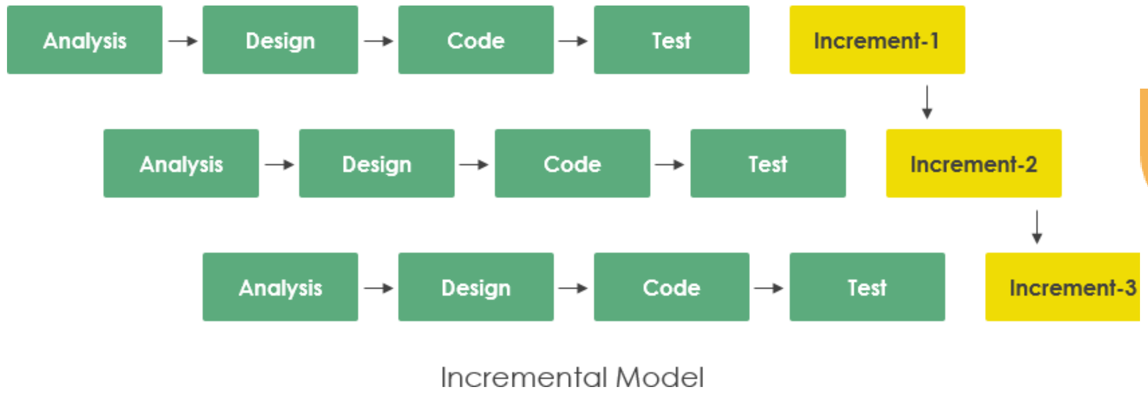
- Maintenance is a step in which the cycle could end.

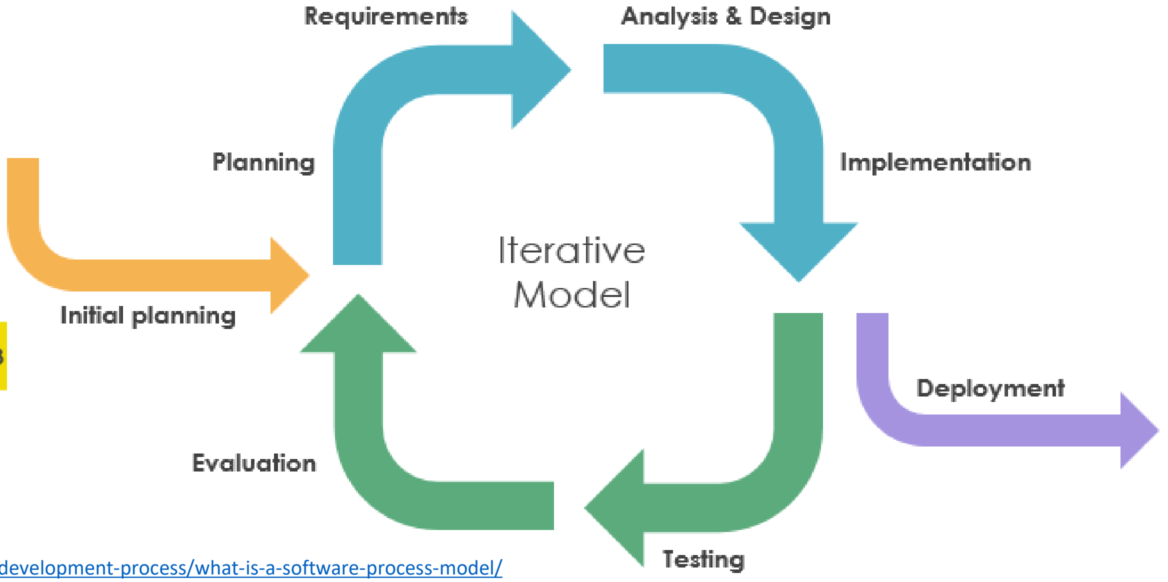
Iterative vs Incremental

- Both iterative and incremental model provide the development team with feedback from the previous increment or iteration.
- Iterative development is suitable for scenarios where the final solution is hard to be defined.
- It is common to use a *hybrid* of the two.



https://alisterbscott.com/2015/02/02/iterative-vs-incremental-software-development/





https://www.visual-paradigm.com/guide/software-development-process/what-is-a-software-process-model/