~~Method~~

Date:
P. No:

# Agile Variants

~~round-table offices~~

~~Extreme programming~~ :-

- get a task ; get it done -> one of the best
-

Rapid feedback ->

Assume simplicity -> Assume it's insanely easy

Incremental change, embracing change ->
Don't work against the change, embrace it.

Scrum :-

b/w ~~extreme~~ EP & Waterfall ->
-> Schwaber & Beedle

Prioritize the product backlog

Sprint backlog is a part of it.

Sprint -> 2 weeks to 30 days (shorter is better)

Don't change the req during 24 hr meet.

Date:
P. No:

The team...

Scrum master → Removes of roadblocks (not controls but planner)
Scrum team → Diversely skilled
Product owner → Can communicate with the custo

The sprint...

Time-boxed activity →
No change within the sprint →

Product backlog...
subdivided in releases

Sprint backlog...
Taken from product backlog

Scrum meetings...

Sprint planning →
Daily scrum → • Sprint review • Sprint retrospe
Stand up meeting → May not inc. product owner

Benefits: Toyota effect → Become market leader & stop

Date:
P. No:

# Open Source Software Dev.

Q The Halloween Docs.

* Usenet → first social network

* AT&T → Intellectual Property Rights

The first Era (1960s to 1980s) free

The second era → GPL (modifications must be free, even upon small usage of open source GPL)

The Third era → Cooperation b/w open source & proprietary

'Debian Guidelines'
MIT License, Berkely (BSD).

GNU
GPL → Must redistribute freely

Problems → Monetization: →

Redhat (LINUX distributors) → Support and new features was paycalled (but first version free)

MySQL tried the same (didn't work out)

GNU's NOT UNIX

Date:
P. No:

Sponsorship:-
Didn't work

SaaS → Software as a Service (e.g. google doc
(aaS) → Saved open-source

Redistributable (Importance)

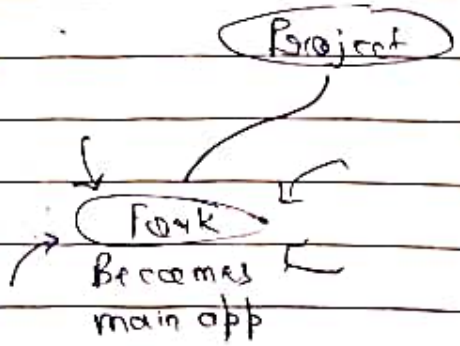
Open-source ≒ Agile

Open source route → IBM

NIH Syndrome

How does it die →

① Forking




e.g. Netscape Navigator → Chrome

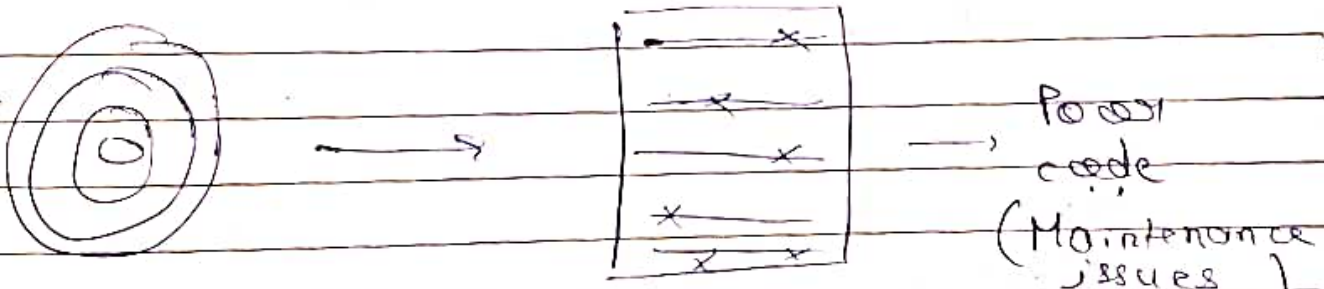
Date:
P. No:

2. Leader : Not interested / unlike orgs.

3. Lack of updates: Outlives its utility

e.g. → ERP on the cloud (pay when you need)
In-house ERP → you can't.
ORACLE offers ERP on cloud.

Problems with incremental development:-




Structure → needs high quality

Software → Must be modular

Modular → high cohesion
low coupling


→ similar & close together
→ minimal coupling

Change in one module should require

Date:
P. No:

With more changes, coupling ↑, cohesion ↓.
(or increments)

# Refactoring

Set of rules to gain simplification, quality & modula

Find red flags and do what works out the best.

↓

Bad smell.

Navigate through the structure, find b.s. and rectify
→ locally.

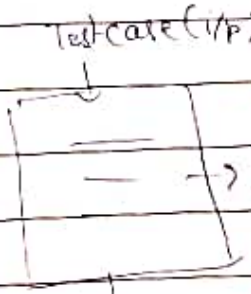
Date:
P. No:



# Software Testing

* Software Testing is a negative activity.
Try hard to find errors.
No errors → bad testing.
Testing is done by third-party (other than devs).
Usually blind.
Review & inspection → Informal phase (usually in agile)
Commercial env. req. more formal methods (test cases)

Test cases → Representatives for i/p domains are tel cases
(can't test for entire input).




O/p (If o/p is correct, then software works properly for that PARTICULAR case)

Check for what's correct → ORACLE

Date:
P. No:

ORACLE → Can be a client
, or a complex mechanism

1. Unit Testing
2. Component (Integration) Testing } part of dev. process
3. System Testing

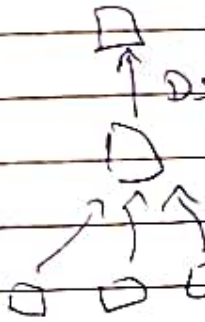
Unit Testing → Done by devs during development
A unit can be single f^n or class.

Driver
Stubs → Unit Testing

Driver → Simulates the f^n of main function
and calls the module (class or f^n)

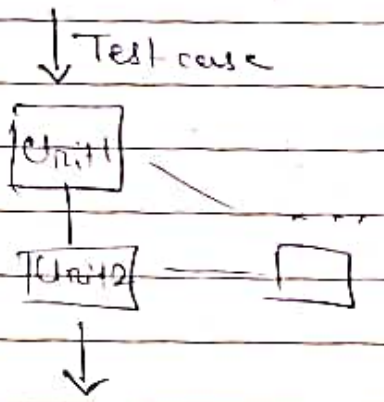
Stubs → F^n that calls other f^n. (simulates)
Checks if a f^n is calling others correctly

Compare results with ORACLE.



Date:
P. No:

Component (Integration) Testing → Connect incremen-
-tally.




: Also done by devs.

Debugging → While testing discovers errors
, debugging is for rectifying them.

System Testing → Goes beyond the code. Integrate
with the final system and test.
Also done by devs.

Software is delivered after these three tests
and goes for formal testing by 3rd-party.

SLA → Service Level Agreement.

e.g. agreed accuracy → 95%
recall accuracy → 92%

Date:
P. No:

Project — Create a doc. (Manual Testing)

| | TC1 | TC2 | TC3 |
|---|---|---|---|
| $R_1$ | | | |
| $R_2$ | | | |
| $R_3$ | | | |
| . | | | |
| . | | | |
| . | | | |
| . | | | |
| . | | | |

For every req.
input for req.
Test cases for each req.
Compare with ORACLE

ORACLE → could be person or client.

→ Must match with SRS

A good test case is the one that finds the errors

→ Testing may or may not tell precisely where the errors are, albeit their mere presence

→ Fraction of test-cases say little about quality. Too high accuracy may be due to poor test cases & vice versa.

Formal testing → Third-party

Generation of test cases →

Too crucial that it's not automated yet.

Date:
P. No:

Methods:-

→ Random

→ Category partition (e.g. addition → $\mathbb{R}^-$, $\mathbb{R}^+$, Q, etc.)
every categ. has few representative test cases.

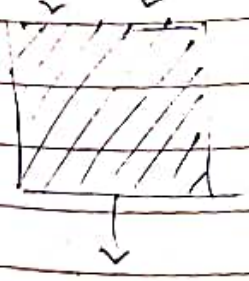
→ Boundary value analysis ($D \in [0, 99]$ → pick 0 or 99) (and)

→ Mutation testing (introduce noise / faulty code fragments and see evaluate if test cases by their ability to detect noise. Keep refining test cases.)

Pesticide paradox :-

Even the best test cases need to be upgraded refining many from they stop unearthing bugs after several

Black-box vs white-box (glass) testing :-



→ White-box testing → deliberately create test-cases that covers few or several or all components.

→ Difficult

→ Used in critical software.

Date:
P. No:

WB Testing →

(i) Statement coverage : Covers every statement

(ii) Control-coverage : Loops, conditionals & branches. Check every part of control statement. (possibility)

eg: if (x > 10 && x < 100)
check for everything 10-100

Statement coverage

WB (Design test cases around the internal struc. Resource extensive).

## Acceptance Testing

Third-party confirms →

- requirements of client
- working properly.

→ Divided into: alpha & beta testing

alpha-testing → At the site of dev. by 3rd party, customers & with min. involvement of devs. Independent of prev. test. Decision by 3rd party (black or white box, etc.)

↓

needs dev

Date:
P. No:

beta-testing → Open to selected or entire customer base.

gmail - 2009 → could only join through invite ⇔ could invite atmost 3 ; report

## Regression Testing

Test the app for every change (e.g. new feature). Follow the process as usual, equally thorough.

Load & Stress Testing → Test cases check if software can handle the load (no. of users, amount of data, etc.) in case of load testing
→ Stress testing goes beyond the expected limit, finds Breaking point.

When do you stop testing?

→ Stop it when you've had enough, in terms of time / satisfaction / ~~money~~ ei budget.

→ Or when no. of bugs become min.

→ Can never be exhaustive.

Date:
P. No:

Final Assessment →

2 min. → Motivation (Include changes in scope.

6 min → Demo

2 min → Contribution of each person

## Maintenance

→ Lasts throughout the software life-cycle.

Different cost : (1) Generic s/w
(2) Customized s/w
(3) Brownfield s/w

(1). Generic → Developers are responsible for maintenance. Discover bugs based on reports, identify general problem, prioritize issue & send patches (updates). (Perfective maintenance)

(2). Customized → Often by third-party, with some input from devs. Reactionary & unplanned, unlike scheduled maint. in generic s/w (ie corrective maintenance).

(3). Brownfield → For legacy systems. Usually involves adapting to the existing system. (adaptive maintenance).

Date:
P. No:

4-types of maintenance →

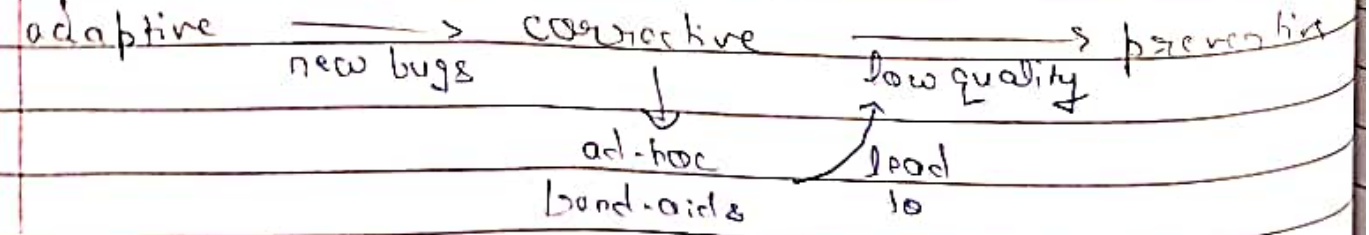
- adaptive
- corrective
- preventive
- perfective

→ corrective → identify, react and fix.

→ perfective → enhance, add new features.
Usually continuous, maybe competitive

Q: → adaptive → change when existing base changes, such as windows update or a new software brought as feature to existing system.

→ preventive → keeps improving the structure, not reactionary to errors, but anticipates future degradation in quality.



Date:
P. No:

Challenges in maintenance :-

→ structural degradation

→ quality of team ↓

→ cost and time (unanticipated cost)

- These challenges lead to 'maintenance' vs 'replacement' (throw away the old software, get new one).
- The structure becomes unacceptably bad, end it.
- Competitive env., new technology.