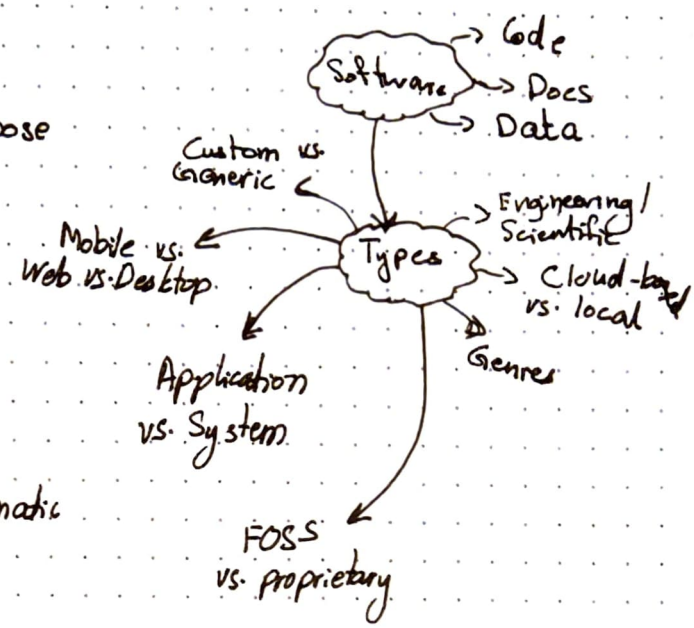
SQE L01 21-08-2024.

"Garbage in, garbage out."

Quality → fitness for purpose




Engineering: finding efficient solutions ~~about~~ for real-life problems in a systematic and efficient manner.

Problems compound (snowball effect, domino effect).

---

SQE L02 23-08-2024

ENIAC → Softwares → First CS dept. → Software Engineering
1940s late 1940s # in US 1962

THRAC-25
mid-late 1980s.

Denver, Colorado Airport Catastrophe

Washington prison malfunction: over 3000 prisoners released earlier

Toyota recalled 600,000 'Prius' cars in 2016

Nissan recalled ~3.5m

Flex

WHY SQE is a separate field?

- nature (virtual/logical vs. physical)
- size
- team setting
- formal ideation
- interfaces
- personnel turnover

Algorithmic Morality

SQE L03 28-08-2024

Software Quality Engineering → Assurance

Great Bookstores
Regal Chowk, Mall Rd.
- Multi-line books
- Allied Books (Nagq~~Book Market~~)
- Pak Book Corporation (Queen's Rd.)
- Vanguard Books (Montgomery Rd.)

Quality:
fitness for purpose
- conformance to explicit and implicit technical and managerial requirements

Perception of Quality ≠ Quality

Functional → services offered
Non-functional → constraints
budget (cost)
schedule (time)

Software Quality Problems:

Software Supplier
- Error: 'to err is human' incorrect action or inaction.
- Fault: a state of the software. not all errors become faults. → Bug/Defect

Software Customer
- Failure: an event. not all faults become failures.
- Incident: detected ~~incidents~~ failure not all failures get noticed.

Software Quality Assurance: the set of systematic, planned activities required to provide adequate confidence in that the software produced through the software development process conforms to the explicit and implicit, technical and managerial requirements.
→ umbrella activity, process oriented

→ subset of SQA, product oriented
Software Quality Control:

"A defect is a delivered fault."
- Beesman not for us.

"An accident is a severe failure"

---

SQE L04 30-08-2024

Capers Jones → Software Project Management

" Short-cutting a day of QA activity early in the project is likely to add 3 to 10 days of unnecessary activity downstream."
- Capers Jones

- The most important deliverable of Reqs. Engineering:


Quality Factors → dimensions, attributes, content groups, SRS headings

-ilities

(Classic, 1977)

McCall's Quality Factor Model → An abstraction of the real thing

total 11 Quality Factors

| day-to-day operations | change | outside-world interaction |
|---|---|---|
| product operation (5) | product revision (3) | product transition (3) |
| - Correctness | - Maintainability | - Portability |
| - Reliability | - Flexibility | - Reusability |
| - Integrity | - Testability | - Interoperability |
| - Efficiency | | |
| - Usability | | |

Correctness: functional requirements, output-mission, output accuracy, output completeness, output up-to-date-ness, availability (response time), standards

Reliability: failure rates, probabilities (critical systems)

Integrity: security (Authentication & Authorization)

Efficiency: optimal use of hardware resources (CPU, RAM, storage, communication bandwidth, battery life)

Usability: (ease-of-use)
- operations
- training

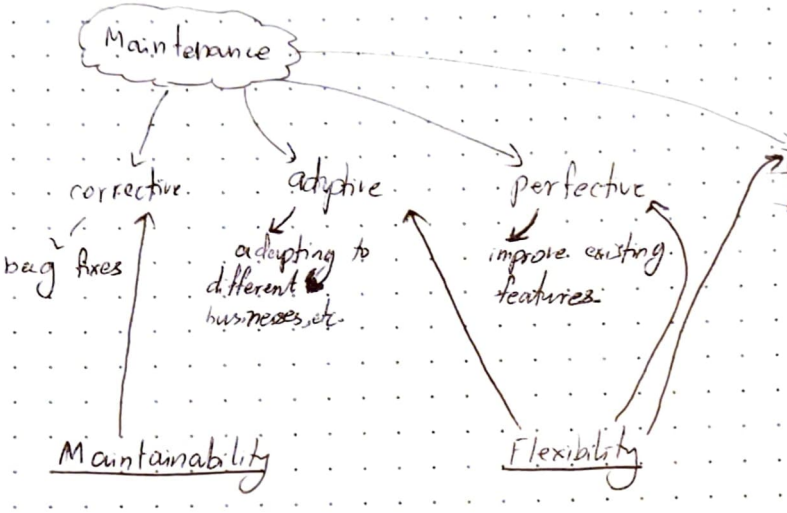
Question 03 (b)

SQE LO5 04-09-2024

BIOS does POST
↓
Power-On Self Test

McCall's QF Model

Product Revisions: change and maintenance

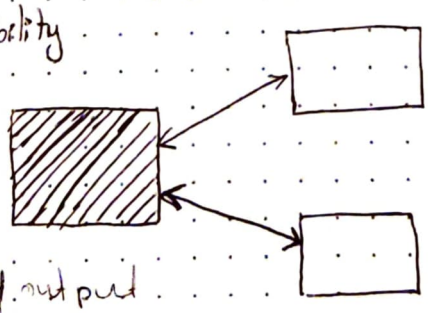



Software Supplier — Software Client/Consumer
$SRS_{SS}$ $SRS_{SC}$
Reusability
certain factors unique to each SRS version.

Testability
- easy to test
- intermediate results or logs
- allow operators and maintainers to test and diagnose the software.
- both comments and readability are important
- comments are not an excuse for bad programming practices.

Product Transition:
- Portability: cross-platform support.
- Reusability
  - for
  - with
    - improve documentation
    - increase cohesion
    - decrease coupling


- Interoperability



- standard output formats

Alternative QF Models.
- Evans & Marciniak (1987)
- Deutsch & Willis (1988)
  - Verifiability → mathematical verifications
  - Expandability ≈ Flexibility
  - Safety
  - Survivability ≈ Reliability
  - Manageability → administrative tools for change management.

human safety
↳ removing unsafe condition

SQE LOG, 06-09-2024

Project Managers, UI/UX, programmers, testers, maintainers, dev ops, etc.
↳ many people can make mistakes

## Components of SQA system

SQA applies to maintenance as well

Component Categories

Applicable to projects at hand
- Pre-project
- Life-cycle
- Infrastructure
- Management

Applicable organization-wide
- Standards
- Human

### Pre-Project Components

-> Contract Review
- RFP → Request for Proposal
- Draft → Draft Review
- SC evaluates submissions
  - ↳ one gets a contract.
  - ↳ contract draft → review

-> development & quality plans
- -> cost, resource, estimations
- -> WBS, Risk plans

SS ——— SC
↳ contract

### Life-cycle components

-> Reviews
- team-based meetings
- variable degree of formality
- types: formal technical reviews (most formal), Inspections, Walk Throughs & (least formal)
  - ↳ FTR, Design Reviews, FDR
  - ↳ done by senior people

Problem Analysis Phase — Solution Phase

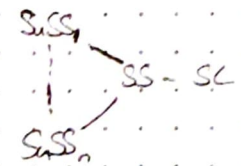
Inc RE A D J I D/D O-M O/R

Part 1 | Part 2

Peer Reviews

-> Testing
- regression, smoke, automated, manual, unit, integration, other tests.
  - black box, white box.

-> Expert Opinions
- outside reviews, etc
- Third Party




COTS / ROTS, GOTS
Commercial off-the-shelf components.

Question 03 (b)

SQE 106 (Continued)

Infrastructure Components

→ Procedures & Work Instructions
→ Supporting Quality Devices
- Templates (SRS, project plan, risk plan, etc.)
-

→ Training & Certifications
- Training improves

PMI → PMP
PMI → ACP
Scrum Master

→ Software Configuration Management
- Version Control
- Git, subversion, etc.

→ Analysis of Past-Projects
- what went wrong
- where most mistakes occur.

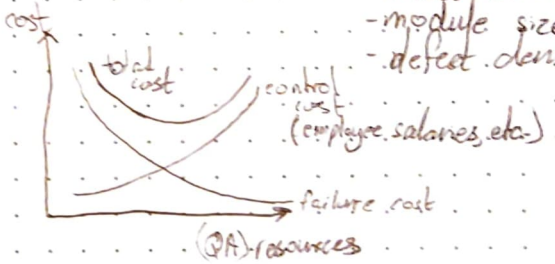
Management Components

→ Progress Control
Managers do planning, monitoring, controlling

→ Quality Metrics
- defect density
- productivity
- defect severity → weighted defects
↳ module size

M1 | M2
100 500
Quality depends on:
- defect severity
- module size
- defect density

→ Cost
- Minimize project cost
failure cost
control cost




Standards Components
→ Quality Management (What)
- CMMI capability Maturity Model Integrated
↳ developed by CMU
- ISO 9000 3
→ Project Process (How)
- IEEE 1012
- ISO/IEEE/IEC 12207

Human Components
→ Employees, team members, project managers.

CEO, CTO, CFO

Top
Department Heads
Middle
Tiers
Bottom

Factors Affecting QA system

- organization size $\propto$ QA effort required
- project complexity $\propto$ QA effort required
- experience (relevant to the project domain) $\propto \frac{1}{\text{QA effort required}}$
- 3rd party engagement $\propto$ QA effort required

- acquaintance of team members with each other $\propto \frac{1}{\text{QA effort required}}$
- extent of reuse (with-of reuse) $\propto \frac{1}{\text{QA effort required}}$
- qualification $\propto \frac{1}{\text{QA effort required}}$

Question 03(b)

SQE LO7+08 11-09-2024
13-09-2024

BRING CALCULATOR USP (Unique Selling Point)

# INTEGRATION OF SQA ACTIVITIES IN SDLC

Usability of drop-down vs. textbox
Recognition vs. Recall

Process Control vs. Quality Management: What vs. How, not product vs. process oriented

## Software Processes

Process: a set of activities approached in a specific order.
↳ approach, roadmap,
ProcessModel: an abstraction of the process.

Spiral ∴ a meta-process

change-friendliness

Conventional, Rigorous, Heavy-weight

Light-weight, lean

Processes

Traditional

I Know It When I See It Syndrome

Agile

Evolutionary

Non-Evolutionary

User Acceptance Test (UAT)

Team intends OOP

XP

project velocity

Scrum

Release

Planning Game

CRC cards

User stories

no. of user stories implemented in the current release

Committment
what user stories by what deadline?

Pair programming

Customer Colocation

refactoring

KISS
Minimalist Design

Throw-away prototypes

Build-upon prototypes

IKIWISI

Prototyping

Planning Problem: when will it end

Spiral

Prototyping

Risk-driven process

Heavy-weight

Big Bang Baching

Waterfall (Classical)

Incremental (Pure)

no. of features.

RAD

↳ Rapid Time, ~60-90 days

design prototypes called spike solutions

Large Projects

or medium

Small Projects

Stable Reqs.

Unstable Reqs.

Change welcome

only in hybrid versions

- large projects,
- stable reqs.
- inacceptable when:
  - unaware customer
  - volatile reqs.
  - feedback needed

Naturally Modularizable Problem

High Manpower

High Expertise

Rapid Delivery Required

- Kent Beck's book on XP.
- Refactoring by Martin Fowler

User Stories

As a who, I want what so that why

- Priorities (SC)
- Cost ($)
- value

February 2001, 17 people signed the agile manifesto in Utah.
↳ Kent Beck, Robert C. Martin

→ later extended to other fields like manufacturing, design, and Project management.

## Agile Manifesto

| | | | |
|---|---|---|---|
| 1- | Individuals & Interactions | vs. | Processes & tools |
| 2- | Working software | vs. | Comprehensive Documentation |
| 3- | Customer Communication | vs. | Contract Negotiation |
| 4- | Responding to change | vs. | Following a plan |

- **Scrum**: Managerial-heavy, as opposed to technical-heavy as in XP.

Iterations => Sprints
- → Time-boxed
- → Requirements do not change during a sprint
- → Backlog (outstanding undone requirements, prioritized)
  - → Product Backlog
  - → Sprint Backlog
- → In care of product-owner
- → Release (Demo)
- → Scrum Poker (for estimation and planning).

Daily Scrum Meetings
- → Led by the scrum master
- → Done standing up in chairs
- Three Questions:
  - What's the progress since the last meeting?
  - What will you do till the next scrum?
  - Are there any problems?

→ Scrum Retrospective

Glorified Hackers
Traditionalists vs. Agilists
Stuck in the mud

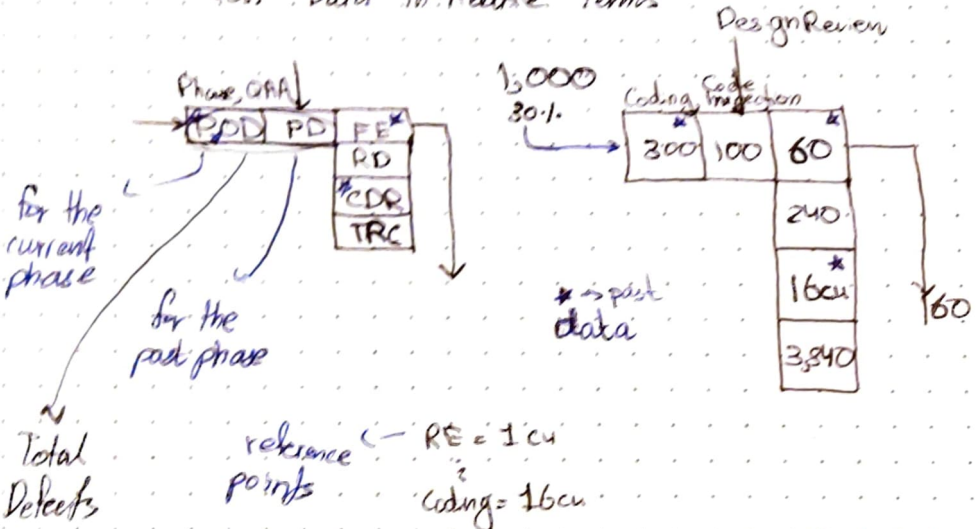
Barry Boehm → Balancing Agility and Discipline

SWE 203 (continued) 13-09-2024

Dimensions of SQA

## Defect Removal Effectiveness & Cost Model

- QA activities are filters, not 100% accurate, but still cost something.
  - Defect
  - Filtering Effectiveness
  - Cost Data in Relative Terms




Verification: are we making the product right?

Validation: are we making the right product?

Qualification:
- coding conventions,
- good SE practices
- general standards

Circle Area Example:
- Verification: Is it programmed as $\pi r^2$?
- Validation: Did the customer actually need this?

Past Data
- Phase Originated Defects (PDD)
- Past Defects (PD)
- Filtering Effectiveness (FE)
- Removed Defects (RD)
- Cost of Defect Removal (CDR)
- Total Removal Cost (TRC)

$$RD = (PDD + PD) \times \frac{FE}{100}$$

$$TRC = RD \times CDR$$

Assumptions:
- All SQA activities are the filters
- Past data is accurate and still relevant.

Question 03 (b)

SQE (09)

| No. | Software Development Phase | originating % of defects in this phase |
|---|---|---|
| 1 | Requirements | 25 % |
| 2 | Design | 40 % |
| 3 | Coding (30% coding, 10% integration) | |
| 4 | Documentation | 10% |

Standard QA plan (Plan A) → Example: 1773.8 cu total

| No. | QA activity | Filtering effectiveness | Cost of removing defects |
|---|---|---|---|
| 1 | Requirement specification Review | 50% | 1 cu |
| 2 | Design review | 50% | 2.5 cu |
| 3 | Unit Test - Code | 50% | 6.5 cu |
| 4 | Integration test | 50% | 16 cu |
| 5 | Documentation Review | 50% | 16 cu |
| 6 | System test | 50% | 40 cu |
| 7 | Operations phase | 100% | 110 cu |

Comprehensive SQA Plan → Example: 1046.4 cu total

| No. | QA Activity | Filtering effectiveness | Cost of Removing defects |
|---|---|---|---|
| 1 | Requirement specification review | 60% | 1 |
| 2 | Design inspection | 70% | 2.5 |
| 3 | Design review | 60% | 2.5 |
| 4 | Code inspection | 70% | 0.5 |
| 5 | Code - ~~room~~ unit test | 40% | 6.5 |
| 6 | Integration test | 60% | 16 |
| 7 | Documentation review | 60% | 16 |
| 8 | System test | 60% | 40 |
| 9 | Operations Phase | 100% | 110 |

# Contract Review

Pre-project component

Crime has Consequences

SS — SC

Legal contract
- handshake
- both parties are signatories

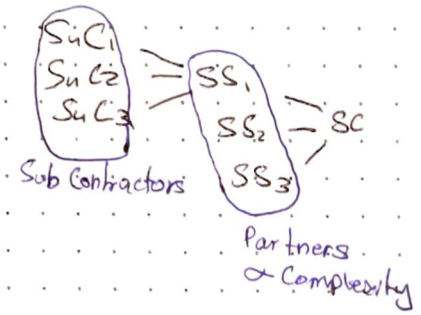
RFP → Proposal Draft → Proposal → Contract Draft
(Request for Proposal)
+ Review
submit
if accepted
+ Review
sign

- Requirements clarified
  - scope
- Alternative Approaches
- Relationship Formality
  - Focal Person representative
  - Project deliverables
  - Change approval process
- Risks
  - technological
  - non-technical (personnel turnover, economic, political conditions)
- Estimates → estimation techniques
  - Effort
  - Duration
- Capacity / Capability
  - Can the supplier do it?
  - Can the consumer pay us?
- Participants
- Intellectual Property
  - Who gets to own the copyrights.

a legal document → Contract

Formality benefits both sides by enforcing protocols.
↳ SQA activities should be scheduled.

SS ~~ SC
blurred when
- SS and SC belong to the same org.
- embedded systems
- products for customers




## Goals of Contract Draft Reviews

- no unclarified issues.
- understandings documented
- no "new" changes

no ambiguity
no less
no more

extent of the contract draft review
- depends on project size, complexity, organisational complexity, acquaintance/familiarity/expertise

## Characteristics of Bad Contracts

- unrealistic estimates
- loose requirements

estimates can vary based on the software development process. But the choice of the process need not be documented.

SQE LII 25-09-2024

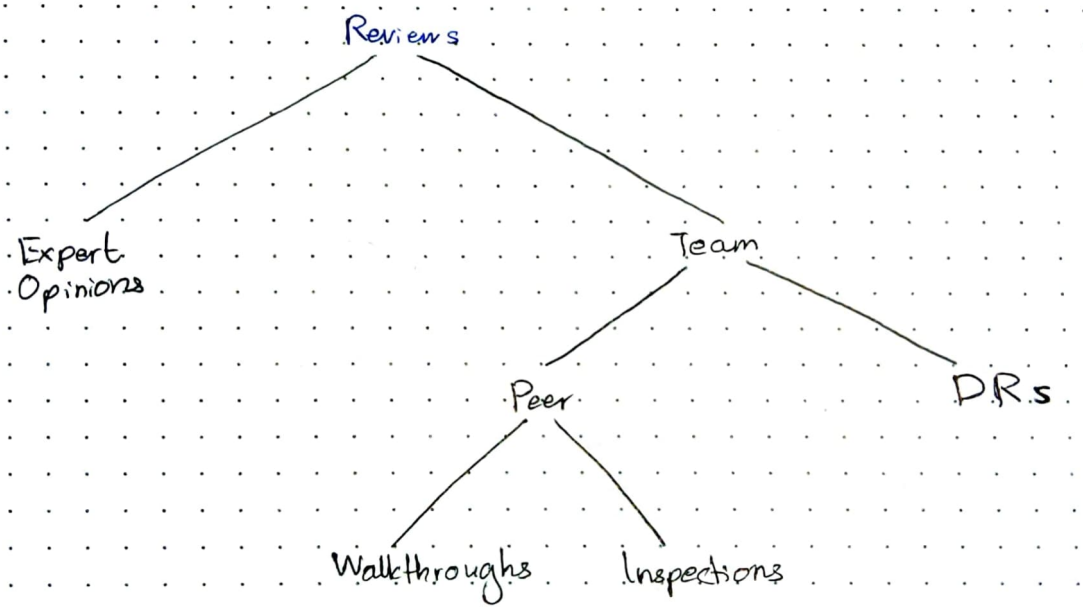
~~FORMAL TECHNICAL~~ REVIEWS → lifecycle components

Review: critical evaluation/appraisal; formal
- Used in thesis reviews, FYP reviews
- double-blind review: no one knows who is the other person
- Reviews are used in other fields

- Reviews also focus on code, but primarily on docs.

- Reviews are generally good at detecting defects. They act as filters to remove defects.

Primary focus of:
- reviews → Software = - Programs, - Docs, - Data
- Testing → Software = - Programs, - Docs, - Data




Emphasis on objectives varies from review to review

Direct Objectives (deal with the project at hand)

- detect and remove as many upstream faults (check incomplete, missing info/parts)
- identify new risks
  - older risks already identified in risk plan
- detect deviations from templates/conventions/standards
  - deviation cause problems downstream in maintenance
- grant or deny approval

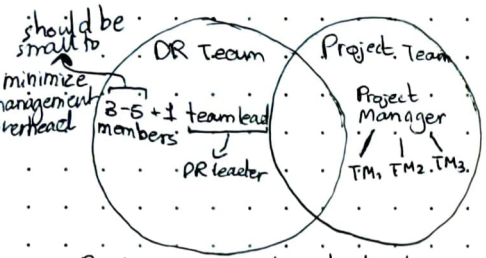
Indirect Objectives (deal with all projects)
- exchange professional knowledge (in meetings of professionals).
- record/analyze data related to faults

Expert Opinions
↳ Outside consultants, senior people

When outsiders are needed?
- when the SC is small and doesn't have enough manpower
- when the in-house expertise is not enough
- resolve disagreement

Design Reviews (a.k.a. Formal Design Reviews)
(a.k.a Formal Technical Reviews)
↳ most formal, most authoritative

4 segments of DR: ~~before~~ planning, before, during, after.



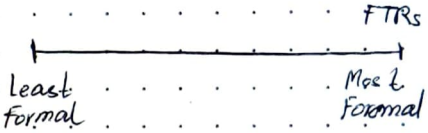

People may overlap, but play different roles.

Planning: assemble the team and appoint DR Leader.

DR leader:
- should not be the project manager of the project at hand
- should be preferably more senior than the project manager
- may be the project manager of another project
- should be familiar with the type of project

DR team members:
- majority of these should not be from the project team
  - only senior people could join in a minority
- should be diverse (to get a diverse perspective of feedback)

The formality spectrum




DR Leader ensures:
- no personal criticisms
- schedule is followed
- offline discussion of internal conflicts
- external experts may be involved

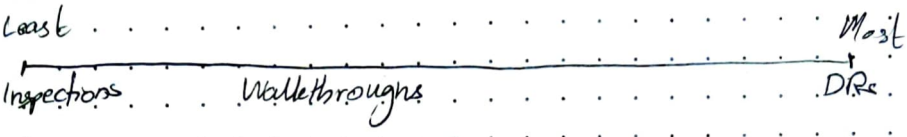
Before
- DR leader assigns responsibilities and sets a schedule.
  - small projects may be reviewed in entirety by everyone, otherwise work will be divided.
- ~~DR is reviewed~~ Review documents against checklists
- DR team comes prepared with comments
- Project team prepares short presentation.

During
- DR session starts with a short presentation.
- The comments and action items are discussed along with assignees, due dates, and verification assignees.
- DR leader is usually the scribe
- Decision (full/partial approval, or denial) is ~~sent~~ made
- As many sessions may be held as needed, but one session should not last more than 2 hours.

After
- PR leader prepares a DR report quickly and it is shared with stakeholders
- Follow-ups to reflect on the ~~process~~ DR.
- If something is not fixed satisfactorily another follow-up is needed.

## PEER Reviews 27-09-2024

- Formality




- Team composition:
  - Peers make-up a PR team
    ↳ peers of the author of the document

| | Inspections | Walkthroughs → quicker but less comprehensive |
|---|---|---|
| Planning Phase | Leader: Moderator<br>Team members:<br>- Author(s)<br>- a designer<br>- a programmer<br>- a tester<br>- Project Manager should not be the leader | Leader: Coordinator<br>Team Members:<br>- Author(s)<br>- a standards enforcer<br>- a user representative<br>- a maintenance person<br>- Project Manager may be the leader |
| Before Phase | -Overview Meeting<br>- Moderator should give an overview of the document being ~~reviewed~~ inspected if team members are outsiders.<br>- Thorough Preparations:<br>- Discuss Comments<br>- Evaluate documents against checklists | -- Brief Preparation<br>-skimming the documents,<br>-possibly giving comments |
| During Phase | - Short presentation<br>-Programmer presents the document<br>- Sessions like in DRs<br>- No decisions taken | - Short presentation<br>- document is presented by the authors<br>- Sessions like in DRs<br>- No decisions taken |
| After Phase | - 2 reports → findings → summary<br>- Follow-up | - 1 report → findings<br>- no follow-up |

| Effort | vs. | Duration |
|---|---|---|
| - person hours<br>- PDs<br>- PW<br>- PM<br>- PY | | - hours<br>- days<br>- weeks<br>- months<br>- years |

SQE 02-10-2024

# Software Testing Strategies

Inc RE A D I T D/D C-M O/R

Testing

- the classical, oldest, last-line-of-defense against faults.
- takes the most amount of SQA resources.
- Testing requires execution of code
  - not every execution of code ~~requires~~ is testing
- Testing requires a certain mindset that faults need to be uncovered.

Coders:
- programmable or moonshot?

Testers:
- quantified
- reqs. testable?

Designers:
- use-cases fishy w.r.t GUI
- modularity, coupling, cohesion
- functional independence

| Software | Testing | Strategies | Techniques |
|---|---|---|---|
| -programs/code<br>-data<br>-documentation | -execution<br>-intention to find faults<br>-formal<br>-approval<br>-third party | -what<br>-who<br>-when | -how |

→ high-level

→ low-level

optional

Strategy vs. Technique

## Objectives of Testing

Direct:
- identify as many faults as possible and fix them
- do this efficiently
- acceptable quality

Indirect:
- to compile a record of fault data

## Software Testing Strategies

General Testing Strategy:

**Unit Testing**
- ~~test integ. related units~~
- test a single unit
- unit = set of related classes/functions

- data structures - UI
- algorithms
- loops
- conditions

**Integration Testing**
- test integrated modules.

- big-bang approach
- incremental approach
  - ~~TD~~ Top-down - Depth first
  - Bottom-up - Breadth first

**Validation Testing**
- test conformance to requirements

**System Testing**
- non-functional requirements
- performance test intended
- stress test on ~~related~~ hardware

- more related to embedded systems.

SQE 04-10-2024

The Hawthorne Effect

Decision-Makers vs. Work-forces

Integration Testing

Big-Bang vs. Incremental

Integrate everything together at the same time.

Works well for small projects.

Pinpointing the errors in the modules is more difficult.

Incremental → Requires "Dummy" Modules

Integrate step-by-step.

Overhead Cost.

Which one to use? Follow the development approach.

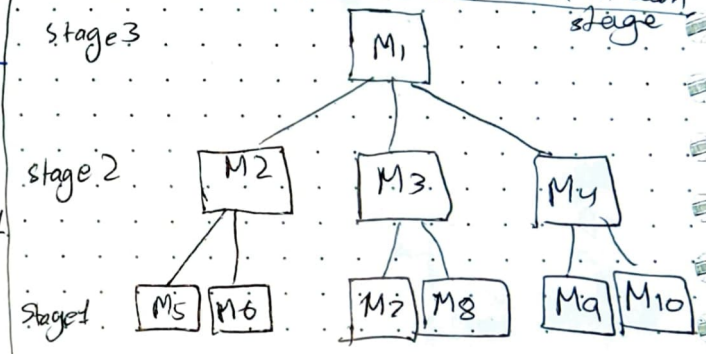
Top-Down ("Stubs") Bottom-Up ("Drivers")

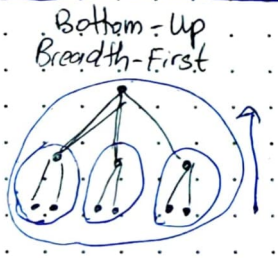
- "Details" "Error Message" are ambiguous
- Database, backend things should not be present in the SRS.
- Formatting issue should be mentioned.

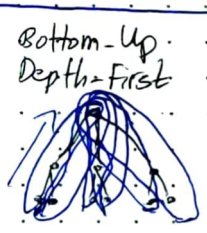
User Acceptance Testing (UAT)

- Bespoke/custom products require UAT.
- Done in natural environment

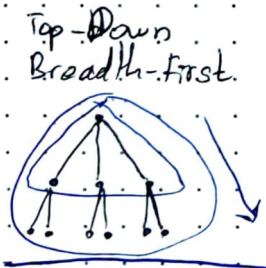
Develop and unit tested in each stage




Depth-First: single features are tested earlier in their entirety.





Smoke Testing:

- Electronics ~~short-circuit~~ release smoke only when there's a short-circuit.
- tests whether
  - software runs
  - does not crash
  - major features work (no show-stoppers)

also known as "Build-Verification Testing"

Validation Testing:

if end users involved; not professional testers

- alpha testing
  - end-users test the software in a controlled environment
- beta testing
  - end-users test the software in their natural environment

differentiation Hawthorne effect

used for generic (non-market) products

Regression Testing:

- Done after each change
- Test subset of previous test cases to check if the change hampers previously working code.

SQE 09-10-2024.

# Software Tests

NCCS

consume the most SQA resources.

consumes the most testing resources

## Concept

The Box is the "software".

- BlackBox (BB) aka functional testing
- WhiteBox (WB) aka Glass-Box, aka structural testing (code, docs + data)
- Hybrid Versions, like Grey-Boxtesting.

## Quality Factors

- Operation
  - Correctness
  - Reliability
  - Efficiency
  - Integrity
  - Usability
- Revision
  - Maintainability
  - Flexibility
  - Testability
- Transition
  - Interoperability
  - Portability
  - Reusability

- Correctness: WB/BB
  - correctness of docs (like manuals, user, installation manuals, etc.)
  - correctness of outputs
    - with/without focusing on details
  - availability testing
    - usually done with load testing
    - response time as a function of load
    - usually automated

- Reliability (BB)
  - MTBF between failures
  - MTTR to repair
  - MDTM down-time per month
  - Statistical models exist for reliability testing

- Efficiency: (BB)
  - Stress tests
    - load testing + ~~durab~~
      - behavior/performance of the system under abnormal load.
    - durability testing
      - expose system to abnormal physical conditions.

- Integrity/Security Testing (mostly BB)
  - Malware, viruses, trojans
  - SQL injection
  - ethical hacking

- Usability Testing (mostly BB)
  - Training usability testing
  - Operational usability testing
  - difficulties increase error chances

- Maintainability (BB/WB)
  - corrective
  - modular design, functional
- Flexibility (BB/WB) independence, low coupling, high-cohesion.
- Testability (BB)
  - intermediate results
  - in-house checks
- Portability (BB)
  - The software works on other systems or not
- Reusability (WB)
  - Satisfies criteria for inclusion in the re-use library
  - Coupling, cohesion, functional independence
- Interoperability (BB)
  - Working with other softwares/interfaces/systems.

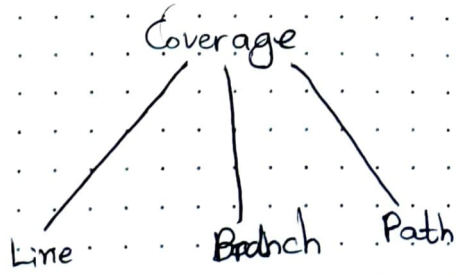
SQE 11-10-2024

# WHITE BOX TESTING

If there are 10 sequential, un-nested conditionals, there are $2^{10} = 1024$ possibilities.

**EXHAUSTIVE TESTING IS NOT PRACTICAL**

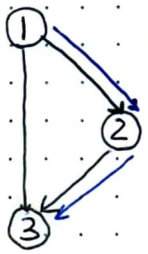
- selective testing is preferred, unless the module is very critical.




- All programming language statements must be executed at least once:
  - ↳ Line coverage = 100%.

100% line coverage ---does not guarantee---> 100% branch coverage

100% branch coverage ---guarantees---> 100% line coverage



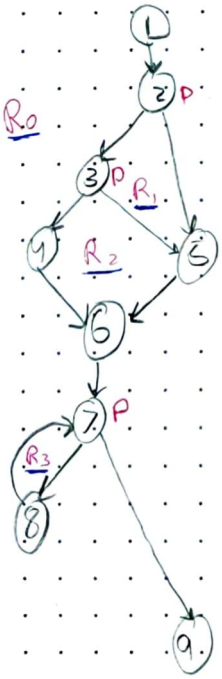

This test case covers 100% of the lines, without covering all branch.

Each branch is

Path → Branch → Line

Basis Path Testing / Basic Path Testing
↳ Guarantees 100% Branch Coverage.

Code → Program Flow Graph (G) (has certain rules) → Cyclomatic Complexity V(G) (Upper bound on the no. of basis paths.) → Basis Paths → Voila! 100% Branch coverage.




V(G) = R = ER + 1 = 3 + 1 = 4 (Regions; Enclosed Region)

V(G) = P + 1 = 3 + 1 = 4

V(G) = E − N + 2 = 11 − 9 + 2 = 4 (Edges; Nodes)

Predicate Nodes: Nodes with 2 or more outgoing edges

Upper bound on the amount of basis paths needed for 100% Branch Coverage.

It is a UPPER BOUND it is possible to achieve 100% branch coverage with lesser test-cases.

lefty-truthy, righty-falsey.

Rules
1. Bunch together all consecutive sequential non-branching statements in a single node.
2. All loop/if/else conditions become one node.

SQE LJ7 16-10-2024

## Set of Basis Paths

IDEs may calculate the cyclomatic complexity.
- It can be done by identifying the nodes and creating an nxn adjacency matrix for the control flow graph.

V(G) is used for test prioritization.

→ Independent (by definition)
→ Subsequent paths remain independent by adding at least one such edge which is not present in any of the previous/former independent paths.

| S# | Path |
|---|---|
| 1. | 1 → 2 → 5 → 6 → 7 → 9 |
| 2. | 1 → 2 → 5 → 6 → 7 → 8 → 7 → 9 |
| 3. | 1 → 2 → 3 → 4 → 6 → 7 → 9 |
| 4. | 1 → 2 → 3 → 5 → 6 → 7 → 9 |

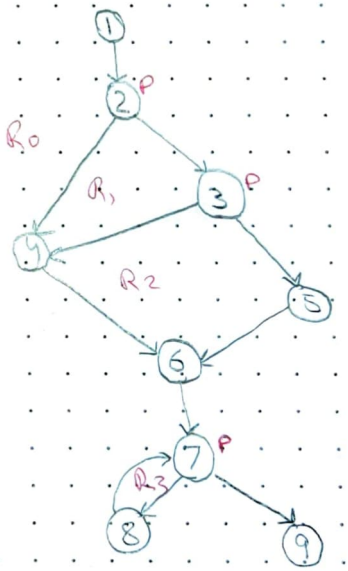
V(G) is an upper limit; there may be less than V(G) paths which guarantee 100% branch coverage

V(G) is the cyclomatic complexity, which is to be used for calculating the Weighted Methods per Class (WMC) in the CK set of O.O. Metrics.
$WMC = \sum c_i$

| S# | Path |
|---|---|
| 1. | 1 → 2 → 3 → 4 → 6 → 7 → 8 → 7 → 9 |
| 2. | 1 → 2 → 5 → 6 → 7 → 9 |
| 3. | 1 → 2 → 3 → 5 → 6 → 7 → 9 |

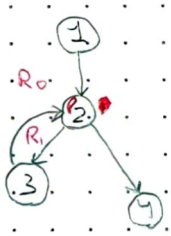
---

| S# | Path |
|---|---|
| 1. | 1 → 2 → 3 → 5 → 6 → 7 → 8 → 7 → 9 |
| 2. | 1 → 2 → 3 → 4 → 6 → 7 → 8 → 7 → 9 |
| 3. | 1 → 2 → 3 → 5 → 6 → 7 → 9 |



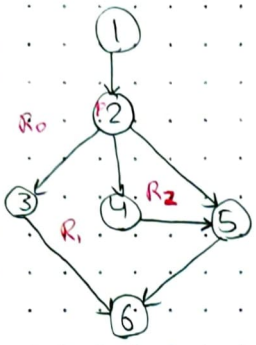

---

| S# | Path |
|---|---|
| 1. | 1 → 2 → 3 → 2 → 4 |




```
void foo() {
    //...          1
    for(int i=0; i<10; i++) {   2
        //...      3
    }
    //...          4
}
```

V(G):
= R# = ER+1 = 1+1 = 2
= E − N + 2 = 4 − 4 + 2 = 2
= P + 1 = 1 + 1 = 2




$V(G) = R = ER + 1 = 2 + 1 = 3$
$= P + 1 = 1 + 1 = 2$
$= E - N + 2 = 7 - 6 + 2 = 3$
$\Rightarrow V(G) = 3$

```
int selection (int option){
  int value;
  int denominator;
  switch(option){
    case 1:
    value = option++;
    denominator = 2;
    break;

    case 2:
    value = option--;
    denominator = 4;
    default:
    value = 0
    denominator = 1;
    break;
  }
  return (value/denominator);
}
```

(1)

```
void foo()
{
  ...
  bar();
  ...
}
```

function calls are non-branching.

```
void bar(){
  ...
}
```

- Recursive calls are just like function calls.

---

SQE L18 18-10-2024

## Testing

## Condition Statements

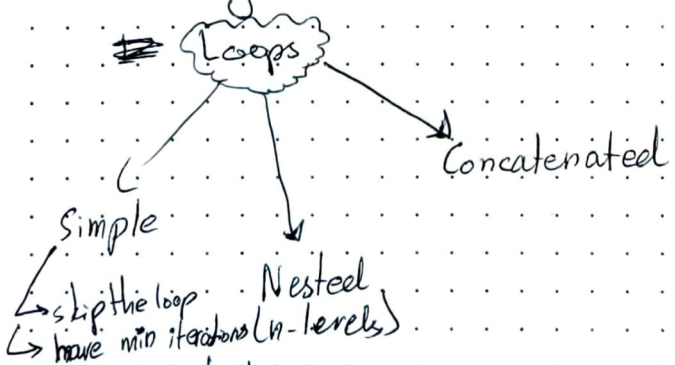
↳ have more error chances
↳ if-else-if/switch
↳ use operators
- relational >, <, >=, <=, ==
- ~~logical~~ logical &&, ||, !
  - incorrect operators
  - missing operators
  - parantheses

Boris Beizer: "Bugs lurk in corners and congregrate at boundaries"

Nested loops:

a- Begin with inner loop, keep all outer loops at min. iterations
b- Move out one loop at a time.

## Loop Testing:




↳ skip the loop
↳ have min iterations (n-levels)
↳ have max iterations
↳ have somewhere in between the range iterations
↳ more than max. iterations

Concatenated:

Independent scenario:

Do simple testing of each loop in sequence.

Dependent Scenario:

Treat loops as nested.

## Advantages of White Box Testing

- Test source code directly
- Coverage can be controlled directly
- Qualification Testing (conformance to standards) can also be tested.

Disadvantages:

- Resource-heavy, takes more time than black-box testing.
- Options are limited

SQE
23-10-2024

~~Chapter 6:~~

~~Hoyd's Algorithm~~

## Black-Box Correctness Testing

↳ since source code is not available, we test the correctness of the outputs.

## Equivalence Class Partitioning (ECP)

Input is partitioned into equivalence classes.

Equivalence Class: a set which contains inputs which make the software behave in the same manner.

- created for each variable, keeping in mind the software.

CGPA
B1) 0.0 - 1.9 → 0.5, 1.2, etc.
B2) 2.0 - 4.0 → 2.1, 3.7, etc.

Invalid Equivalence Classes: sets of input values to which the software responds with specific error messages. Each type of invalid ~~e~~ input will have a separate equivalence class.

## Coverage in Correctness Testing:

We need minimum test-cases

Rules:
1- Write a separate test case for each invalid e.class and test only on e.c in each test case.
2- Write a single test case for all the valid e.classes together.

The idea is that separately checking incorrect inputs easily identifies the underlying error.

→ Black-box test cases may be written even before the code is available. Although code is needed to execute test-cases, it is not needed to 'write' these test-cases.

## Boundary Value Analysis (BVA)

The boundary values of all equivalence classes used for testing.

- can not be used for discrete variables, like gender, types, etc.

SC Example
↳ Swimming Centre

- Day of the week → Weekday, Week-end day
- Visitor's status → Member, One-time visitor
- Visit Time → Shift-1, Shift-2
- Age → Young, Middle, Elderly

Possible Valid Combinations = 2×2×2×3 = 24

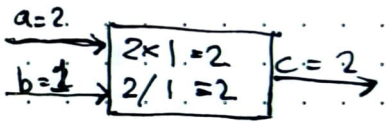
- Costs given in ~~table~~ 9.7, Galin.

Number of test-cases in 1A = maximum no. of ~~input~~ equivalence classes of any input variable

" in 1B = maximum no. of boundaries

" in 2 = no. of invalid equivalence classes.

= twice the number of equivalence classes of a continuous input variable with the maximum no. of equivalence classes




Advantages

- Cheaper than whitebox testing (where both can be used)
- Many options

Disadvantages

- Qualification testing not possible
- No control on coverage
- We may get the correct output by chance

no fine control on code coverage. Some control on conceptual coverage.

SQE 30-10-2024

# THE TESTING PROCESS

Decide Methodology → Plan → Test Case Design → Execute

(Reports are prepared perhaps automated.)

- Acceptable level of quality
- Strategies & Techniques

- What do we test?
- Who performs the tests? ss/sc/consultants
- Where is the testing performed? ss/sc/consultants
- When do we stop testing? How much testing is enough?

(1-5) A Damage Level

(1-5) B Risk Level

High-level decisions to be made at start of the project:

- Acceptable Level of Quality
- Strategies & Techniques
  - Incremental / Big-Bang
    - Top-down / bottom-up
      - Breadth-first / Depth-first

A → How dangerous is the damage?

B → How likely is the damage.

i - A+B

ii - kA + mB

iii - A * B ← Risk Exposure (Expected damage)

Used in project management to prioritize project risks.

Stopping:

- Stop when all is tested
  - assumes unlimited budget
  - may work for tiny projects.
- stop when we have run out of ~~no more~~ money/time
  - surprisingly, very common.
- stop based on mathematical models
  - a threshold is defined for the "defect detection rate" based on past data
  - criticisms:
    - different severity-level defects treated equally
    - current data not reliable predictor of future data
- fault seeding
  - intentionally inject defects
  - define threshold
  - premise: correspondence between seeded and actual defects.
  - seeded faults removed before delivery.

- dual independent testing teams.

# TEST CASE DESIGN

Test Case: a set of input values along with operating conditions and an expected output.

At least 1 test case for each course of action of every use-case.

Priority = {"High", "Medium", "Low"},

---

## Dual Independent Testing Teams Method

$N_a$ → no. of faults detected by team A
$N_b$ → no. of faults detected by team B
$N_{ab}$ → no. of faults detected by both teams
$N$ → total no. of faults
$T$ → threshold

$$P_a = \frac{N_a}{N} = \frac{N_a}{\frac{N_a \times N_b}{N_{ab}}} = \frac{N_{ab}}{N_b}$$

$$P_b = \frac{N_b}{N} = \frac{N_b}{\frac{N_a \times N_b}{N_{ab}}} = \frac{N_{ab}}{N_a}$$

$$P_{ab} = \frac{N_{ab}}{N} = P_a \times P_b = \frac{N_a}{N} \times \frac{N_b}{N} = \frac{N_{ab}}{N}$$

$$\frac{N_a \times N_b}{N_{ab}} = \frac{N^2}{N} = N$$

$$N_{(a)(b)} = P_{(a)(b)} \times N$$

$$P_{(a)(b)} = (1 - P_a)(1 - P_b)$$

$$= \left(1 - \frac{N_a}{N}\right)\left(1 - \frac{N_b}{N}\right)$$

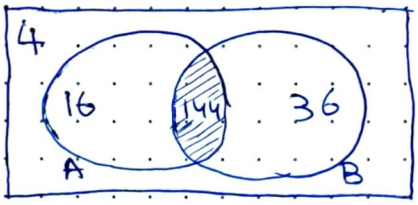
$$= \frac{(N - N_a)(N - N_b)}{N^2}$$

$$= \left(1 - \frac{N_{ab}}{N_b}\right)\left(1 - \frac{N_{ab}}{N_a}\right)$$

$$= \frac{(N_b - N_{ab})(N_a - N_{ab})}{N_a \times N_b}$$

$$= \frac{\left(\frac{N_a \times N_b}{N_{ab}} - N_a\right)\left(\frac{N_a \times N_b}{N_{ab}} - N_b\right)}{\left(\frac{N_a \times N_b}{N_{ab}}\right)^2}$$

$$= \frac{(N_a \times N_b - N_a \times N_{ab})(N_a \times N_b - N_b \times N_{ab})}{(N_a \times N_b)^2}$$




### Example:

$N_a = 160$
$N_b = 180$
$N_{ab} = 144$
$T = 2.5\% = 0.025$

$$P_{(a)(b)} = \left(1 - \frac{160}{200}\right)\left(1 - \frac{180}{200}\right)$$

$$= \left(\frac{40}{200}\right)\left(\frac{20}{200}\right)$$

$$= 0.02 = 2\%$$

since $2\% < 2.5\%$, testing may stop

$$N_{(a)(b)} = P_{(a)(b)} \times N$$

$$= 0.02 \times 200$$

$$= 4$$

$$\frac{N_{ab}}{N} = \frac{144}{200} = 0.72$$

$$P_a \times P_b = \frac{160}{200} \times \frac{180}{200} = 0.72$$

Using simplified formula:

$$P_{(a)(b)} = \frac{(180 - 144)(160 - 144)}{160 \times 180}$$

$$= 0.02$$

SQE L3 06-11-2024

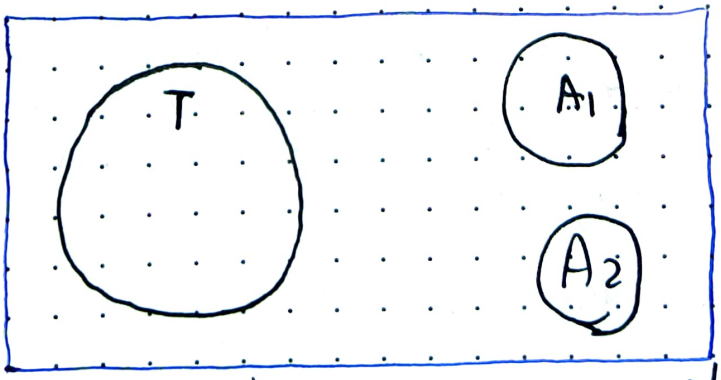
# Test Case Data Sources

## Real life samples

- usually done when a manual process is automated.
- minimizes test-case preparation effort since inputs and outputs are already present.
- more execution effort since size of test-case file is large. (→ in case of pure random sampling.)
  - test file is large because many sample data points are typical, and alternate cases are limited.

## Synthetic

- done using ECP and BVA
- more effort in preparations since inputs and outputs need to be synthesized or calculated.
- less execution effort since size of test-case file is small.




Sample

Circles do not actually exist; these are "scenarios" or "strata"

These strata are defined from the standard and alternate courses of action of each test case.

Consensus: to add both synthetic and real life data points to the test case file.
Rationale: unexpected things happen in real-life.

## Automated Testing

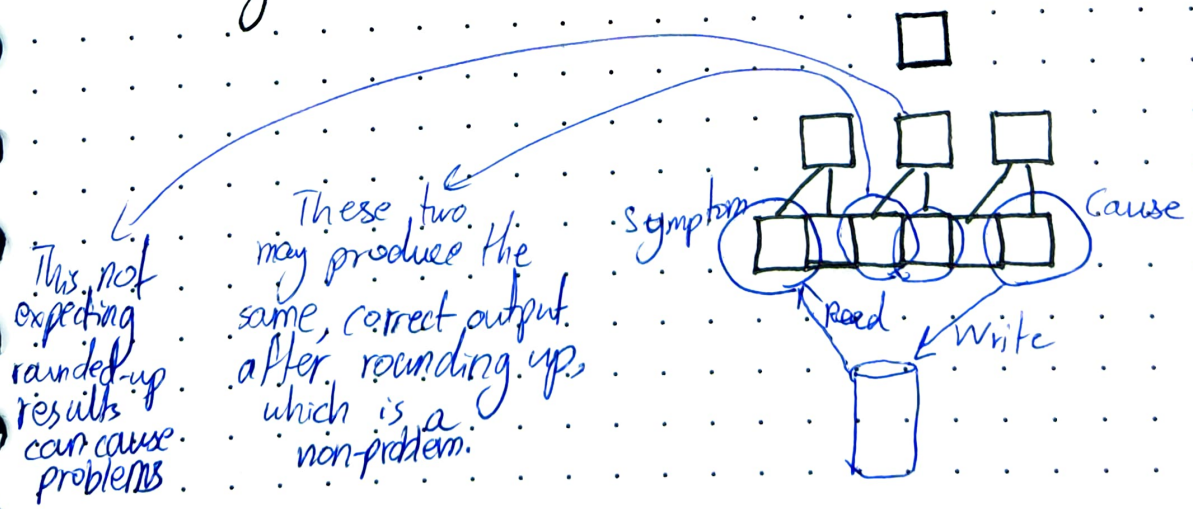
- Code auditors
  - e.g. CheckStyle for Java
  - automate qualification testing
  - generates compliance/non-compliance/summary reports
  - requires definition of these standards as input
- Coverage Monitors
  - monitor line/branch/path/conceptual coverage
  - Jacoco (Java Code Coverage)
- Functional Testing
  - black-box functionality testing (black-box output correctness tests)
  - Selenium, QTP, Catalog Studio, etc.
- Load Testing
  - LoadRunner, Apache JMeter
  - ~~output~~ simulate high load scenarios (hits-per-minute, transactions-per-minute)
- Test Management
  - BugZilla, Jira
- Usability Tools
  - Maze,

| Automated | vs. | Manual |
|---|---|---|
| - require tool licensing, training costs<br>- test-case preparation is still manual.<br>- overcome human shortcomings, like forgetfulness, boredom, demotivation.<br>- generate more accurate and complete bug reports | | |

Not everything can be / has been automated in testing.

Debugging → Trying to connect symptoms to causes. IS NOT TESTING!!!

Occurs as a consequence of a test-case that successfully fails.




Debugging may be intermittent: sometimes it gives correct output, other times not.

## Debugging Strategies:

- brute force.
  check everything when no suspected cause ~~problems~~ found.
- back tracking
  check symptom code, then the code calling the code, and so on.
- cause elimination
  write extra test cases to ensure that the suspected problem is the actual problem.

SQE 08-11-2024

# Object-Oriented Software Testing

Overall testing strategy:

small → large

unit → integration → validation → system testing

same for conventional and OO software

In OO software, unit and integration testing change meanings.

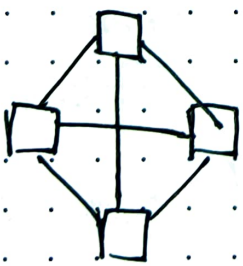
## OO Unit Testing

- Unit is a class (data and behavior)

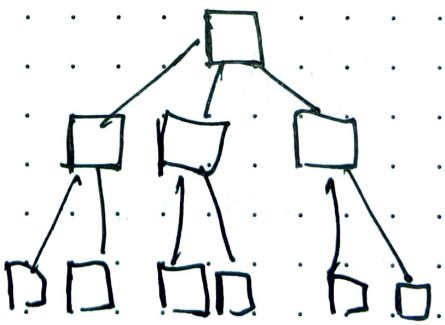
## OO Integration Testing

- ~~Related~~ related groups of classes.
- Since architectural style changes, top-down and bottom-up approaches are irrelevant.
  - stubs and drivers still exist though.

OO Architecture



Call & Return



## Integration testing approaches:

Thread-based

Classes handling similar events are tested together

eg. Student Registration, Course, etc. all handle registration events.

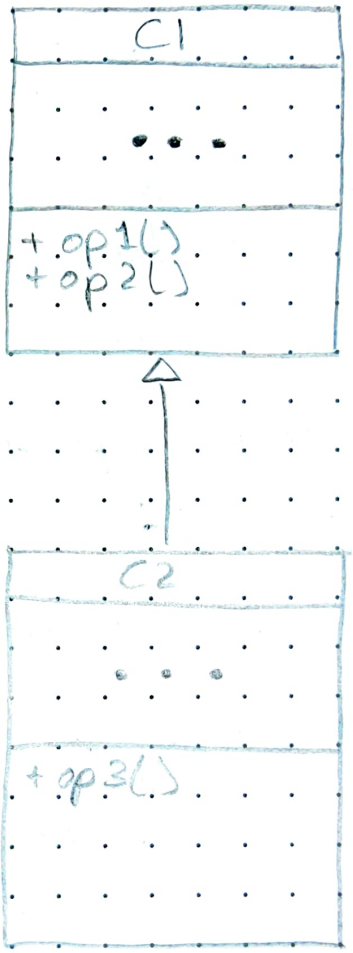
## Use-based approach:

Start by testing the independent classes, and then proceed outwards.

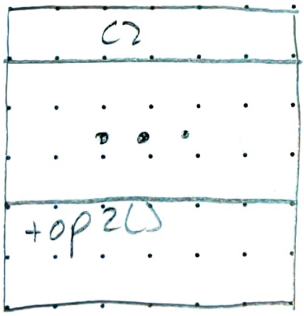
$C_1$ ——uses——▶ $C_2$
($C_1$ server) ($C_2$ client)

test $C_1$ before $C_2$.

Use test cases for $C_1$'s op1 & op2 when testing $C_2$, and write new test cases for op3.




if C2 is




write new test cases for op2 as well

if C2 is as above AND C1's op1 is:

```
// In this case
// ...
  op2();
// ...
```

we need to rewrite new test cases for op1 as well as op2.

Example of Unit Testing:

Unit: "BankAccount" class.

minimum possible sequence:

open → deposit → withdraw → close

↳ Anything can come here, e.g. as below:

→ Random Testing:

open → deposit → deposit → deposit → balance → statement →
withdraw → withdraw → close

add any random sequence of valid function calls between the boundaries.

→ Partition-based testing:

Partitioning approaches:

↳ Categories:
group operations into conceptual categories
e.g., initialization, termination, computation, query

↳ State-based:
partition based on whether the state has changed or not

↳ Attribute-based:
for each attribute, create 3 partitions:
- functions that change its value
- functions that simply use its value
- functions which do neither.

SQE 13-11-2024

# Software Configuration Management (SCM)

It is an umbrella activity: it spans the entire SDLC.

Software Configuration = Set of Software Configuration Items
=> SC = Set of SCIs.

SCIs
- documents (SRS, test plan, test-case plan, etc.)
- code

SCIs evolve over time, hence SCIs have different 'versions', hence the Software Configurations also have 'versions'.

## SCM Tasks

### Change Management

We should not 'let' change happen, changes must be controlled to avoid derailing the project. Further, changes must be accompanied by regression testing.

Software Change Request (a.k.a Change Request)

upon encounter → thorough evaluation → after evaluation → Decide

not an 'order'!

SCO (SC order)

Decide →
- Approve
- ~~Accept~~ Approve, but delay
- Deny

thorough evaluation:
- factors: importance, urgency, amount of effort, amount of time
- no. of resources, no. of impacted SCIs

Depending on contractual clauses, the customer may be charged for changes.

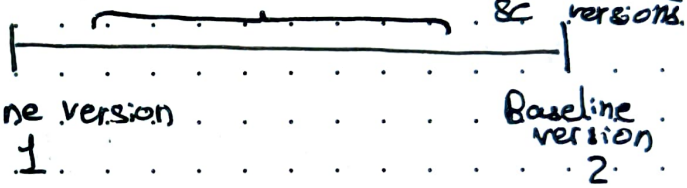
Change Management is overseen by a department called Change Control Authority/Board (CCA/CCB).

## Release Management

A software configuration (SC) is released during deployment/delivery.

Types of versions:

- Baseline: correspond to milestones, and are pre-planned.
- Intermediate: added as a result of one or more change requests




Software Configuration Management Plan:

- how many baseline versions?
- when?
- contents / impacted SCIs in baseline versions?

<Software> Ver. <Version #>.<Revision #>

↳ technically called "decimal enumeration"

↳ decimal point

↳ minor change, usually done in intermediate versions

| SCI | Flex Ver. 1.0 | Flex Ver. 1.1 | Flex Ver. 2.0 | F. Ver 2.1 |
|---|---|---|---|---|
| SRS | Ver 1.3 | Ver 1.4 | Ver 1.5 | Ver 1.5 |
| Class D. | - | - | Ver 1.1 | Ver 1.2 |
| State D. | - | - | Ver 1.2 | Ver 1.3 |
| M1 code | - | - | - | |
| M2 code | - | - | - | |
| M3 code | - | - | - | |
| Test Plan | - | - | - | |
| User Manual | - | - | - | |

It is possible for SCI versions to remain unchanged across different configurations.

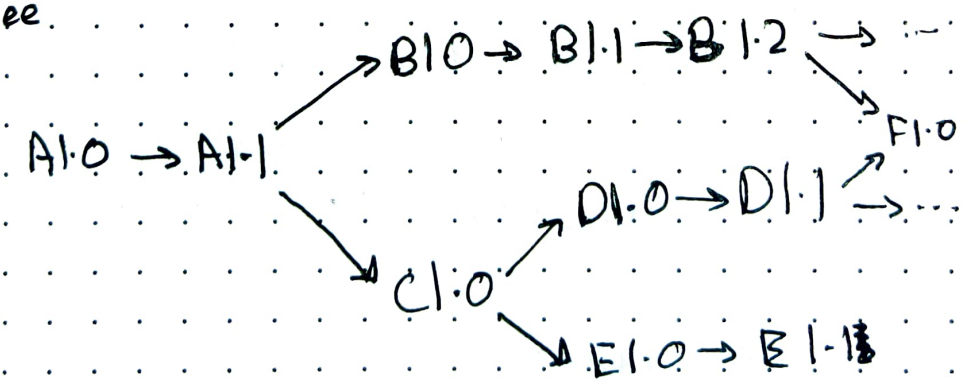
## Software Evolution Approaches

- Linear

A1.0 → A1.1 → A1.2 → A1.3 → A2.0 → A2.1 → ...

- no branches
- used for single client or when only the latest versions needs to be supported.

- Tree




- branches are multiple versions
- branches may merge, but continuing development will depend on the scenario.
- all leaf nodes are supported

- Provision of Information (SCM Task)

Version ~~Release~~ Description Document

- changes
- open issues / known issues
- deployment info (location and version)
- etc

Auditing

- verify compliance to SCM procedures..
  - e.g. how many unapproved SCRs were implemented?
  - eg: how many SCOs were delayed

SQE 15-11-2024

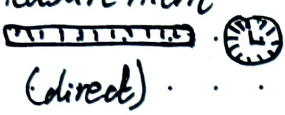
From book [4].

# Measurement Theory

**Measurement**: A process of assigning numbers/symbols to attributes of entities using certain rules.

**Quantification:**

- Measurement (direct)

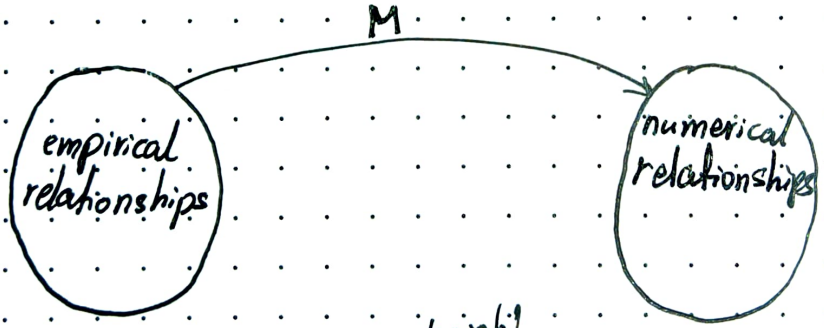


- Calculation (indirect)

If you want to understand, control, or improve something you must measure.

As if it wasn't formal enough, here's a more formal definition:

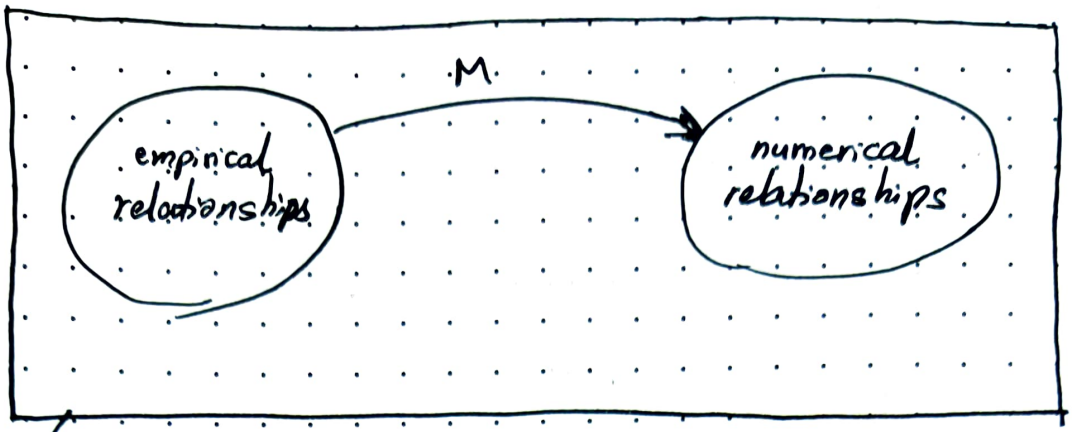
**Measurement** is a mapping between the real world and the formal mathematical world.




eg. "is taller than" — height → >

A is taller than B iff $M(A) > M(B)$

representation condition: a measure is valid if it satisfies the representation condition




- context; the measurement scale.
- measurement produces measurement(s).

## Measurement Scales.

- **Nominal scale:**
  classes where no class precedes another.
  e.g. lions, camels, monkeys → animals
  a, b, c → lions
  d, e → camels
  f, g, h, i, j → monkeys

- **Ordinal scale:**
  classes but ordered.
  e.g. all 'M' T-shirts are smaller than all 'L' T-shirts.
  The difference, interval, Δ (delta) need not be constant.

- **Interval Scale:**
  - the difference/interval/delta (Δ) between any two classes is constant.
  - e.g., temperature in °C or °F
  - zero quantity is not absolute zero.

- **Ratio Scale**
  - the ratio between any two classes is constant.
  - zero quantity is actual zero (absolute zero)

- Absolute Scale

  - no. of ex. (anything)
  - counting

Classification of Measurement Scale

| | | |
|---|---|---|
| | Nominal | Qualitative |
| | Ordinal | |
| Continuous | Interval | Quantitative |
| | Ratio | |
| Discrete | Absolute | |

Arithmetic Operations

| | | |
|---|---|---|
| Nominal, Ordinal | → | No arithmetic operations |
| Interval | → | Addition, Subtraction |
| Ratio, Absolute | → | All arithmetic operations |

Measures of Central Tendency

| | | |
|---|---|---|
| Nominal | → | mode |
| Ordinal | → | mode, median |
| Interval, Ratio, Absolute | → | mode, median, mean |

Admissable Transformation

| | | |
|---|---|---|
| Nominal | one-to-one | |
| Ordinal | monotonically increasing transformation | |
| Interval | $M' = aM + b$ | (affined transformation) $(a > 0)$ |
| Ratio | $M' = aM$ | (ratio transformation) $(a > 0)$ |
| Absolute | $M' = M$ | (identity transformation) |

| Nominal: | Ordinal | Interval | Ratio | Absolute |
|---|---|---|---|---|
| M → M' | 1st → 1 | $F = \left(\frac{9}{5}\right)C + 32$ | Inches = 12 × feet | only 1 way to measure |
| Monkey → X | 2nd → 4 | | | |
| Horse → Y | 3rd → 200 | | | |
| Camel → Z | 4th → 40000 | | | |

SQE 20-11-2024

# Software Quality Measures and Metrics

measured — calculated

## Objectives

direct:
- Understand: understand what's going on
- Control: if things don't go as planned, controlling ~~the~~ steps are taken.

indirect:
- Improving: implement improvements

---

Measures & Metrics

- process (pre-deployment/delivery)
  - Quality
  - Schedule/ Time-table
  - Productivity & Effectiveness
- product (post-deployment/delivery)
  - Help Desk
    - Quality
    - Productivity & Effectiveness
  - Maintenance (corrective)
    - Quality
    - Productivity & Effectiveness

Process metrics can be used for other types of maintenance with some minor adaptations.

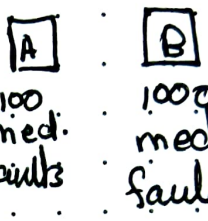
## Attributes for Successful Metrics

- Relevant: should measure attributes of significance
- Valid: should actually measure what they are designed to measure (satisfy the representation condition)
- Mutually Exclusive: ~~no two~~ measure something that has not been measured by another metric
- Reliable: produce similar results under similar circumstances
- Comprehensive: should not have limited applicability

- Easy & simple: easy to measure/calculate
- No independent data collection:

- Immune to biased intervention

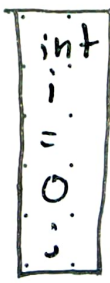
Size Metrics are used to normalize data. Comparisons can only be made once data is normalized:




Quality of A and B can be compared

Size Metrics:

- KLOC: Kilo-lines-of-code
  = $\frac{\text{lines of code}}{1000}$
  - code counters can count lines automatically
    - rules: comments/blank lines
  - objective value
  - value depends on the programming language
  - also somewhat dependent on programmer style: not immune to biased intervention




- FP: function points:
  - require human intervention: subjective
  - functionality remains the same irrespective of the programming language
  - not influenced by programmer style

IFPUG: International Function-Point User Group

- User-stories/stories points
- class-points

- function points can be calculated from the detailed SRS, hence they can be used for project estimation.

## Calculating Function Points Values (Overview)

1- CFP (Crude Function Points):
- a.k.a Unadjusted Function Points
- Components:
  - User Inputs — UI
  - User Outputs — UO
  - User Online Queries — UCQ
  - Logical Files (LDB tables) — LF
  - External Interfaces — EI
- Each component is weighed on an ordinal scale:
  simple, medium, complex

2- RCAF (Relative Complexity Adjustment Factor)

$$RCAF = \sum_{i=1}^{5} S_i \quad \text{where } S_i = 0, 1, 2, 3, 4, 5$$

$$0 \leq RCAF \leq 70$$

3- $FP = CFP(0.65 + 0.01 RCAF)$

Gearing / Conversion Factors / Ratios

QSM

for attend-master in Java:

$FP = 85.86$ Java Gearing Factor = 53

$FP = 85.86 \times 53 = 4,550.58$ LOC

$\frac{4550.58}{1000} \approx 4.55$ KLOC

QSM $CF_{language}$

$$CF_{Language} = \frac{LOC}{FP}$$

$$FP \times CF_{language} = \cancel{FP} \times \frac{LOC}{\cancel{FP}} = LOC$$

$$KLOC = \frac{LOC}{1000}$$

---

## ERROR DENSITY METRICS → Process Quality

$$Density = \frac{Mass}{Volume}$$ for software, volume = size

NCE = no. of code errors
NDE = no. of development errors

$CE \subset DE$
$\Rightarrow NCE \leq NDE$

WCE = weighted NCE
WDE = weighted NDE

**CED** = Code Error Density = $\frac{NCE}{KLOC}$

**DED** = Development Error Density = $\frac{NDE}{KLOC}$

**WCED** = Weighted CED = $\frac{WCE}{KLOC}$

**WDED** = Weighted DED = $\frac{WDE}{KLOC}$

**WCEF** = Weighted CE per FP = $\frac{WCE}{NFP}$

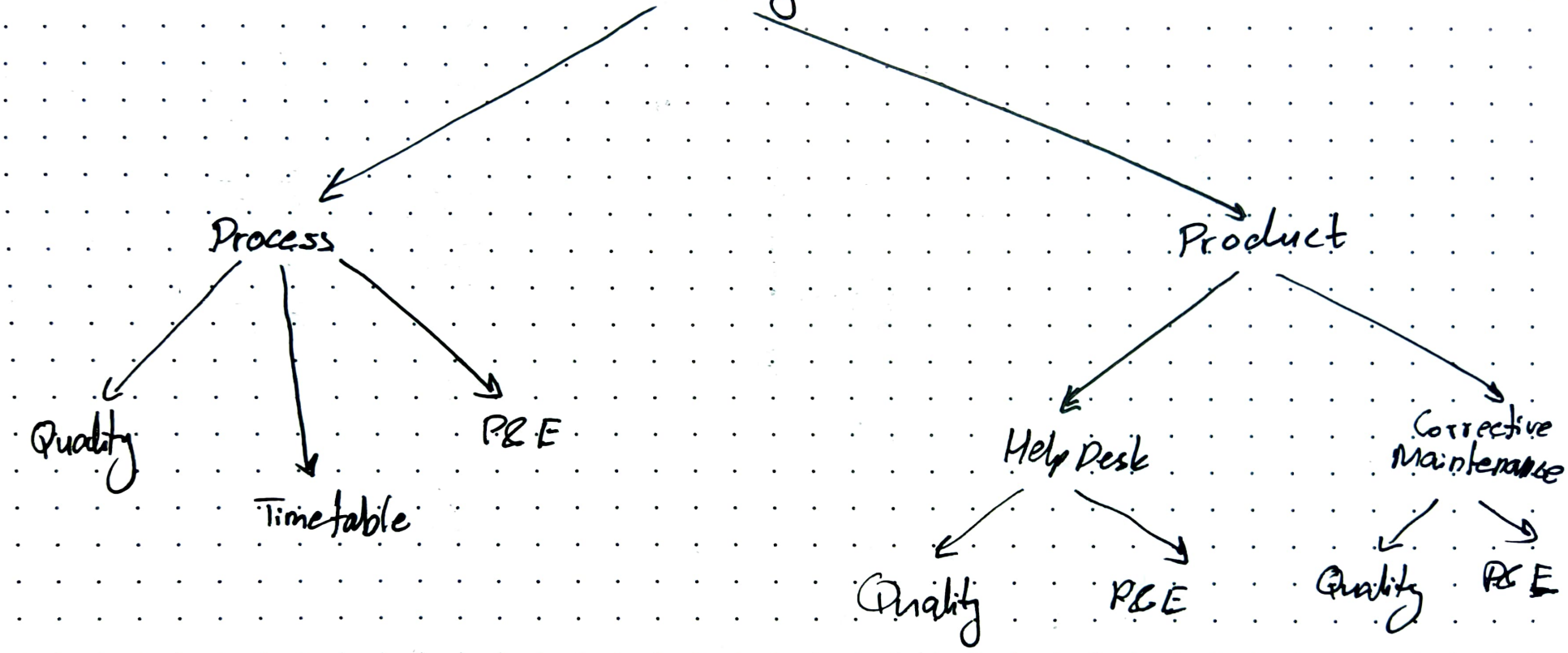
**WDEF** = Weighted DE per FP = $\frac{WDE}{NFP}$

Example:

NCE = 70

| | | | | |
|---|---|---|---|---|
| simple | 40 | 1 | = | 40 |
| medium | 20 | 3 | = | 60 |
| complex | 10 | 9 | = | 90 |
| | 70 | | | 190 → WCE |

# Software Quality Metrics




SQE 22-11-2024

## Error Severity Metrics (Higher values bad)

ASCE = Average Severity of Code Errors = $\frac{WCE}{NCE}$

ASDE = Average Severity of Development Errors = $\frac{WDE}{NDE}$

## Process Timetable Metrics

TTO = Time Table Observance = $\frac{MSOT}{MS}$

ADMC = Average Delay of Milestone Completion = $\frac{TCDAM}{MS}$

MSOT = Milestones completed on time
MS = Total number of milestones
TCDAM = Total completion Delays (days, weeks, etc.) for all milestones.

ideally, MSOT = MS, and TTO = 1.
before-time completion is negative delay.

## Error Removal Effectiveness Metrics

ideally, NYF should be zero.

## Process Productivity Metrics

DevP = Development Productivity = $\frac{DevH}{KLOC}$

minimize DevP. (more DevH/KLOC ⇒ less prod.)

FDevP = $\frac{DevH}{NFP}$

## HD Calls Density Metrics (HD quality)

$$HDD = \text{HD calls density} = \frac{NHVC}{KLMC}$$

KLMC = KLOC - unmaintained code.

NMFP = NFP - reused part (unmaintained part).

## Severity of HD Calls Metrics

$$ASHC = \frac{WHVC}{NHVC}$$

## HD Success Metrics (quality)

$$HDS = \frac{NHVOT}{NHVC}$$

## HD P&E

$$HDP = \frac{HDVM}{KLMC}$$ (lower the better)

HDE (lower the better).

## Software System Failure Density Metrics

$$SSFD = \frac{NVF}{KLMC}$$ + weighted + FP variant.

IN Severity Metrics

## Failure of maintenance service metrics

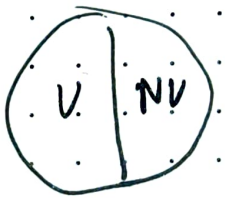
$$MRepF = \frac{RepXF}{NXF}$$ (lower the better)

## Software System Availability Metrics

Uptime / Downtime

FA → full (vital + non-vital)
VitA → vital (only vital)
TUA → none (no availability)

ideally, FA, VitA → 1, TUA → 0




$$NXFH \geq NXVitFH \geq NXTFH$$

$$\frac{NXserH - NXFH}{NXserH} \leq \frac{NXserH - NXVitH}{NXserH} \leq \frac{NXserH - NXTFH}{NXserH}$$

$$FA \leq VitA \leq (1 - TUA)$$

- FA: everything was available
- VitA: all vitals were available
- (1 − TUA): something was available