Introduction

23/09/24

There's nothing called finished product in SE.

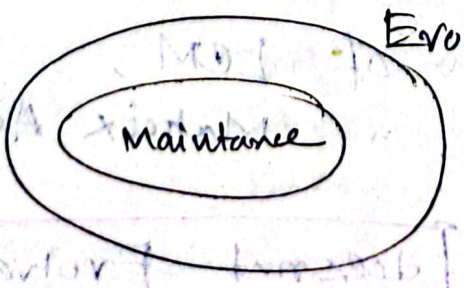
- core cost in Software maintainence.
- Software Evolution:
  - New feature addition.
- Maintainence Activity.
  - Bug fixing
  - feature upgrade

} through this process the software evolves.

- Difference between Maitainance vs Evolution.

Software এর requirements থেকে deviate না করে এটার জন্য maintainence করতে হয়।
- bug fix
- update feature

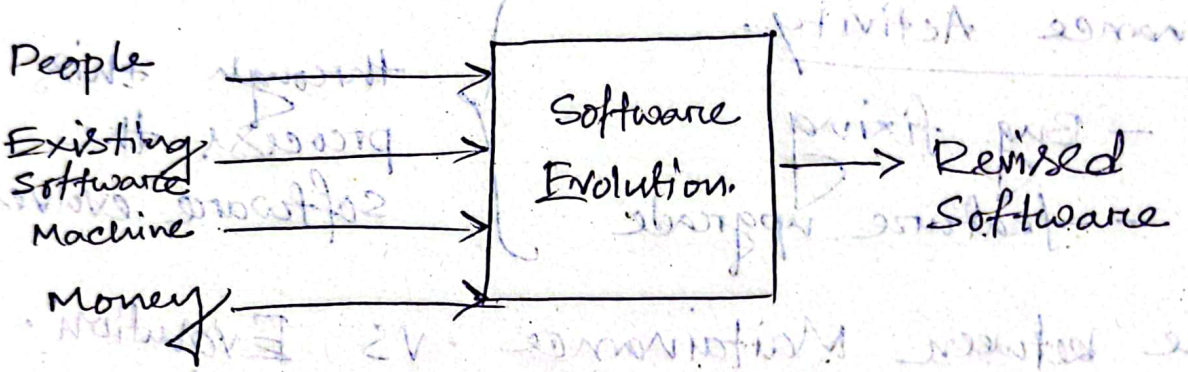
Software evolve করে new new feature add করে, better state এ grow করাকে বলা হয়।
- addition of new feature.
- new requirement
- new environment এ adapt করা



* All activities after software delivery
↓
Maintance

all activities to effect new changes in requirements.
↓
evolution

* which types of software actually evolves:



* SPE Taxonomy
  - Evolving
  - Problem
  - Specified

* S-type program: (Specified)

Formally/mathematically defined.

Example:

* Calculation of LCM,
* " " Matrix Addition.

These progreams [doesn't Evolve]

* P-type program :

100% accurate answer is not expected.

Example:

- Chess Game.
- Gomuku Game

Each steps are defined thus those step calculation are S-type But overall the software P-type

* E-type program:

Always evolving. Embedded in real-world & it changes as the world does. No concrete requirements.

Laws of Software Evolving :

1. Continuing change. (Update)
2. Increasing Complexity.
   ↳ should work on maintainance to reduce complexity.
3. Self-regulation
   ↳ [illegible] irrelavant change [illegible], Their action control check & balances the resource usage.

{develop stable & Control manner

4. Conservation of Organisational Stability.

→ Activity rate is constant.

eg. total new feature/bugs in each release.

Release to release activity rates are stable thus organisational stability remains unchanged.

5. Conservation of familiarity.

→ In between release the products looks almost similar to its previous release so that new release doesn't make the software unfamiliar to the stakeholders. The core functionality should remain almost as same as prev. release.

6. Continuing Growth.

→ Change এ existing features এ change করার। (1st law).

Growth এ time to time new feature integrate করতে হবে। The software should evolve to be better state from the current/prev. state. Else we'll lose user satisfaction.

7. Declining Quality.

→ If the software remains the same without any maintenance/adaptation it will be perceived as declined quality software.

2019 এ কেনা phone 2024 এ slow হয়ে গেল as আগে এত used to হয়ে গেছি And in the meantime it didn't evolved enough.

8. Feedback System:

→ Found new bug → create issue
↓
Test ← PR review ← create PR
↳ merging into main.

30/09/24

Maintance Stages

① Pre-delivery stage
② Post delivery stage
↳ changing software
→ provide training
→ Operating help line

page-26

Development vs Maintance :

Start with Requirements → Timeline আছে

Event triggered like bugs → No fixed timeline
→ No planned activity

→ Bug খুঁজে বাহির করে handle করার জন্য আমরা plan করে থাকি।

Three types

① → Corrective Maintance – output ভুল, ফিক্স করা
② → Perfective " –
③ → Preventive " –
④ → Adaptive "

11/17/2024, 13:05

① Processing Failure — 2+2=5 এরর।
Performance " — query performance এ্যারর, latency এরর।

Corrective Maintance is a Reactive approach.

④ ② Changing Environment এ adapt করার জন্য Adaptive ~~maintainee~~ maintenance. core i3 এ software রান core i5 এ চলে।

② Efficiency, maintainability বাড়ানো। ~~Perform~~ perform করার Perfective maintenance করা হয়।

→ e.g. non-readable code, code smell error, refactor করার etc.

→ maintance for the sake of maintance.

③ Preventive Maintenance: coding style এ standard follow করার এবং future এ এ্যারর # of error reduced হয়।

Activity Based Classification of Software Maintenance :

- Correction
- Enhancement

01/10/24

—×—

Software maintance করলে এটার পূর্বের state এ নিয়ে যায় না কিন্তু hardware এ change এ একদম নতুন হতে পারে।

এবার software কে কত easily নতুন feature adept করা যায় / maintenance করা যায় তখন সেটাকে maintainability বলে।

↓

Software এর measurement, characteristic

Maintainability measure করার জন্য source code এনালাইসিস করি।

- Size ও maintainability Low.
  - → LOC
  - → # of files.
  - → # of methods.

  } all these measures are vague.

So we **estimate** instead of solid/concrete method like size/LOC etc.

- Complexity ↑ = Maitainability ↓ (CC)
  - ↓ (Language independent)
  - Halstead complexity. (HV)

  perCom - % of Comments

- Maintainability Index

$$= 171 - 5.2 * \ln(HV) - 0.23 * (CC) - 16.2 * \ln(LOC) + 50 * \text{sqrt}(2.46 * \text{perCOM})$$

- Bug fix করলে easy fix হয় Maintainability ↑
- Fault prone করে ↑

change prone (বারবার হলে maintainability কমে যায়)

Code Smell হলে Maintainability ↓

Bug Resolving Duration related to maintainability.

Qualitative Measures = Lab

— x —

21/10/24

Requirement + Dev =

67% operations & maintainance.

⇒ Maintenance is restricted.
Timeframe is restricted.
Testcase → regression test (কোন Testcase priority দিবে।)

Quick fix Model: (Reuse oriented)

— Immediate goal fix করা (bug)

Start from Code.

requirements ← higher level arch artifact

• Code ← lower " "

Redesign the ~~test~~ case

Iterative Enhancement:

higher level to lower level.

* Analysis is a new step in this model.
* Ripple effect.
* It works like a circular way.

In this model after updating / fixing bug the documents are properly aligned.

* Benefit

Full Reuse Model:

- Maitain separate repository for each level.
- we try to reuse the old system to upgrade the new system.

* Iterative & Incremental

Each time create executable version → make improved version of the software

non working product (new feature integrates only)

Staged model for ~~open~~ closed source software : (CSS)

open source → source code open & based on license we can modify / make commercial use.

CSS

1. Initial Development
2. Evolution ↺ → Deploy/handover
3. servicing. → major change incorporation.
4. Phaseout
5. closedown.

version changed staged

FLOSS System.

— Here in open source we can trace back to previous step (like evolution, service)

— <product><version><release><build>

—x—

Chapter-3

CR Workflow

CR — Change Request.

8 ta (state):

Current software release Committed a

CR Request form

↓

Review → Analysis / Decline → change request / duplicate check

↓

Chang

→ provide unique id for each method of SRS document.

Bug → assign → Develop → Analysis → Commit / Decline

Commit

implementation

In the mean time
① Design
② Test case generation
③ infeasible → decline

why there's no decline state from Commit stage

→ যেহেতু implement করার পরে এটা design করতেই তো decline করা যায়।

Engineering Change Document : (ECD)

Main Document ও EC document থাকে।

verified → Decline

→ Closed

MID Syllabus

04/11/24

Re-Engineering

Already implemented system কে Quality improve করার জন্য reconstruct করি।

Objectives :

1. Improve maintainability
2. Migrating to a new ~~[redacted]~~ Tech.
3. Improving Quality.
4. Prepare for functional improvement.

- increasing complexity.
- continuing change (adapt)
- Declining Quality. (Ripple effects)
  - 1টা bug solve করতে গিয়ে আরো new bug create হয়। So quality compromise করতে হয়।
  - So reengineering করলে ripple effect খুব কমে।

- Continuing Growth. (new feature)
  New feature যেন সহজে incorporate করতে পারি সেজন্য re-engineering করি।

RISKS:

① Target system is of Lower quality
② failed to met reengineering deadline.

Concepts:

① Principle of Abstraction
② " " Refinement

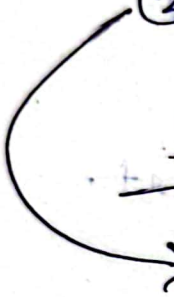


fig: 4.1.

→ Foreward Engineering

Abstract → Concrete / Refinement

#Conceptual Levels:

* why does the system exists?

Requirements level:

* what does the system do?

Design Level:

* what are features of the system?
* How it will gonna work?

Implementation Level :

Implement the application.

Backward Engineering (Reverse Engineering)

Principle of Alternation

Reverse + Alternation + Forward Engineering

↓

Re-engineering

Re-engineering Strategy :

(PY – Java)

① Recreate. (Change directly in codebase)

② Rework. → don't need to redo everything only update what the part needs.

③ Replace. → Higher level এ ISIS code দিয়ে main code এ Change করা।

Reengineering Variation

MID Syllabus

- Software evolution. (1.1)
- Categorization (2.1.1, 2.1.2)
- SPE taxonomy (2.3.1, 2.3.2)
- Reuse. Oriented Model (3.2, 3.3, 3.4, 3.6, 3.7, 3.9)
- Re-engineering Concepts (4.2, 4.3)
- Maintainability Measurement, index.

[Theory, Analytical, Justification]

① ② ③

marks - 20.

Reverse Engineering Labs

- Decompilation — find tools
- Translation — others lang to Java

[.jar → .java]

(Done 12/11/24)