UNIT–5 : Software Maintenance & Project Management (PYQ Solution)

AKTU · Software Engineering (BCS601) · Solutions by **Chirag Jain**, Ycotes

SECTION A Answers (≈70 Words Each)

Q1. What is Software Reverse Engineering and Re-engineering? (May 2026, May 2025)

Reverse engineering is analysing an existing system, often with no documentation, to recover its design, specification or higher-level representation from its code. Software re-engineering (restructuring) improves or transforms an existing system by reverse engineering it, restructuring the code and data, and then forward engineering it into a new, better form. It makes old systems more maintainable and cheaper than writing them from scratch.

Q2. Define adaptive maintenance. (2023)

Adaptive maintenance is the modification of software so that it keeps working when its environment changes, such as a new operating system, hardware, database version, tax rule or law. No new functionality is added for the user; the software is only adapted to the new conditions. It accounts for roughly 20-25% of maintenance effort.

SECTION B Answers (≈250 Words Each)

Q1. Discuss various problems during maintenance. Describe some solutions to these problems. (May 2026)

Software maintenance is modifying software after delivery to correct faults, improve performance or adapt it to a changed environment. It is difficult and expensive (60-70% of total life cycle cost).

Problems

- **Poor documentation** – missing or outdated design and code documents make the system hard to understand.
- **Poor code structure** – "spaghetti code", high coupling, no standards.
- **Staff turnover** – the original developers leave, taking knowledge with them; maintenance staff are often less experienced and less motivated.
- **Ripple effect** – a change in one place causes errors elsewhere.
- **Unstructured, ageing systems** – obsolete languages, hardware and tools.
- **Lack of testing support** – regression test cases are not available.
- **Changing user demands** and pressure for quick fixes.
- **Poor configuration management** – difficulty tracking versions and changes.
- **Maintenance cost estimation** is hard and not valued by management.

Solutions

- Prepare **complete, updated documentation** during development and keep it current.
- Follow **coding standards and modular design** (low coupling, high cohesion).
- Use **reverse engineering and re-engineering** or **restructuring** for old code.
- Apply **configuration and change management** with version control.
- Maintain **regression test suites** and use automated tests.
- Do **impact analysis** before every change.
- Use **CASE tools**, train the staff and give maintenance a proper status.
- Plan **preventive maintenance** and replace the system when it is too costly to maintain.

Problem	Solution
No documentation	Reverse engineering, documentation standards
Ripple effect	Impact analysis, regression tests
Staff turnover	Knowledge transfer, training
Old technology	Re-engineering, migration

Conclusion: most maintenance problems originate in development, so good design, documentation and configuration control reduce maintenance effort.

Q2. Justify the statement "Software Maintenance is Inevitable". Discuss various types of software maintenances along with their relative maintenance cost. (May 2025)

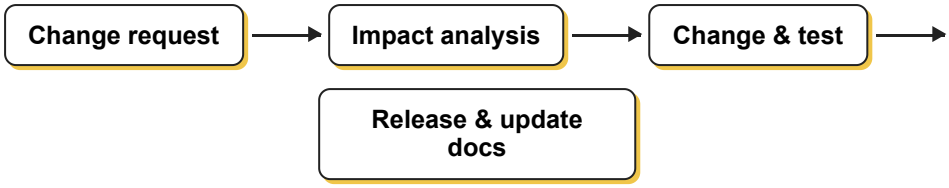
Why maintenance is inevitable

- Even well-tested software contains **latent defects** that appear only in use.
- The **user's needs** and business rules change, so new features are asked for.
- The **environment** (operating system, hardware, laws, databases) changes.
- Software is also **evolving**: as it is used, it must be changed to remain useful (Lehman's laws).
- Hence software can never be called "finished"; it ends only when it is retired.

Types of maintenance

- **Corrective** – fixing faults found after delivery (about 20%).
- **Adaptive** – changing the software to suit a new environment (about 20-25%).
- **Perfective** – adding new features or improving performance on user request (about 50-60%), the largest part.
- **Preventive** – restructuring or documenting to prevent future problems and improve maintainability (about 5%).

Type	Purpose	Approx. cost share
Corrective	Fix bugs	20%
Adaptive	New environment	20-25%
Perfective	New features / improvement	50-60%
Preventive	Future-proofing	~5%



Maintenance process

Overall, maintenance consumes **60-70% of the total cost** of the software life cycle, and the cost grows with the age and complexity of the system.

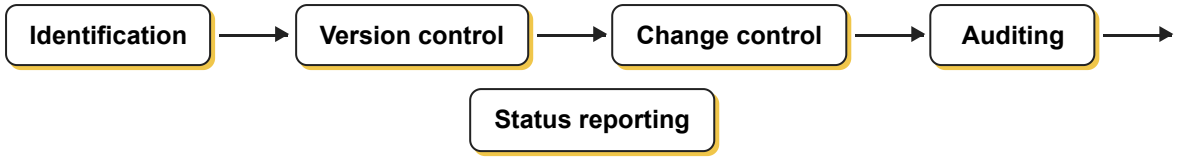
Conclusion: because of defects, changing needs and changing environments, maintenance is unavoidable, and most of its effort goes into perfective changes.

Q3. Discuss Software Configuration Management and various tasks in SCM process. (2023)

Software Configuration Management (SCM) is a set of activities for **identifying, controlling, auditing and reporting changes** in the software and its work products throughout the life cycle. Its aim is to maintain integrity and traceability while changes are made. A **software configuration item (SCI)** is any controlled work product such as source code, design documents, test cases or manuals.

SCM tasks

1. **Configuration identification** – define and name the SCIs and set baselines (a baseline is an approved version that changes only through formal control).
2. **Version control** – manage the different versions of each SCI with check-in/check-out, branching and merging (e.g. Git).
3. **Change control** – a formal procedure: change request → evaluation and impact analysis → approval by the change control board (CCB) → implementation → verification.
4. **Configuration auditing** – verify that the change was made correctly (formal technical review and configuration audit).
5. **Configuration status reporting** – record and report what changed, when, by whom and what else is affected.
6. **Build and release management** – assemble the correct versions into a build and release it.



SCM process tasks

Benefits: controlled change, fewer conflicts among developers, repeatable builds, traceability and easy rollback.

Conclusion: SCM keeps the evolving software consistent and controlled, and it is an umbrella activity applied throughout the project.

Q4. Explain version control and various types of project risks. (2023)

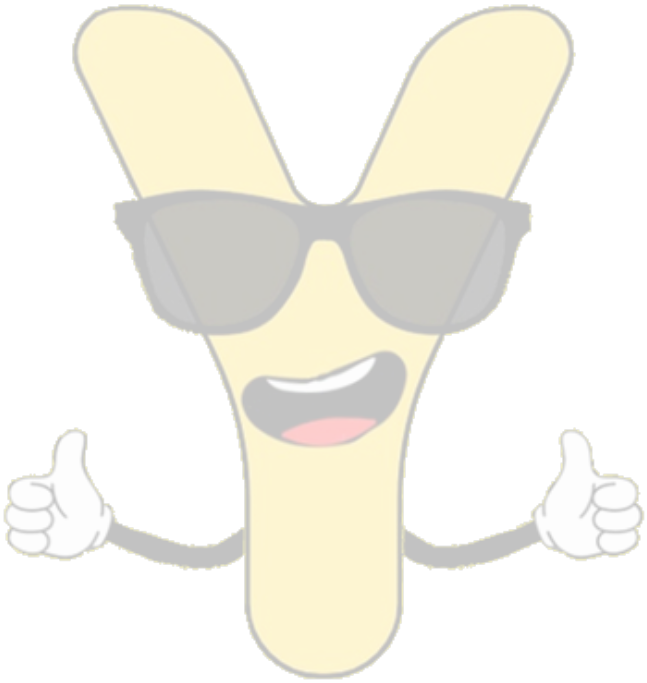
Version control is the SCM mechanism that stores and manages every version of the software items so that changes can be tracked, compared and reversed. Features: a **repository**, **check-in/check-out** with locking or merging, **branching and merging** for parallel work, **tags/baselines** for releases, and history of who changed what. Systems are **local** (RCS), **centralised** (SVN) and **distributed** (Git). It lets teams work in parallel, restores earlier versions and identifies the origin of defects.

Types of project risks

- **Project risks** – threaten the plan: budget, schedule, staffing, resources, customer and requirements problems, e.g. staff leaving or unclear requirements.
- **Technical risks** – threaten quality and timeliness: design, implementation, interface, verification and maintenance problems, new unproven technology.
- **Business risks** – threaten viability: **market risk** (no one buys the product), **strategic risk** (does not fit company strategy), **sales risk**, **management risk** (loss of support) and **budget risk**.
- **Known, predictable and unpredictable risks** – known ones come from careful evaluation, predictable ones from past experience (staff turnover), and unpredictable ones are hard to foresee.

Risk type	Example
Project	Schedule slip, staff loss
Technical	Complex design, new technology
Business	Product loses market

Conclusion: version control keeps change history safe, and identifying the project, technical and business risks early allows them to be managed through risk planning.



SECTION C Answers (≈500 Words Each)

Q1. Write a brief description of various categories of software maintenances. Give a proportionate distribution of maintenance cost to these categories. Discuss the role of Reverse Engineering in software maintenance. (May 2025)

Software maintenance is the modification of a software product after delivery to correct faults, improve performance or adapt it to a changed environment. It is the longest phase of the life cycle and takes **60-70% of total cost**.

Categories of maintenance

- **Corrective maintenance** – finding and fixing errors (bugs) reported by the users after release, e.g. correcting a wrong tax calculation.
- **Adaptive maintenance** – changing the software to work in a changed environment: new operating system, hardware, database, rules or laws.
- **Perfective maintenance** – adding new features or improving performance and usability at the user's request, e.g. a new report or faster search. The largest category.
- **Preventive maintenance** – restructuring, optimising and documenting the code so that future maintenance is easier and failures are avoided.

Distribution of maintenance cost (approximate)

Category	Share of maintenance effort
Perfective	50-60%
Adaptive	20-25%
Corrective	about 20%
Preventive	about 5%



Share of maintenance effort

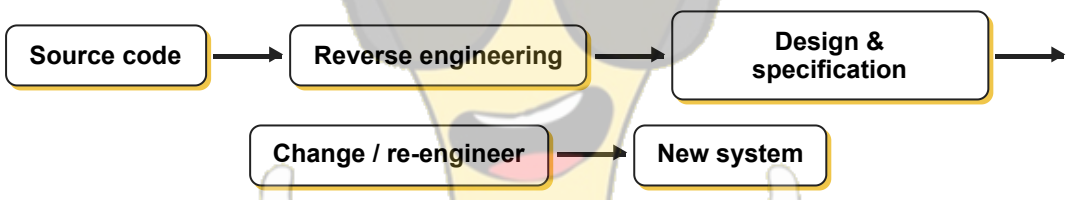
So only about one fifth of the maintenance effort fixes errors; most is enhancement and adaptation.

Role of Reverse Engineering in maintenance

Reverse engineering is the process of analysing the existing code to recover its design, specification and documentation without the original documents.

- Many old systems have **no documents or outdated ones**; reverse engineering regenerates the design so that maintainers understand the system.
- **Steps:** collect the source code → improve readability (restructure, rename, indent) → extract data and processing information → build a **structure chart, data flow diagram, ER diagram** and the control-flow model → derive the specification and write documents.
- It supports **impact analysis** of a change, so that the ripple effect is estimated.
- It is the first step of **re-engineering** and **migration** to new technology.
- It helps to find reusable components and to detect hidden faults and security risks.
- Tools: code analysers, cross-reference and call-graph generators, CASE reverse-engineering tools.

Levels: implementation level (code), structure level (modules), functional level (what the program does) and domain level (the business rules).



Reverse engineering in maintenance

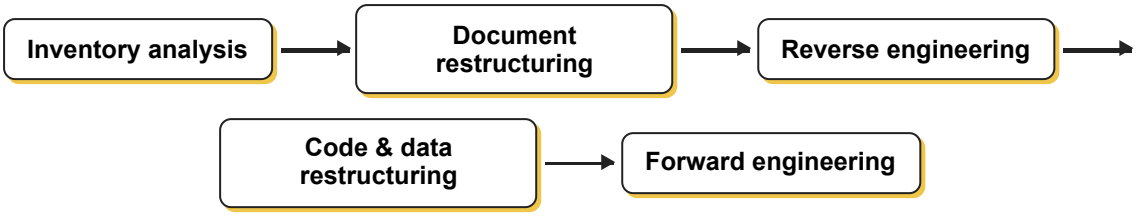
Limits: it is time-consuming, tool-dependent and cannot recover the original intent fully.

Conclusion: maintenance has four categories, with perfective the largest, and reverse engineering restores the understanding that the maintainers need to change old, undocumented software safely.

Q2. Draw the general model of software re-engineering and elaborate it. (2023)

Software re-engineering is the examination, analysis and restructuring of an existing system to reconstitute it in a new, improved form. It is applied to old (legacy) systems that are costly to maintain but still valuable. It is cheaper and less risky than writing a new system.

General model (re-engineering process)



Software re-engineering process model (a cycle of activities)

Activities

1. **Inventory analysis** – list all applications with details such as age, size, business criticality, maintenance cost and current usability; candidates for re-engineering are the important but badly maintainable ones.
2. **Document restructuring** – create or update documentation; options: leave as is, document only the critical parts, or document the whole system.
3. **Reverse engineering** – analyse the code to extract the design, data and procedural information and specifications.
4. **Code restructuring** – re-write spaghetti code into a structured form without changing behaviour; the code is then reviewed and tested.
5. **Data restructuring** – analyse and redesign data architecture and data structures, e.g. move flat files to a relational database.
6. **Forward engineering** – use the recovered information to redesign and rebuild the system with modern technology, adding new functions and better performance.

Cost-benefit: re-engineering is attractive when maintenance cost is high and the application is business-critical, as shown by the comparison below.

Option	When
Keep maintaining	System is stable and cheap to maintain
Re-engineer	Valuable but poorly structured system
Replace	System obsolete and beyond saving

Benefits: reduced maintenance cost, improved quality and understanding, reuse, migration to new platforms and lower risk than a rewrite. **Problems:** limited by the quality of the old system; needs skilled people and tools.

Conclusion: the re-engineering model passes from inventory analysis to forward engineering through the activities of reverse engineering and restructuring, and renews a legacy system at lower cost than building from scratch.

