Introduction to CASE

CASE stands for Computer-Aided Software Engineering. It refers to the use of software tools to help in the development, testing, and maintenance of software. These tools make the software development process faster, more efficient, and less error-prone.

Scope of CASE

CASE tools are used in different phases of software development – from planning and designing to coding, testing, and maintenance. They help teams work better together and improve the quality of the final product.

5 Key Points

1. Automation – CASE tools help automate tasks like coding, testing, and documentation.
2. Consistency – They ensure that software is developed in a uniform and standard way.
3. Productivity – Developers can work faster and with fewer errors.
4. Teamwork – These tools support collaboration among team members.
5. Quality – CASE tools help create reliable and high-quality software.

Introduction to CASE Tools

CASE is **Computer Aided Software Engineering**

There are lots of automated tools to assist in software engineering.

Their purpose is to make the work of software development and maintenance easier and more reliable.

"**CASE tools are computerized software development tools that support the developer when performingone or more phases of the software life cycle and/or support software maintenance.**"

Categories of Case Tools:Classic CASE tools:

Interactive debuggers, compilers, project progress control systems

Upper CASE:

support information planning, project identification and selection, projectinitiation and planning, analysis and design

Lower CASE:

support the implementation and maintenance phases of the systems developmentlife cycle

Cross life-cycle CASE:

support activities that occur across multiple phases of the systemsdevelopment life cycle

CASE Tool Types and their Support/Uses:

Diagram tools

These tools are used to represent system components, data and control flow among various softwarecomponents and system structure in a graphical form. For example, Flow Chart Maker tool for creating state-of-the-art flowcharts.

Process Modeling Tools

Process modeling is method to create software process model, which is used to develop the software.Process modeling tools help the managers to choose a process model or modify it as per the requirement ofsoftware product. For example, EPF Composer

Project Management Tools

These tools are used for project planning, cost and effort estimation, project scheduling and resource planning. Managers have to strictly comply project execution with every mentioned ste p in software projectmanagement. Project management tools help in storing and sharing project information in real-time throughoutthe organization. For example, Creative Pro Office, Trac Project, Basecamp.

Documentation Tools

Documentation in a software project starts prior to the software process, goes throughout all phases ofSDLC and after the completion of the project.Documentation tools generate documents for technical users and end users. Technical users are mostlyin-house professionals of the development team who refer to system manual, reference manual, trainingmanual, installation manuals etc. The end user documents describe the functioning and how-to of the systemsuch as user manual. For example, Doxygen, DrExplain, Adobe RoboHelp for documentation.

Analysis Tools

These tools help to gather requirements, automatically check for any inconsistency, inaccuracy in thediagrams, data redundancies or erroneous omissions. For example, Accept 360, Accompa, CaseComplete forrequirement analysis, Visible Analyst for total analysis.

Design Tools

These tools help software designers to design the block structure of the software, which may further be broken down in smaller modules using refinement techniques. These tools provides

detailing of each moduleand interconnections among modules. For example, Animated Software Design.

Configuration Management Tools

An instance of software is released under one version. Configuration Management tools deal with Version and revision management.

Baseline configuration management.

Change control managementCASE tools help in this by automatic tracking, version management and release management. For example,Fossil, Git, Accu REV.

Change Control Tools

These tools are considered as a part of configuration management tools. They deal with changes madeto the software after its baseline is fixed or when the software is first released. CASE tools automate changetracking, file management, code management and more. It also helps in enforcing change policy of theorganization.

Programming Tools

These tools consist of programming environments like IDE (Integrated Development Environment),
in- built modules library and simulation tools. These tools provide comprehensive aid in building software product and include features for simulation and testing. For example, Cscope to search code in C, Eclipse.

Prototyping Tools

Software prototype is simulated version of the intended software product. Prototype provides initiallook and feel of the product and simulates few aspect of actual product.Prototyping CASE tools essentially come with graphical libraries. They can create hardwareindependent user interfaces and design. These tools help us to build rapid prototypes based on existinginformation. In addition, they provide simulation of software prototype. For example, Serena prototypecomposer, Mockup Builder.

Web Development Tools

These tools assist in designing web pages with all allied elements like forms, text, script, graphic and soon. Web tools also provide live preview of what is being developed and how will it look after completion. Forexample, Fontello, Adobe Edge Inspect, Foundation 3, Brackets.

Quality Assurance Tools

Quality assurance in a software organization is monitoring the engineering process and methodsadopted to develop the software product in order to ensure conformance of quality as per organizationstandards. QA tools consist of configuration and change control tools and software testing tools. For example,SoapTest, AppsWatch, JMeter.

Maintenance Tools

Software maintenance includes modifications in the software product after it is delivered. Automaticlogging and error reporting techniques, automatic error ticket generation and root cause Analysis are fewCASE tools, which help software organization in maintenance phase of SDLC. For example, Bugzilla fordefect tracking, HP Quality Center.

CASE Environment

A **CASE environment** is the full setup or platform where different CASE tools work together. It supports all steps of software development, like planning, designing, coding, testing, and maintaining the software.

Key Points

1. **Tool Integration** – Different CASE tools work together in one place.
2. **Supports Full Development** – Helps in all stages of making software.
3. **Data Sharing** – Tools can share and use the same data easily.
4. **User-Friendly** – Makes work easier for developers with helpful interfaces.
5. **Improves Workflow** – Speeds up tasks and reduces mistakes.

CASE Support in Software Life Cycle

CASE tools help at every stage of the **Software Development Life Cycle (SDLC)**. They make it easier to plan, build, test, and maintain software.

Key Points

1. **Planning Help** – Tools support project planning and requirement gathering.
2. **Design Support** – Help in creating models and system designs.
3. **Coding Help** – Assist in writing and generating code.
4. **Testing Support** – Find and fix bugs quickly.
5. **Maintenance Help** – Make updates and changes easier after software is released.

Other Characteristics of CASE Tools

Besides helping in software development, CASE tools have extra features that make work easier and better.

Key Points

1. **Documentation Support** – They help create and manage project documents.
2. **Error Checking** – They can find mistakes early in the process.
3. **Reusability** – Allow reuse of code or design parts in other projects.
4. **Standardization** – Help follow set rules and formats.
5. **Project Tracking** – Keep track of progress and tasks.

   Towards Second Generation CASE Tools

   Second generation CASE tools are improved versions of the first ones. They are smarter, faster, and better at working together. These tools focus more on teamwork, flexibility, and full project support.

   Key Points:

1. Better Integration – Tools work together more smoothly.
2. User-Friendly – Easier to use with better interfaces.
3. Supports Teamwork – Helps many developers work together.
4. Covers Full Project – Supports all stages of software development.
5. More Flexible – Can be used in different types of projects.

   Architecture of CASE Environment

   The **CASE (Computer-Aided Software Engineering) environment architecture** is the structure that connects all tools, data, and users in the software development process. It helps teams work better by organizing how tools and information are used and shared.

   Main Components:

1. **Tool Set**
   - A group of software tools for different tasks like designing, coding, testing, and documentation.
2. **Central Repository**

- A shared database where all project data, models, and documents are stored. This ensures consistency and easy access for all users.

3. **User Interface**
    - A friendly and easy way for developers to use the tools and view project information.
4. **Integration Framework**
    - This layer allows all tools to work together, share data, and stay updated in real-time.
5. **Management Tools**
    - Used for project planning, tracking progress, assigning tasks, and handling reports.
6. **Communication Support**
    - Helps team members share ideas, updates, and files easily.
7. **Security and Access Control**
    - Protects the data and controls who can view or change certain parts of the project.
8. **Standardization Support**
    - Ensures that all work follows common methods, formats, and coding standards.
9. **Scalability**
    - The system can grow and handle both small and large software projects.
10. **Support for Entire Software Life Cycle**

From planning to maintenance, the architecture supports every phase of software development.

Why it :

- Improves teamwork and communication
- Reduces errors and duplication
- Saves time and increases productivity
- Helps deliver high-quality software

Software Maintenance Process Models

Software maintenance is the process of changing, updating, or improving software after it has been delivered to the user. To manage these changes effectively, different **process models** are used. These models provide a step-by-step method for handling updates, fixing bugs, and adding new features.

Common Software Maintenance Process Models

1. **Quick Fix Model**
    - Focuses on fixing problems quickly without deep analysis.
    - Useful for urgent bug fixes, but may affect long-term quality.
2. **Iterative Enhancement Model**
    - Changes are made step-by-step in small updates.
    - Each version improves the system gradually.
    - Good for adding features over time.
3. **Reuse-Oriented Model**
    - Tries to reuse existing components or code to save time and effort.
    - Focuses on improving efficiency and reducing cost.
4. **Boehm's Model (or Spiral Model for Maintenance)**
    - Uses a risk-based approach.
    - Each cycle includes planning, risk analysis, engineering, and evaluation.
    - Suitable for complex and long-term maintenance.
5. **Maintenance as Part of the Full SDLC**
    - Treats maintenance as a regular part of the Software Development Life Cycle.
    - Follows stages like requirement analysis, design, implementation, testing, and delivery for every change.

Key Goals of These Models

- Improve software performance
- Fix errors and bugs
- Add new features or functions
- Make the system work with new hardware or software

Estimating Maintenance Costs: Key Points

1. **Identify Maintenance Needs**
    - Determine whether the maintenance is corrective (bug fixes), adaptive (updates), perfective (new features), or preventive (avoiding future issues).
2. **Assess Software Complexity**
    - Complex systems with more code, dependencies, or integrations will cost more to maintain.
3. **Estimate Required Resources**
    - Calculate the number of people (developers, testers) and tools needed to perform the maintenance tasks.
4. **Evaluate Time and Effort**

    - Estimate how long each maintenance task will take, including testing and validation.
5. **Account for Labor Costs**
    - Estimate the cost of developer hours, testers, and other personnel needed for maintenance.
6. **Consider Operational Costs**
    - Include costs for infrastructure (servers, cloud services), tools, and software licenses required for maintenance.
7. **Factor in Testing Costs**
    - Maintenance often involves testing to ensure changes don't cause issues elsewhere in the software.
8. **Monitor Long-Term Impact**
    - Efficient maintenance can reduce long-term costs by improving software performance and stability.
9. **Track Historical Data**
    - Use data from past maintenance activities to predict future costs more accurately.
10. **Account for Unpredictable Changes**

- Unexpected issues (like new bugs or system failures) can increase maintenance costs.

These points give you a clear framework for estimating the costs involved in maintaining software.

What is software reuse?

Software reuse is a term used for developing the software by using the existing software components. Some of the components that can be reuse are as follows;

- Source code
- Design and interfaces
- User manuals
- Software Documentation
- Software requirement specifications and many more.

Introduction **Software reuse**

Software reuse is the practice of using existing software components, code, or modules to build new applications. It helps developers save time and effort by not having to create everything from scratch. Reusing software can reduce costs and improve quality since the reused components are often tested and reliable. This approach promotes efficiency in software development. Ultimately, software reuse leads to faster development cycles and more consistent software performance.

What are the advantages of software reuse?

- **Less effort:** Software reuse requires less effort because many components use in the system are ready made components.
- **Time-saving:** Re-using the ready made components is time saving for the software team.
- **Reduce cost:** Less effort, and time saving leads to the overall cost reduction.
- **Increase software productivity:** when you are provided with ready made components, then you can focus on the new components that are not available just like ready made components.
- **Utilize fewer resources:** Software reuse save many sources just like effort, time, money etc.
- **Leads to a better quality software:** Software reuse save our time and we can consume our more time on maintaining software quality and assurance.

What are stages of reuse-oriented software engineering?

Requirement specification: First of all, specify the requirements. This will help to decide that we have some existing software components for the development of software or not. **Component analysis** Helps to decide that which component can be reused where. **Requirement updations / modifications.** If the requirements are changed by the customer, then still existing components are helpful for reuse or not. **Reuse System design** If the requirements are changed by the customer, then still existing system designs are helpful for reuse or not. **Development** Existing components are matching with new software or not. **Integration** Can we integrate the new systems with existing components? **System validation** To validate the system that it can be accepted by the customer or not.

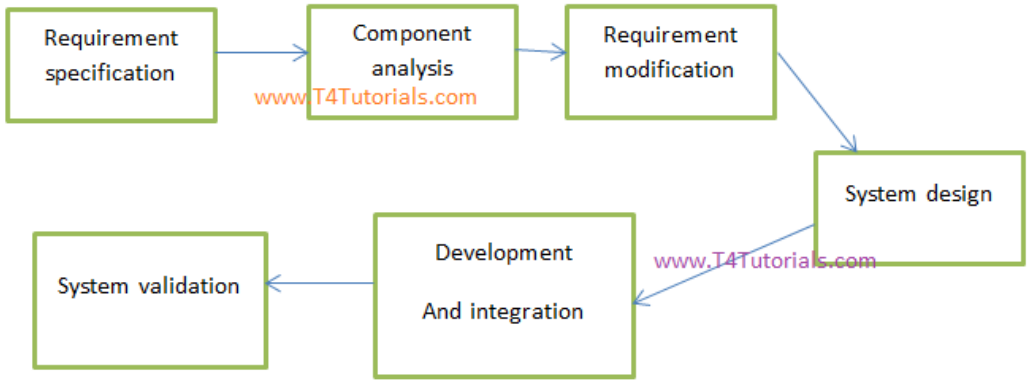

Figure: reuse-oriented software engineering

software reuse success factors

1. Capturing Domain Variations
2. Easing Integration
3. Understanding Design Context
4. Effective Teamwork
5. Managing Domain Complexity

the **reasons behind no software reuse so far**:

Lack of Standardization – There are no consistent guidelines or standards for creating reusable software components.

Compatibility Issues – Reusable software may not always be compatible with different systems or environments.

Inadequate Documentation – Poor or missing documentation makes it hard to understand and reuse existing code effectively.

High Initial Effort – Developing reusable components often requires significant effort, which can be seen as time-consuming.

Customization Needs – Reused software may need to be heavily customized to fit new projects, reducing its practical use.

Unclear Benefits – Organizations may not see immediate value in reusing software and prefer building new solutions from scratch.

Fragmented Software – Many software components are isolated, making it difficult to reuse them in other contexts.

These challenges have limited the widespread adoption of software reuse.