**SOFTWARE ENGINEER**

**Software User Interface Design**

**User interface** is part of software and is designed such a way that it is expected to provide the user insight of the software. UI provides fundamental platform for human-computer interaction.

**UI** can be **graphical, text-based, audio-video based**, depending upon the underlying **hardware and software combination**. UI can be hardware or software or a combination of both.

The software becomes more popular if its **user interface** is:

* Attractive
* Simple to use
* Responsive in short time
* Clear to understand
* Consistent on all interfacing screens

**UI** is broadly divided into **two** categories:

* Command Line Interface
* Graphical User Interface

**Command Line Interface (CLI)**

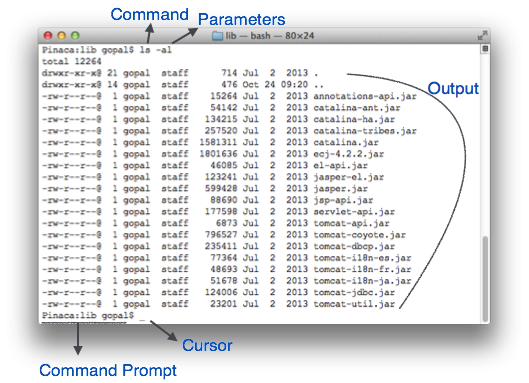
CLI has been a great tool of interaction with computers until the video display monitors came into existence. CLI is first choice of many technical users and programmers. CLI is minimum interface a software can provide to its users.

CLI provides a command prompt, the place where the user types the command and feeds to the system. The user needs to remember the syntax of command and its use. Earlier CLI were not programmed to handle the user errors effectively.

A command is a text-based reference to set of instructions, which are expected to be executed by the system. There are methods like macros, scripts that make it easy for the user to operate.

CLI uses less amount of computer resource as compared to GUI.

**CLI Elements**



A text-based command line interface can have the following elements:

**Command Prompt** - It is text-based notifier that is mostly shows the context in which the user is working. It is generated by the software system.

**Cursor -** It is a small horizontal line or a vertical bar of the height of line, to represent position of character while typing. Cursor is mostly found in blinking state. It moves as the user writes or deletes something.

**Command -** A command is an executable instruction. It may have one or more parameters. Output on command execution is shown inline on the screen. When output is produced, command prompt is displayed on the next line.

**Graphical User Interface**

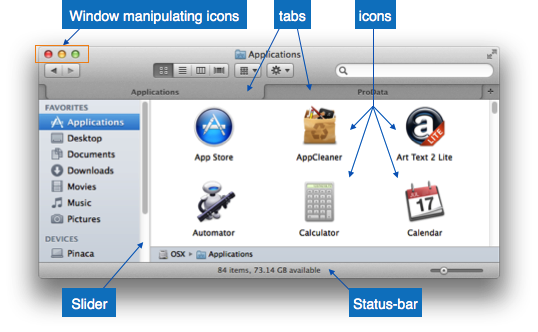
Graphical User Interface provides the user graphical means to interact with the system. GUI can be combination of both hardware and software. Using GUI, user interprets the software.

Typically, GUI is more resource consuming than that of CLI. With advancing technology, the programmers and designers create complex GUI designs that work with more efficiency, accuracy and speed.

**GUI Elements**

GUI provides a set of components to interact with software or hardware.

Every graphical component provides a way to work with the system. A GUI system has following elements such as:



**Window -** An area where contents of application are displayed. Contents in a window can be displayed in the form of icons or lists, if the window represents file structure. It is easier for a user to navigate in the file system in an exploring window. Windows can be minimized, resized or maximized to the size of screen. They can be moved anywhere on the screen. A window may contain another window of the same application, called child window.

**Tabs -** If an application allows executing multiple instances of itself, they appear on the screen as separate windows. Tabbed Document Interface has come up to open multiple documents in the same window. This interface also helps in viewing preference panel in application. All modern web-browsers use this feature.

**Menu -** Menu is an array of standard commands, grouped together and placed at a visible place (usually top) inside the application window. The menu can be programmed to appear or hide on mouse clicks.

**Icon -** An icon is small picture representing an associated application. When these icons are clicked or double clicked, the application window is opened. Icon displays application and programs installed on a system in the form of small pictures.

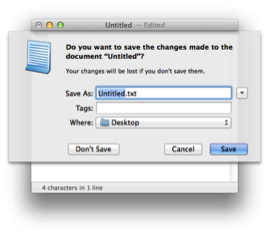
**Cursor -** Interacting devices such as mouse, touch pad, digital pen is represented in GUI as cursors. On screen cursor follows the instructions from hardware in almost real-time. Cursors are also named pointers in GUI systems. They are used to select menus, windows and other application features.

**Application specific GUI components**

A GUI of an application contains one or more of the listed **GUI** elements:

**Application Window -** Most application windows use the constructs supplied by operating systems but many use their own customer created windows to contain the contents of application.

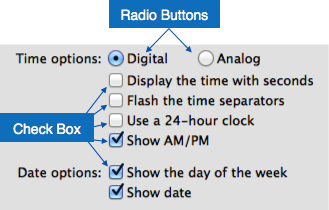
**Dialogue Box** - It is a child window that contains message for the user and request for some action to be taken. For Example: Application generate a dialogue to get confirmation from user to delete a file.



**Dialogue Box**

**Text-Box** - Provides an area for user to type and enter text-based data.

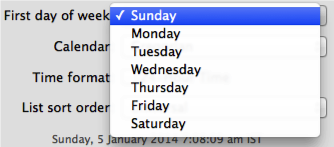
**Buttons** - They imitate real life buttons and are used to submit inputs to the software.



**Radio-button** - Displays available options for selection. Only one can be selected among all offered.

**Check-box** - Functions similar to list-box. When an option is selected, the box is marked as checked. Multiple options represented by check boxes can be selected.

**List-box** - Provides list of available items for selection. More than one item can be selected.



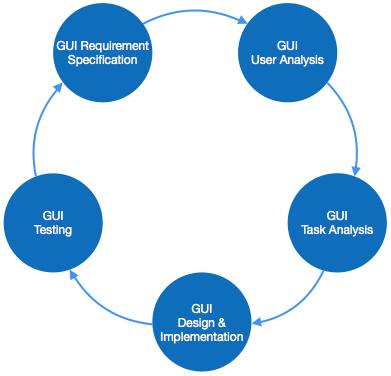
Other impressive **GUI** components are:

* Sliders
* Combo-box
* Data-grid
* Drop-down list

**User Interface Design Activities**

There are a number of activities performed for designing user interface. The process of GUI design and implementation is alike SDLC. Any model can be used for GUI implementation among Waterfall, Iterative or Spiral Model.

A model used for GUI design and development should fulfill these GUI specific steps.



**GUI Requirement Gathering** - The designers may like to have list of all functional and non-functional requirements of GUI. This can be taken from user and their existing software solution.

**User Analysis** - The designer studies who is going to use the software GUI. The target audience matters as the design details change according to the knowledge and competency level of the user. If user is technical savvy, advanced and complex GUI can be incorporated. For a novice user, more information is included on how-to of software.

**Task Analysis** - Designers have to analyze what task is to be done by the software solution. Here in GUI, it does not matter how it will be done. Tasks can be represented in hierarchical manner taking one major task and dividing it further into smaller sub-tasks. Tasks provide goals for GUI presentation. Flow of information among sub-tasks determines the flow of GUI contents in the software.

**GUI Design & implementation** - Designers after having information about requirements, tasks and user environment, design the GUI and implements into code and embed the GUI with working or dummy software in the background. It is then self-tested by the developers.

**Testing -** GUI testing can be done in various ways. Organization can have in-house inspection, direct involvement of users and release of beta version are few of them. Testing may include usability, compatibility, user acceptance etc.

**GUI Implementation Tools**

There are several tools available using which the designers can create entire GUI on a mouse click. Some tools can be embedded into the software environment (IDE).

GUI implementation tools provide powerful array of GUI controls. For software customization, designers can change the code accordingly.

There are different segments of GUI tools according to their different use and platform.

**Example**

Mobile GUI, Computer GUI, Touch-Screen GUI etc. Here is a list of few tools which come handy to build GUI:

* FLUID
* AppInventor (Android)
* LucidChart
* Wavemaker
* Visual Studio

**User Interface Golden rules**

The following rules are mentioned to be the golden rules for GUI design, described by Shneiderman and Plaisant in their book (Designing the User Interface).

**Strive for consistency -** Consistent sequences of actions should be required in similar situations. Identical terminology should be used in prompts, menus, and help screens. Consistent commands should be employed throughout.

**Enable frequent users to use short**-**cuts** - The user’s desire to reduce the number of interactions increases with the frequency of use. Abbreviations, function keys, hidden commands, and macro facilities are very helpful to an expert user.

**Offer informative feedback** - For every operator action, there should be some system feedback. For frequent and minor actions, the response must be modest, while for infrequent and major actions, the response must be more substantial.

**Design dialog to yield closure** - Sequences of actions should be organized into groups with a beginning, middle, and end. The informative feedback at the completion of a group of actions gives the operators the satisfaction of accomplishment, a sense of relief, the signal to drop contingency plans and options from their minds, and this indicates that the way ahead is clear to prepare for the next group of actions.

**Offer simple error handling** - As much as possible, design the system so the user will not make a serious error. If an error is made, the system should be able to detect it and offer simple, comprehensible mechanisms for handling the error.

**Permit easy reversal of actions** - This feature relieves anxiety, since the user knows that errors can be undone. Easy reversal of actions encourages exploration of unfamiliar options. The units of reversibility may be a single action, a data entry, or a complete group of actions.

**Support internal locus of control -** Experienced operators strongly desire the sense that they are in charge of the system and that the system responds to their actions. Design the system to make users the initiators of actions rather than the responders.

**Reduce short-term memory load** - The limitation of human information processing in short-term memory requires the displays to be kept simple, multiple page displays be consolidated, window-motion frequency be reduced, and sufficient training time be allotted for codes, mnemonics, and sequences of actions.

**CASE** stands for Computer Aided Software Engineering. It means, development and maintenance of software projects with help of various automated software tools.

**CASE Tools**

CASE tools are set of software application programs, which are used to automate SDLC activities. CASE tools are used by software project managers, analysts and engineers to develop software system.

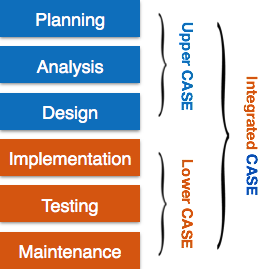
There are number of CASE tools available to simplify various stages of Software Development Life Cycle such as Analysis tools, Design tools, Project management tools, Database Management tools, Documentation tools are to name a few.

Use of CASE tools accelerates the development of project to produce desired result and helps to uncover flaws before moving ahead with next stage in software development.

**Components of CASE Tools**

CASE tools can be broadly divided into the following parts based on their use at a particular SDLC stage:

**Central Repository** - CASE tools require a central repository, which can serve as a source of common, integrated and consistent information. Central repository is a central place of storage where product specifications, requirement documents, related reports and diagrams, other useful information regarding management is stored. Central repository also serves as data dictionary.



**Upper Case Tools** - Upper CASE tools are used in planning, analysis and design stages of SDLC.

**Lower Case Tools** - Lower CASE tools are used in implementation, testing and maintenance.

**Integrated Case Tools** - Integrated CASE tools are helpful in all the stages of SDLC, from Requirement gathering to Testing and documentation.

CASE tools can be grouped together if they have similar functionality, process activities and capability of getting integrated with other tools.

**Scope of Case Tools**

The scope of CASE tools goes throughout the SDLC.

**Case Tools Types**

Now we briefly go through various CASE tools

**Diagram tools**

These tools are used to represent system components, data and control flow among various software components 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 of software 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 step in software project management. Project management tools help in storing and sharing project information in real-time throughout the 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 of SDLC and after the completion of the project.

Documentation tools generate documents for technical users and end users. Technical users are mostly in-house professionals of the development team who refer to system manual, reference manual, training manual, installation manuals etc. The end user documents describe the functioning and how-to of the system such 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 the diagrams, data redundancies or erroneous omissions. For example, Accept 360, Accompa, CaseComplete for requirement 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 provide detailing of each module and 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 management

CASE 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 made to the software after its baseline is fixed or when the software is first released. CASE tools automate change tracking, file management, code management and more. It also helps in enforcing change policy of the organization.

**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 initial look and feel of the product and simulates few aspect of actual product.

Prototyping CASE tools essentially come with graphical libraries. They can create hardware independent user interfaces and design. These tools help us to build rapid prototypes based on existing information. In addition, they provide simulation of software prototype. For example, Serena prototype composer, Mockup Builder.

**Web Development Tools**

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

**Quality Assurance Tools**

Quality assurance in a software organization is monitoring the engineering process and methods adopted to develop the software product in order to ensure conformance of quality as per organization standards. 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. Automatic logging and error reporting techniques, automatic error ticket generation and root because Analysis are few CASE tools, which help software organization in maintenance phase of SDLC. For example, Bugzilla for defect tracking, HP Quality Center.

Q.What is computer software?

A. Computer software is a complete package, which includes software program, its documentation and user guide on how to use the software.

Q.Can you differentiate computer software and computer program?

A. A computer program is piece of programming code which performs a well defined task where as software includes programming code, its documentation and user guide.

Q.What is software engineering?

A. Software engineering is an engineering branch associated with software system development.

Q.When you know programming, what is the need to learn software engineering concepts?

A. A person who knows how to build a wall may not be good at building an entire house. Likewise, a person who can write programs may not have knowledge of other concepts of Software Engineering. The software engineering concepts guide programmers on how to assess requirements of end user, design the algorithms before actual coding starts, create programs by coding, testing the code and its documentation.

Q.What is software process or Software Development Life Cycle (SDLC)?

A.Software Development Life Cycle, or software process is the systematic development of software by following every stage in the development process namely, Requirement Gathering, System Analysis, Design, Coding, Testing, Maintenance and Documentation in that order.

Q.What are SDLC models available?

A. There are several SDLC models available such as Waterfall Model, Iterative Model, Spiral model, V-model and Big-bang Model etc.

Q.What are various phases of SDLC?

A. The generic phases of SDLC are: Requirement Gathering, System Analysis and Design, Coding, Testing and implementation. The phases depend upon the model we choose to develop software.

Q.Which SDLC model is the best?

A. SDLC Models are adopted as per requirements of development process. It may very software-to-software to ensuring which model is suitable.

We can select the best SDLC model if following answers are satisfied -

Is SDLC suitable for selected technology to implement the software ?

Is SDLC appropriate for client’s requirements and priorities ?

Is SDLC model suitable for size and complexity of the software ?

Is the SDLC model suitable for type of projects and engineering we do ?

Is the SDLC appropriate for the geographically co-located or dispersed developers ?

Q.What is software project management?

A. Software project management is process of managing all activities like time, cost and quality management involved in software development.

Q.Who is software project manager?

A. A software project manager is a person who undertakes the responsibility of carrying out the software project.

Q.What does software project manager do?

A. Software project manager is engaged with software management activities. He is responsible for project planning, monitoring the progress, communication among stakeholders, managing risks and resources, smooth execution of development and delivering the project within time, cost and quality contraints.

Q.What is software scope?

A. Software scope is a well-defined boundary, which encompasses all the activities that are done to develop and deliver the software product.

The software scope clearly defines all functionalities and artifacts to be delivered as a part of the software. The scope identifies what the product will do and what it will not do, what the end product will contain and what it will not contain.

Q.What is project estimation?

A. It is a process to estimate various aspects of software product in order to calculate the cost of development in terms of efforts, time and resources. This estimation can be derived from past experience, by consulting experts or by using pre-defined formulas.

Q.How can we derive the size of software product?

A. Size of software product can be calculated using either of two methods -

Counting the lines of delivered code

Counting delivered function points

Q.What are function points?

A. Function points are the various features provided by the software product. It is considered as a unit of measurement for software size.

Q.What are software project estimation techniques available?

A. There are many estimation techniques available.The most widely used are -

Decomposition technique (Counting Lines of Code and Function Points)

Empirical technique (Putnam and COCOMO).

Q.What is baseline?

A. Baseline is a measurement that defines completeness of a phase. After all activities associated with a particular phase are accomplished, the phase is complete and acts as a baseline for next phase.

Q.What is Software configuration management?

A. Software Configuration management is a process of tracking and controlling the changes in software in terms of the requirements, design, functions and development of the product.

Q.What is change control?

A. Change control is function of configuration management, which ensures that all changes made to software system are consistent and made as per organizational rules and regulations.

Q.How can you measure project execution?

A. We can measure project execution by means of Activity Monitoring, Status Reports and Milestone Checklists.

Q.Mention some project management tools.

A. There are various project management tools used as per the requirements of software project and organization policies. They include Gantt Chart, PERT Chart, Resource Histogram, Critical Path Analysis, Status Reports, Milestone Checklists etc.

Q.What are software requirements?

A. Software requirements are functional description of proposed software system. Requirements are assumed to be the description of target system, its functionalities and features. Requirements convey the expectations of users from the system.

Q.What is feasibility study?

A. It is a measure to assess how practical and beneficial the software project development will be for an organization. The software analyzer conducts a thorough study to understand economic, technical and operational feasibility of the project.

Economic - Resource transportation, cost for training, cost of additional utilities and tools and overall estimation of costs and benefits of the project.

Technical - Is it possible to develop this system ? Assessing suitability of machine(s) and operating system(s) on which software will execute, existing developers’ knowledge and skills, training, utilities or tools for project.

Operational - Can the organization adjust smoothly to the changes done as per the demand of project ? Is the problem worth solving ?

Q.How can you gather requirements?

A. Requirements can be gathered from users via interviews, surveys, task analysis, brainstorming, domain analysis, prototyping, studying existing usable version of software, and by observation.

Q.What is SRS?

A. SRS or Software Requirement Specification is a document produced at the time of requirement gathering process. It can be also seen as a process of refining requirements and documenting them.

Q.What are functional requirements?

A. Functional requirements are functional features and specifications expected by users from the proposed software product.

Q.What are non-functional requirements?

A. Non-functional requirements are implicit and are related to security, performance, look and feel of user interface, interoperability, cost etc.

Q.What is software measure?

A. Software Measures can be understood as a process of quantifying and symbolizing various attributes and aspects of software.

Q.What is software metric?

A. Software Metrics provide measures for various aspects of software process and software product. They are divided into –

Requirement metrics : Length requirements, completeness

Product metrics :Lines of Code, Object oriented metrics, design and test metrics

Process metrics: Evaluate and track budget, schedule, human resource.

Q.What is modularization?

A. Modularization is a technique to divide a software system into multiple discreet modules, which are expected to carry out task(s) independently.

Q.What is concurrency and how it is achieved in software?

A. Concurrency is the tendency of events or actions to happen simultaneously. In software, when two or more processes execute simultaneously, they are called concurrent processes.

Example

While you initiate print command and printing starts, you can open a new application.

Concurrency, is implemented by splitting the software into multiple independent units of execution namely processes and threads, and executing them in parallel.

Q.What is cohesion?

A. Cohesion is a measure that defines the degree of intra-dependability among the elements of the module.

Q.What is coupling?

A. Coupling is a measure that defines the level of inter-dependability among modules of a program.

Q.Mentions some software analysis & design tools?

A. These can be: DFDs (Data Flow Diagrams), Structured Charts, Structured English, Data Dictionary, HIPO (Hierarchical Input Process Output) diagrams, ER (Entity Relationship) Diagrams and Decision tables.

Q.What is level-0 DFD?

A. Highest abstraction level DFD is known as Level 0 DFD also called a context level DFD, which depicts the entire information system as one diagram concealing all the underlying details.

Q.What is the difference between structured English and Pseudo Code?

A. Structured English is native English language used to write the structure of a program module by using programming language keywords, whereas, Pseudo Code is more close to programming language and uses native English language words or sentences to write parts of code.

Q.What is data dictionary?

A. Data dictionary is referred to as meta-data. Meaning, it is a repository of data about data. Data dictionary is used to organize the names and their references used in system such as objects and files along with their naming conventions.

Q.What is structured design?

A. Structured design is a conceptualization of problem into several well-organized elements of solution. It is concern with the solution design and based on ‘divide and conquer’ strategy.

Q.What is the difference between function oriented and object oriented design?

A. Function-oriented design is comprised of many smaller sub-systems known as functions. Each function is capable of performing significant task in the system. Object oriented design works around the real world objects (entities), their classes (categories) and methods operating on objects (functions).

Q.Briefly define top-down and bottom-up design model.

A. Top-down model starts with generalized view of system and decomposes it to more specific ones, whereas bottom-up model starts with most specific and basic components first and keeps composing the components to get higher level of abstraction.

Q.What is the basis of Halstead’s complexity measure?

A. Halstead’s complexity measure depends up on the actual implementation of the program and it considers tokens used in the program as basis of measure.

Q.Mention the formula to calculate Cyclomatic complexity of a program?

A. Cyclomatic complexity uses graph theory’s formula: V(G) = e – n + 2

Q.What is functional programming?

A. Functional programming is style of programming language, which uses the concepts of mathematical function. It provides means of computation as mathematical functions, which produces results irrespective of program state.

Q.Differentiate validation and verification?

A. Validation checks if the product is made as per user requirements whereas verification checks if proper steps are followed to develop the product.

Validation confirms the right product and verification confirms if the product is built in a right way.

Q.What is black-box and white-box testing?

A. Black-box testing checks if the desired outputs are produced when valid input values are given. It does not verify the actual implementation of the program.

White-box testing not only checks for desired and valid output when valid input is provided but also it checks if the code is implemented correctly.

Criteria Black Box Testing White Box Testing

Knowledge of software program, design and structure essential No Yes

Knowledge of Software Implementation essential No Yes

Who conducts this test on software Software Testing Employee Software Developer

baseline reference for tester Requirements specifications Design and structure details

Q.Quality assurance vs. Quality Control?

A. Quality Assurance monitors to check if proper process is followed while software developing the software.

Quality Control deals with maintaining the quality of software product.

Q.What are various types of software maintenance?

A. Maintenance types are: corrective, adaptive, perfective and preventive.

Corrective

Removing errors spotted by users

Adaptive

tackling the changes in the hardware and software environment where the software works

Perfective maintenance

implementing changes in existing or new requirements of user

Preventive maintenance

taking appropriate measures to avoid future problems

Q.What is software re-engineering?

A. Software re-engineering is process to upgrade the technology on which the software is built without changing the functionality of the software. This is done in order to keep the software tuned with the latest technology.

Q.What are CASE tools?

A. CASE stands for Computer Aided Software Engineering. CASE tools are set of automated software application programs, which are used to support, accelerate and smoothen the SDLC activities.