COURSE NAME

SOFTWARE ENGINEERING

CSC 3114

(UNDERGRADUATE)

# CHAPTER 9

# SOFTWARE DESIGN

# DESIGN

Mitch Kapor, the creator of Lotus 1-2-3, presented a "software design manifesto" in *Dr. Dobbs Journal*. He said:

- Good software design should exhibit:
  - *Firmness:* A program should not have any bugs that inhibit its function
  - *Commodity:* A program should be suitable for the purposes for which it was intended
  - *Delight:* The experience of using the program should be pleasurable one

# MODULARITY

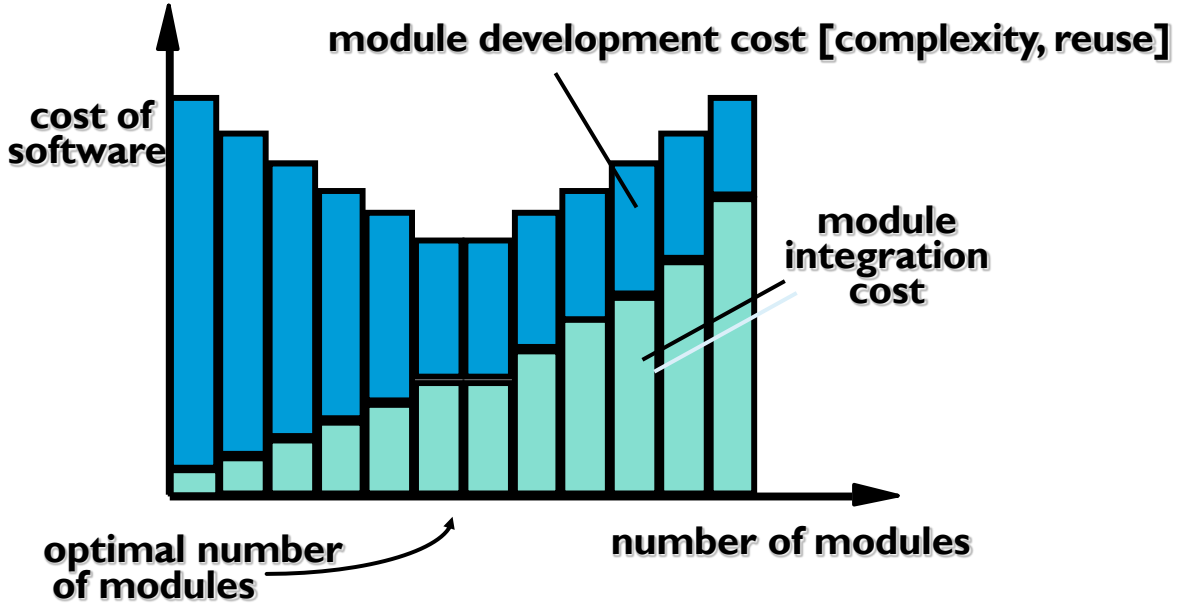
- Modularity is an attribute of software that allows a program to be intellectually manageable into distinct logical parts
- Modularity is the degree to which a system's components are logically separated into distinct parts called module and recombined again
- Monolithic software (i.e., a large program composed of a single module) cannot be easily grasped by a software engineer. The number of control paths, span of reference, number of variables, and overall complexity would make understanding close to impossible.
- In almost all instances, you should break the design into many modules, hoping to make understanding easier and as a consequence, reduce the complexity and cost required to build the software.

# FUNCTIONAL INDEPENDENCE

- Cohesion is an indication of the relative functional strength of a module. A cohesive module performs a single task, requiring little interaction with other components in other parts of a program. Stated simply, a cohesive module should (ideally) do just one thing.
- Coupling is an indication of the relative interdependence among modules. Coupling depends on the interface complexity between modules, the point at which entry or reference is made to a module, and what data pass across the interface.
- Aspect is a representation of a cross-cutting concern. Consider two requirements, *A* and *B*. Requirement *A* *crosscuts* requirement *B* "if a software decomposition [refinement] has been chosen in which *B* cannot be satisfied without taking *A* into account.
- Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code [design] yet improves its internal structure (sort algorithm)

# MODULARITY : TRADE-OFFS

*What is the "right" number of modules for a specific software design?*



# USER INTERFACE DESIGN

- Easy to learn?
- Easy to use?
- Easy to understand?

## Typical Design Errors

- **lack of consistency**
- **too much memorization**
- **no guidance / help**
- **no context sensitivity**
- **Obscure/ unfriendly**

# GOLDEN RULE - PLACE THE USER IN CONTROL

- Define interaction modes in a way that does not force a user into unnecessary or undesired actions - The user should always be able to enter and exit the mode with little or no effort.
- Provide for flexible interaction (color, font, language, etc.) - Because different users have different interaction preferences, choices should be provided by using keyboard commands, mouse movements, digitizer pen or voice recognition commands.
- Allow user interaction to be interruptible and undoable - A user should be able to interrupt a sequence of actions to do something else without losing the work that has been done. The user should always be able to "undo" any action.
- Streamline interaction as skill levels advance and allow the interaction to be customized - Allow to design a macro if the user is to perform the same sequence of actions repeatedly

# GOLDEN RULE - PLACE THE USER IN CONTROL

- Hide technical internals from the casual user - The user interface should move the user into the virtual world of the application. A user should never be required to type O/S commands from within application software.
- Design for direct interaction with objects that appear on the screen - The user feels a sense of control when able to manipulate the objects that are necessary to perform a task in a manner similar to what would occur if the object were a physical thing (progress bar).

# GOLDEN RULE – REDUCE USER'S MEMORY LOAD

- Reduce demand on short-term memory (navigation) - Provide visual cues that enable a user to recognize past actions, rather than having to recall them
- Establish meaningful defaults - A user should be able to specify individual preferences; however, a reset option should be available to enable the redefinition of original default values (balance 0.00).
- Define shortcuts that are intuitive (intuitive - having the ability to understand or know something without any direct evidence or reasoning process) - "Alt-P to print"
- The visual layout of the interface should be based on a real world metaphor - Enable the user to rely on well-understood visual cues, (Print Symbol. )
- Disclose information in a progressive fashion - The interface should be organized hierarchically. The information should be presented at a high level of abstraction.

# GOLDEN RULE – MAKE THE INTERFACE CONSISTENT

- Allow the user to put the current task into a meaningful context - The user should be able to determine where he has come from and what alternatives exist for a transition to a new task.
- Maintain consistency across a family of applications - "MS Office Suite"
- If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so - Once a particular interactive sequence has become a de-facto standard (Alt-S ➔ save file), the user expects this in every application she encounters.

# INTERFACE ANALYSIS

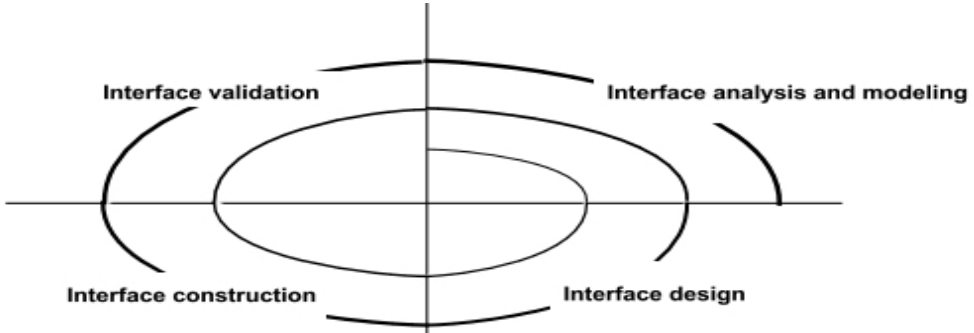
Interface analysis means understanding :

(1) the people (end-users) who will interact with the system through the interface

(2) the tasks that end-users must perform to do their work

(3) the content that is presented as part of the interface

(4) the environment in which these tasks will be conducted (e.g. embedded system)




User Interface Design Process

# USER ANALYSIS

- Are users trained professionals, technician, official, or manufacturing workers?
- What level of formal education does the average user have?
- Are the users capable of learning from written materials or have they expressed a desire for classroom training?
- Are users expert typists or keyboard phobic?
- What is the gender and age range of the user community?
- How are users compensated for the work they perform? Do users work normal office hours or do they work until the job is done? (banking software)
- Is the software to be an integral part of the work users do or will it be used only occasionally?
- What is the primary spoken language among users?
- What are the consequences if a user makes a mistake using the system?
- Are users experts in the subject matter that is addressed by the system?
- Do users want to know about the technology the sits behind the interface?

# TASK ANALYSIS AND MODELLING



- Answers the following questions …
  - What work will the user perform in specific circumstances?
  - What tasks and subtasks will be performed as the user does the work?
  - What specific problem domain objects will the user manipulate as work is performed?
  - What is the sequence of work tasks (hierarchy)—the workflow?
- Use-cases define basic interaction
- Object elaboration identifies interface objects (classes)
- Task elaboration refines interactive tasks
- Workflow analysis defines how a work process is completed when several people (and roles) are involved (swimlane diagram)

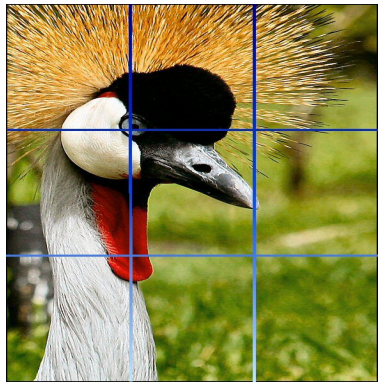
# ANALYSIS OF DISPLAY CONTENT

- Are different types of data assigned to consistent geographic locations on the screen? (e.g., photos always appear in the upper right hand corner)
- If a large report is to be presented, how should it be partitioned for ease of understanding?
- Will mechanisms be available for moving directly to summary information for large collections of data?
- Will graphical output be scaled to fit within the bounds of the display device? (e.g. smart phones)
- How will color to be used to enhance understanding? (errors are in red color)
- How will error messages and warning be presented to the user? (dialogue box, or text message)

Remember that the font carry a message to!

**PEACE**

*WAR*



# INTERFACE DESIGN PRINCIPLES-I

- Anticipation — WebApp should be designed so that it anticipates the use's next move
- Communication —The interface should communicate the status of any activity initiated by the user (progress bar)
- Consistency —The use of navigation controls, menus, icons, and aesthetics (e.g., color, shape, layout) consistent in each webpage.
- Controlled autonomy —The interface should facilitate user movement throughout the WebApp, but it should do so in a manner that enforces navigation conventions that have been established for the application.
- Efficiency —The design of the WebApp and its interface should optimize the user's work efficiency, not the efficiency of the Web engineer who designs and builds it or the client-server environment that executes it.

# INTERFACE DESIGN PRINCIPLES-II



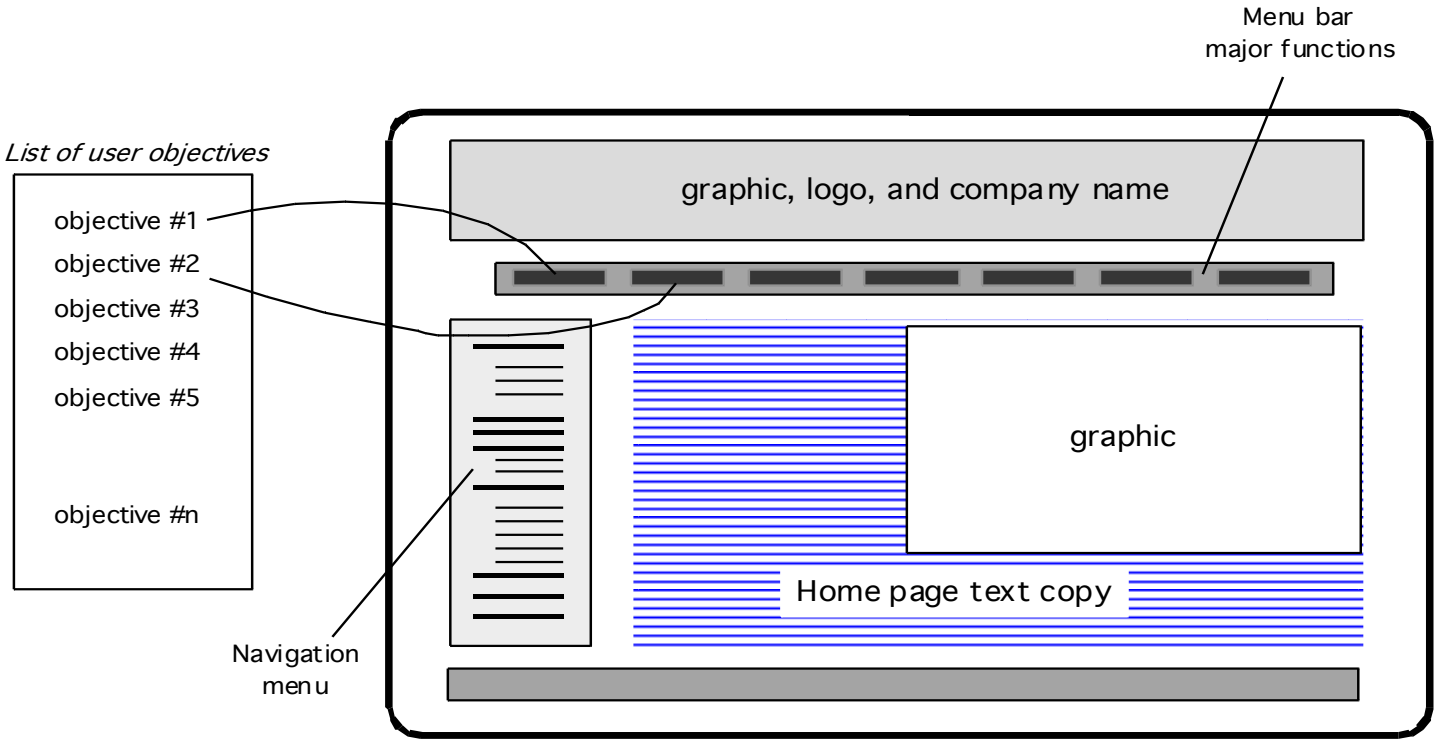
- Focus— WebApp interface (and the content it presents) should stay focused on the user task(s) at hand.
- Fitt's Law—"The time to acquire a target is a function of the distance to and size of the target."
- Human interface objects—A vast library of reusable human interface objects has been developed for WebApps (bootstarp).
- Latency reduction— WebApp should use multi-tasking in a way that lets the user proceed with work as if the operation has been completed.
- Learnability— WebApp interface should be designed to minimize learning time, and once learned, to minimize relearning required when the WebApp is revisited.

# INTERFACE DESIGN PRINCIPLES-III

- Maintain work product integrity—A work product (e.g., a form completed by the user, a user specified list) must be automatically saved so that it will not be lost if an error occurs.
- Readability—All information presented through the interface should be readable.
- Track state—When appropriate, the state of the user interaction should be tracked and stored so that a user can logoff and return later to pick up where she left off.
- Visible navigation—A well-designed WebApp interface provides "the illusion that users are in the same place, with the work brought to them." (rather than SCROOLLING)
- Don't be afraid of white space
- Emphasize content rather style
- Organize layout elements from top-left to bottom right

# MAPPING USER OBJECTIVES (WAREFRAMMING)





# REFERENCES


- R.S. Pressman & Associates, Inc. (2010). *Software Engineering: A Practitioner's Approach.*
- Kelly, J. C., Sherif, J. S., & Hops, J. (1992). An analysis of defect densities found during software inspections. *Journal of Systems and Software*, *17*(2), 111-117.
- Bhandari, I., Halliday, M. J., Chaar, J., Chillarege, R., Jones, K., Atkinson, J. S., & Yonezawa, M. (1994). In-process improvement through defect data interpretation. *IBM Systems Journal*, *33*(1), 182-214.