Software Engineering

Lecture 7:

Software Implementation

User Interface Design : Cont. Lecture 6

- *User interface design is an essential part of the overall software design process and that because.*

1. ***System users often judge a system by its interface rather than its functionality.***
2. ***A poorly designed interface can cause a user to make catastrophic errors.***
3. ***Poor user interface design is the reason why so many software systems are never used.***

UID: Cont. Lecture 6

- *Good user interface design is critical for system dependability.*
- *Many so-called 'user errors' are 'caused by the fact that user interfaces do not consider the capabilities of real users and their working environment.*
- *A poorly designed user interface means that users will probably be unable to access some of the system features, will make mistakes and will feel that the system hinders rather than helps them in achieving whatever they are using the system for.*

UID: Cont. Lecture 6

- ***User Interface Design Principles:*** *Design principles are high-level principles that can guide the design of a user interface. These principles are:*
  1. ***User familiarity :*** *The interface should use terms and concepts drawn from the experience of the people who make most use of the system.*
  2. ***Consistency :*** *The interface should be consistent in a way that, wherever possible, comparable operations should be activated in the same way.*

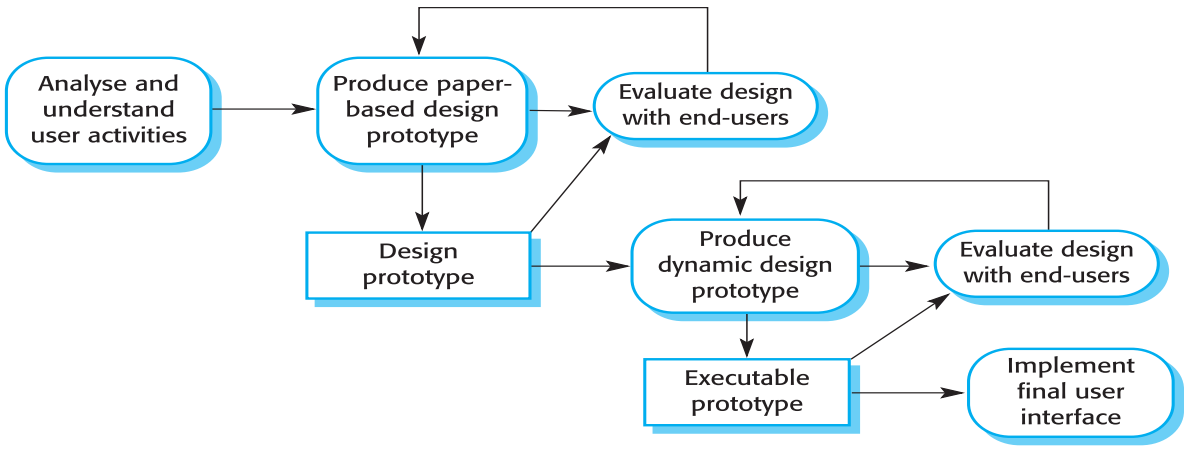
UID: Cont. Lecture 6

3. ***Minimal surprise :*** *Users should never be surprised by the behavior of a system.*

4. ***Recoverability :*** *The interface should include mechanisms to allow users to recover from errors.*

5. ***User guidance :*** *The interface should provide meaningful feedback when errors occur and provide context-sensitive user help facilities.*

UID: Cont. Lecture 6



UID: Cont. Lecture 6

- ***User Interface Design Process:*** *User interface design is an iterative process involving close correlation between users and designers. The 3 core activities in this process are:*

1. ***User analysis****: Understand what the users will do with the system. If you don't understand what the users want to do with a system, you have no realistic prospect of designing an effective interface.*

- *User analysis have to be described in terms that users and other designers can understand.*

UID: Cont. Lecture 6

2. ***System prototyping****: Develop a series of prototypes for experiment; the aim of prototyping is to allow users to gain direct experience with the interface. With out such direct experience, it is impossible to judge the usability of an interface.* ***Prototyping techniques:***

A. ***Script-driven prototyping****: Develop a set of scripts and screens using a tool such as Macromedia Director. When the user interacts with these, the screen changes to the next display.*

B. ***Visual programming:*** *Use a language designed for rapid development such as Visual Basic.*

C. ***Internet-based prototyping:*** *Use of web browser and associated scripts.*

UID: Cont. Lecture 6

3. ***Interface evaluation****: Experiment with these prototypes with users. Some evaluation of a user interface design should be carried out to assess its suitability.*

- ***Simple evaluation techniques:***

1. ***Questionnaires for user feedback.***
2. ***Video recording of system use and subsequent tape evaluation.***

Software Implementation: Build or buy

- *In a wide range of domains, it is now possible to buy* ***off-the-shelf systems (COTS)*** *that can be adapted and tailored to the users' requirements.* ***For example****, if you want to implement a medical records system, you can buy a package that is already used in hospitals. It can be cheaper and faster to use this approach rather than developing a system in a conventional programming language.*
- *When you develop an application in this way, the design process becomes concerned with how to use the* ***configuration features*** *of that system to deliver the system requirements.*

Software Implementation

- *Implementation is the process of realizing the design as a program. One of the most important implementation decisions that has to be made at an early stage of a software project is whether or not you should buy or build the application software.*
- *In a wide range of domains, it is now possible to buy* ***commercial off-the-shelf systems (COTS)*** *that can be adapted and tailored to the users' requirements.*

Software Implementation: Reuse

- *From the 1960s to the 1990s, most new software was developed from scratch, by writing all code in a high-level programming language.*
- *Most modern software is constructed by reusing existing components or systems.*
- *How to reuse existing knowledge and software should be the first thing you should think about when starting a software development project.*
- *The reuse of existing software is now generally used for* ***business systems****,* ***scientific software****, and, increasingly, in* ***embedded systems engineering****.*

Software Implementation: Reuse

- ***Software reuse is possible at a number of different levels:***

1. ***The abstraction level:*** *At this level, you don't reuse software directly but rather use knowledge of successful abstractions in the design of your software.*
2. ***The object level:*** *At this level, you directly reuse objects from a library rather than writing the code yourself.*
3. ***The component level:*** *Components are collections of objects and object classes that operate together to provide related functions and services. You often have to adapt and extend the component by adding some code of your own.*

Software Implementation: Reuse

4. ***The system level:*** *At this level, you reuse entire application systems. This usually involves some kind of configuration of these systems. This may be done by adding and modifying code (if you are reusing a software product line) or by using the system's own configuration interface.*

- *Most commercial systems are now built in this way where generic* ***COTS (commercial off-the-shelf)*** *systems are adapted and reused. Sometimes this approach may involve reusing several different systems and integrating these to create a new system.*

Software Implementation: Reuse

- *By reusing existing software, you can develop new systems **more quickly**, with **fewer development risks** and **also lower costs.***
- *As the reused software has been tested in other applications, it should be **more reliable** than new software. However, there are costs associated with reuse:*

1. ***The costs of the time spent in looking for software to reuse and assessing whether or not it meets your needs.** You may have to test the software to make sure that it will work in your environment, especially if this is different from its development environment.*

Software Implementation: Reuse

2. ***The costs of buying the reusable software**. For large off-the shelf systems, these costs can be very high.*
3. ***The costs of adapting and configuring the reusable software components** or systems to reflect the requirements of the system that you are developing.*
4. ***The costs of integrating reusable software elements with each other (if you are using software from different sources)** and with the new code that you have developed. Integrating reusable software from different providers can be difficult and expensive because the providers may make conflicting assumptions about how their respective software will be reused.*

Software Implementation: Challenges

- *There are some challenges faced by the development team while implementing the software. Some of them are mentioned below:*

1. ***Code-reuse*** *- Programming interfaces of present-day languages are very sophisticated and are equipped huge library functions. Still, to bring the cost down of end product, the organization management prefers to re-use the code, which was created earlier for some other software. There are huge issues faced by programmers for compatibility checks and deciding how much code to re-use.*

Software Implementation: Challenges

2. ***Version Management:*** *Every time a new software is issued to the customer, developers have to maintain version and configuration related documentation. This documentation needs to be highly accurate and available on time.*

3. ***Target-Host:*** *The software program, which is being developed in the organization, needs to be designed for host machines at the customers end. But at times, it is impossible to design a software that works on the target machines.*