Ask professor when he will make the clarification

We can ask TAs about ORD

6. EXTERNAL DESIGN

Last Mod: 2024-05-26

Extends in UML diagram:
means that it is an extension to a scenario

This section of the course will look at:

Will need to figure out the sequence of steps that will occur when creating an issue, among other commands

- Why project planning should come next in the course but doesn't.

Use case diagram does not need to be intricate.
You only need to document each scenario

- A very brief overview of user interface design issues.
- Writing a User Manual.

Optional Readings (neither are in coursepak):

- Optional: Pages 395-405 of [Pressman97]

Recommended textbooks

- Optional: Pages 319-343 of [Sommerville96]

Table of Contents

6.1 Why Project Planning is Not Next

After requirements are specified, normally senior staff do a "project management" plan. This normally involves:

- Estimating project size Number of lines of code
- Estimating project effort
- Estimating project resources needed (people, machines, etc.)
- Estimating budget
- Estimating project schedule

- Writing a Risk Management strategy:
  - list the risks that could affect the project,
  - how the risk might/will be prevented,
  - how monitored for emergence, and
  - if occur how could they be handled.

Ex: use sprints to break things up into two week tasks
Breaking things up piece by piece, we can see if we are falling behind

- Writing Quality Assurance Plan

How the testing and issue tracking will be done

- Writing a Configuration Management Plan

How the project will be done (what will be the workflow of the project, eg what branches will be used for development, canary, etc)

At SFU in Cmpt 276, only have one semester.

- At UBC, software engineering is taught over 2 semesters.

And our nice phased "set" assignments require the material for each assignment be taught before each assignment begins.

So there is simply not time now to present this material.

- I'll not have you to write a project plan, QA plan, nor CM plan, so this material is deferred until near the end of the course.

6.2 Design Phase

There are 4 sub-phases of the design phase:

1. **External Design** - How the user and other entities interact with the system.
   - Deliverable is a draft user manual, or network interaction documentation.
2. **Architectural Design (High-Level Design)** – How to partition into smaller, manageable pieces (subsystems).
   - Possibly each running on separate CPUs.
   - The major deliverable is the Architectural Design Document.

Ejemplo dado:
Russell preguntaba la gente porque
la informacion def un telefono necesitaba estar
guardado en una base de datos.
Tal vez tienes que query quien llamo en
la ultima hora

3. **Software Interface Design (Mid-Level Design)** - Where externally-visible definitions of major elements of a modules are defined.
   - I.e. Function prototypes/signatures, class skeletons, and global variables and types are defined and commented, *so that other programmers can thence work in parallel coding and compiling calls to your module. (Linking can come later).*
   - The major deliverable is well-commented header files (or whatever your language calls them).
     - Note Java does not use header/interface files, though you can rough out the classes and use a tool to extract the prototypes and a 'special' subset of the comments.

4. **Detailed Design (Low-Level Design)** - Basically deciding, from among several options, which
   - data representation/structures will be used, and
   - which algorithms will be used.

6.3 External Design

In External Design sub-phase, we plan/define how the system should interact with the world around it. This has 3 aspects:

- User Interface Design
- Networking interfaces to other applications.
- File interfaces for files that should be readable by other applications.

Of these three aspects of External Design, in Cmpt 276 we'll only cover User Interface Design.

You can't design and code a system if you don't know:

- what the menus are supposed to look like,
- how they will operate, and
- whether the date on printouts is supposed to be 8 characters in dd-mm-yy or yy/mm/dd format, or 37 characters where the month is spelled out. Any should the date be in the top left or top right of a screen or printout.

This stuff is NOT up to the programmer. Generally, people who design user interfaces (not GUIs, but the user interface for a particular application) have some training in this area.

E.g. SFU Cmpt-363: User Interface Design

CALENDAR DESCRIPTION:

This course provides a comprehensive study of user interface design. Topics include: goals and principles of UI design (systems engineering and human factors), historical perspective, current paradigms (widget-based, mental model, graphic design, ergonomics, metaphor, constructivist/iterative approach, and visual languages) and their evaluation, existing tools and packages (dialogue models, event-based systems, prototyping), future paradigms, and the social impact of UI.

COURSE DETAILS:

This course introduces students to the art and science of designing usable, useful and enjoyable human-computer interfaces, with an emphasis on user-centered design techniques. By the end of the course, students will be familiar with different user-centered design approaches, understanding user needs, prototyping methods, and interface usability evaluation techniques. Students will gain valuable knowledge and experience by working on a hands-on design project.

Here is an extract from an old Cmpt 212 assignment to illustrate. The application maintains a list of retail businesses and the wholesale dealers they buy from.

Here are three *shockingly* different layouts for the menu, sub-menu, and sub-sub-menu items!

Cases in use case diagram should be influenced by what is in the list of possible commands

Menu Design A:

```
List                    Miscellaneous                   Exit
    Add Business            Copy the list
    Add Dealer              Save To
    Display list            Restore From
    Empty the list          Number in list
```

Menu Design B:

```
File                    List                            Number in list
    Save To                 Add to list
    Restore From                Add Business
    Exit                        Add Dealer
                            Display list
                            Copy the list
                            Empty the list
```

Menu Design C:

```
Add to list         Display             Manipulate          Exit
    Business            List                Copy  list
    Dealer              Total_Number        Save To
                                            Restore From
```

Often many ways to design something.

This is what design is about:

- **Not** choosing the first thing that comes into your head,
- but *evaluating and choosing* among alternatives
  - (none of which are perfect; each has its pros and cons).

Which of the above 3 menu structures do you like best?

- Some have *verbs* like 'Add', 'Display', and 'Manipulate' as top menu items.
- Others have *things* like 'File', 'List', and 'Total_Number' as the top level items.

External Design easy to document by writing and getting review feedback for a first draft of the user manual.

- Which, interestingly, is a kind of rapid prototype!.

The software programmers often have to write a user manual anyway.

- Though they hope the job given someone else (E.g. technical writers).

But others can't write a manual until they know what the product will do. And often the UI is designed before you can run the code to see what it will look like and do.

And even if the code is written so the technical writers can try operating it, they may not know the application domain well.

Many design decisions that SFU students might previously have made while programming, really should be made in an “external design” sub-phase.

A draft manual is best shown to customer/user/market-oriented personnel for review and approval, BEFORE THE SOFTWARE IS DESIGNED OR CODED.

- So the designers do not design a product that doesn't meet the users requirements for:
  - functionality, User manual forces programmers to follow the design
  - friendliness, and
  - UI system features (e.g. flying file folders animation).

Management should NOT assume programmers are good UI designers and let them start programming the application, because some atrocious UIs can result.

- Just like bridges should not be built until more than one person has reviewed the design for adequate strength!

Once the draft user manual has been approved, system design and coding can proceed with:

- far LESS DISTRACTION with esthetic issues, and
- LESS WORRY that half-way through implementation, someone will object to the way the software looks or operates and order an UI design or feature change (thus throwing code away!).

Is this actually true? Could you not have a capricious client?

Also, changes late in development often require major changes to code.

And worse yet, sometimes to the architecture of the whole design.

Where does this number come from?

- This often costs 10 times as much to correct late (vs. early in development), when simply writing a draft user manual early will inexpensively prevent this risk.
  - And write a manual that must be written anyway).

Having prototyped on paper:

Does this mean that the number of features to implement is limited?

- the design can proceed in a focussed fashion, with the designers not having to leave so many implementation options open, and

Does the design limit implementation options?

Does this mean that the design dictates the implementation?

- the designers are more likely to have had feedback on the features and even future features required.
  - The manual acts as a stimulus for getting more interaction with the customer/user, and
- programmers can be employed who are less familiar with the UI and application specific aspects of the design. They can be told to simply “make it so”.

6.4 User Classes

Many systems present different screens or possibly menu items to different users, depending on the user's:

- job classification or
- on security issues.

6.5 UI Technology

There are several categories of UI technology:

- Command line programs with command line parameters.
  - E.g. Git for Unix shell. Must memorize commands and command parameters.
- Scrolling, character-based console applications that could potentially use character rather (than GUI) menus.
  - E.g. "Console" programs like Cmpt 276 project.
- Screen-oriented, character-based UIs where user can tab around.
  - E.g. Old airline check in counter screens.

- Graphical user interfaces (GUIs).
  - Not all GUIs use mouse paradigm.
  - Some may use virtual reality input sensors like gloves.

Generally, GUI applications require an entirely different design than character-based UIs.

- This is why UI technology decisions should be made early.

GUIs generally use an ‘event driven’ design.

- In an event-based interface, the application ‘registers’ the events it is interested in by telling the GUI system what functions to call when:
  - each kind of input (left click vs. right click) happens
  - in each part of screen (on a GUI button, vs. on scrollbar, etc.).

You store what subsequent actions should occur when clicking a certain part of the screen

- At runtime start up, telling the GUI system which function to call is an important use of function pointer variables.
  - Thus instead of your program calling screen input/output functions, instead the screen input/output software (GUI) “calls back” your program!
  - Java doesn’t have function pointers, but does something that has the same callback nature.

6.6 Human Factors Issues

There are important 'human factors' issues in UI design, that need thought by NONTECHNICAL people.

Unfortunately, those who are both involved in the design, and technical in personal background,

- find it **extremely hard** to stand back and think about how a non-computer literate person,
- who might not even be familiar with the application domain,

would cope with starting to use the application.

If you are making an application to someone, is it not oriented towards people who are familiar with the domain?

Why is the spell checker on the:

- Utilities menu in MS-Word and on the
- Edit menu in Framemaker and on the NeXT WriteNow word processors?

Where do other applications on the same OS put such things?

Where is it most convenient?

Should it be put both/two places for convenience?

  – Or only one place but also available by Cntl-F1?

When done with a menu item, should the system go back to the menu, or ask the user whether they want to do another similar operation?

There are many users who cannot handle complicated menus or screens.

- Maybe a taller menu tree with fewer choices on each menu would be more suitable?
- Maybe initially the menu system could be simple (so novices can't delete files), but users can later change to a more full-featured menu (become members of the expert user class/state)?
  - This protects them until they can figure out how to set themselves to be expert users.

Could you not have users manipulate the system to let them use features designated for experts?

- Maybe an ICON oriented touch screen would be better?

Finally, there are UI aspects that are:

- human factors,
- friendliness, and
- technically advantageous.

E.g. Scrolling lists allow users to easily select from a set of existing data.

- In addition to reducing typing and typing mistake delays, users selecting from a list are less likely to cause loss of referential integrity because the data in the list is known to exist!

For the assignment, we do not need to verify that a department within a company exists.

6.7 Use Case Scenarios

The smooth feeling that a user gets about a UI design is strongly related to the attention that designers paid to use case scenarios.

A use case ‘scenario’ is a time-ordered sequence of interactions between the user and the system. As you know, to complete a complicated user operation, there is often moderately long sequence of subsidiary steps involved in a user operation (e.g. “confirm you want to deleted this file?”).

Use case typically starts with the system in a 'quiescent' mode.

- Often just doing nothing but showing the top level menu.

The scenario, once started, causes the system to go through a **sequence of 'solicited' interactions** with the user (i.e. multiple prompt and user response events).

Here's an example:

1. Show submenu, then user picks 'Add Object' item.
2. Prompt for object key, then user enters ID.
3. If object ID already exists, then show error message and ask user if they want to try again or abort (Note: UIs should be designed to allow users a way to abort).
4. Prompt for object attribute, then user enters attribute.
5. System might reject entry as invalid format (e.g. character string entered instead of an integer). Show error message, etc.
6. Prompt for another attribute, and user entered attributes.

7. For foreign key attributes, check if the corresponding foreign object instance actually exists. If not, ask the user if they want to either:
   - enter a new foreign object,
   - correct their mis-entered attribute, or
   - abort.
8. Ask if user is happy and actually wants to go ahead with the complete object add.
   - Common on bank or shopping web sites.
9. Ask user if they want to enter another of the same kind of object?
   - This saves them from having to go back to the menu and reselect this same user operation.

- There are many jobs where clerks spend most of the day entering data.
  - E.g. Think of entering your grades for each course into a database.
  - Often data entry operators can acknowledge that they want to enter another object with just a carriage return.

10. If not show submenu again (or might main menu be more appropriate in this application domain?).

The above scenario is well thought out:

- Starts in a known quiescent state.
- Requests key attributes first, as there is no use wastefully entering the rest of the attributes if that object that already exists.
- Systems:
  - by nature (you may not be able to change the readInteger() library function), or
  - by your design, should usually check input.

Typically you want to check for:

- **illegal format** (alphabetic characters into an integer read),
- **key/object uniqueness** (you don't want two object with same key and different data in the database, because which one that a query would find is then ill defined), and
- **referential integrity** (you don't want a foreign key that formalizes a relationship that when navigated gives a dead end!).

A scenario properly identifies where in the sequence that errors or exceptional situations from the above checks can affect the normal flow of the scenario.

And the way the user is informed, and extracts him/herself from each situation is made clear.

- The user is given ways to abort along the way, and (sometimes annoyingly) at the end.
  - User HATE BEING TRAPPED (e.g. can't remember password).
- User is given shortcuts for commonly needed sub-scenarios:
  - e.g. Foreign object does not exist: jump to an add screen for that object, and when done, then back to this scenario.
    - This nesting is an important emerging element in the application of scenario analysis.
  - e.g. Adding many objects in a row: It is nice to not have to reselect this menu command between each object entry.

- When a scenario is complete, it's documented what the software will present to the user next.
  - Preferably that which will be convenient for the most users doing the most operations.

6.8 Good Advice

[Pressman97] has 3 good lists of advice on UI design. I will only name his points. The optional readings contain a paragraph on each point.

General Interaction Advice List:

- Be consistent (e.g. with key shortcuts, with color, etc.)
- Offer meaningful feedback.
  - Especially give feedback to assure the system is working hard, not stalled!
- Ask for verification of any non-trivial destructive action.

Limit the interaction needed with the user
Place icons in places close to the user

- Permit easy reversal of most actions (e.g. undo).
- Reduce amount of info that must be remembered between actions.

Amount of info the user has to remember?

- Seek efficiency in dialog, motion, and thought.
- Forgive mistakes.

For each day the assignment is late, 5% is taken off

You do not need to justify why the prototype has certain features

  - System should protect itself and
  - help user recover.
- Categorize activities by function and organize screen geography accordingly.
- Provide help facilities that are context sensitive (if possible).

- Use simple action verbs or short verb phrases to name commands.

Remember there may need to be help files available, a command reference manual written, training videos, training course curricula developed, trainers trained, etc.

Information Display List:

- Display only information relevant to context.
- Don't bury the user with data; use a presentation format that enables rapid assimilation of data.
- Use consistent labels, standard abbreviations, and predicable colors.
- Allow the user to maintain visual context of where in system she is.
- Produce meaningful error messages.
- Use upper and lower case, indentation, and text grouping to aid in understanding.

For showing issues, you want to show them only 20 at a time and may want to give more space for the description
Place description to the left
If you limit the issues shown to those within the last two releases, you run the risk of not demonstrating an issue has been fixed in an older release

Predictable?

Can use one letter label for status of issue

- Use windows to compartmentalize different types of info.
- Use ‘analog’ displays to represent information that is more easily assimilated with this form of presentation (e.g. a thermometer-shaped bar graph for temperature).
- Consider the (limited) available geography of the display screen and use it efficiently.

Data Input Advice List:

- Minimize the number of input actions required of the user.
  - Reduce number of mouse clicks needed.
  - Especially on frequent operations.
  - And on always-used operations like logging out or shutting down.
- Maintain consistency between data input and information displayed (e.g. fonts, colors, precision)
- Allow the user to customize input.
- Interaction should be flexible but also tuned to the user's preferred mode of input.

- Deactivate commands that are inappropriate in the current context.
- Let the user control the interactive flow.
- Provide help to assist with all input actions.
- Eliminate “mickey mouse” inputs that are unnecessary.

Micky Mouse inputs are inputs that are invalid

6.9 Other Manuals

For a huge software system, there may be many other manuals that need to be written:

- Installation Manual
- Instead of just a User Manual, two documents:
  - Introductory Manual and
  - Reference Manual
- Functional Description
- System Administrator's Manual.

6.10 References

[Pressman97] “Software Engineering, a Practitioner’s Approach, 4th ed.”, by Roger Pressman, McGraw-Hill, 1997.

[Sommerville96] “Software Engineering, 5th ed.” by Ian Sommerville, Addison-Wesley, 1996.

6.11 Appendix A - TOC and Layout Sheets

A separate document will be provided that contains an example Table of Contents for an application's user manual.

- And within it miscellaneous advice on writing a user manual.

In addition, it will provide an 80-character wide screen layout planning sheet, and a 136-character wide printed report layout planning sheet.

- Make several copies of the screen planning sheet, to discuss, plan, and document screen layout plans.