Hangman Game

- Simple design exercise similar to what you will do for your CS102 project (and what happens in industry!)
- Stages:
  - Requirements,
  - UI,
  - Detailed Design,
  - implementation,
  - Testing,
  - Maintenance, ...
- Objective:
  - Demonstrate process and problems associated with building software in groups.
  - Being aware of the problems may help you avoid some of them!

Requirements Stage

- Groups of 4/5
- Each group should come up with a written description of the game
- Explain it to little brother/sister!
- Each group read their description.
  - Similarities
  - Omissions, etc.

Description

- In the game of Hangman, the player must find a word chosen secretly by the program. The player can try one letter at a time and the program says how many times and where the letter appears in the secret word. If the player tries more than a certain number of incorrect letters (i.e., letters which do not appear in the secret word), they lose the game and the program tells them the secret word; otherwise, they continue until they uncover the complete word.
- Usually, players are able to see the partially formed word made by the letters so far correctly guessed, any unknown letters being left blank. They may also be able to see the set of all letters that might be in the word (with those already used possibly being removed), as well as the number of incorrect tries made so far (this is often shown graphically, by the number of visible body parts of a cartoon character which is hung when complete –hence, the name of the game!)

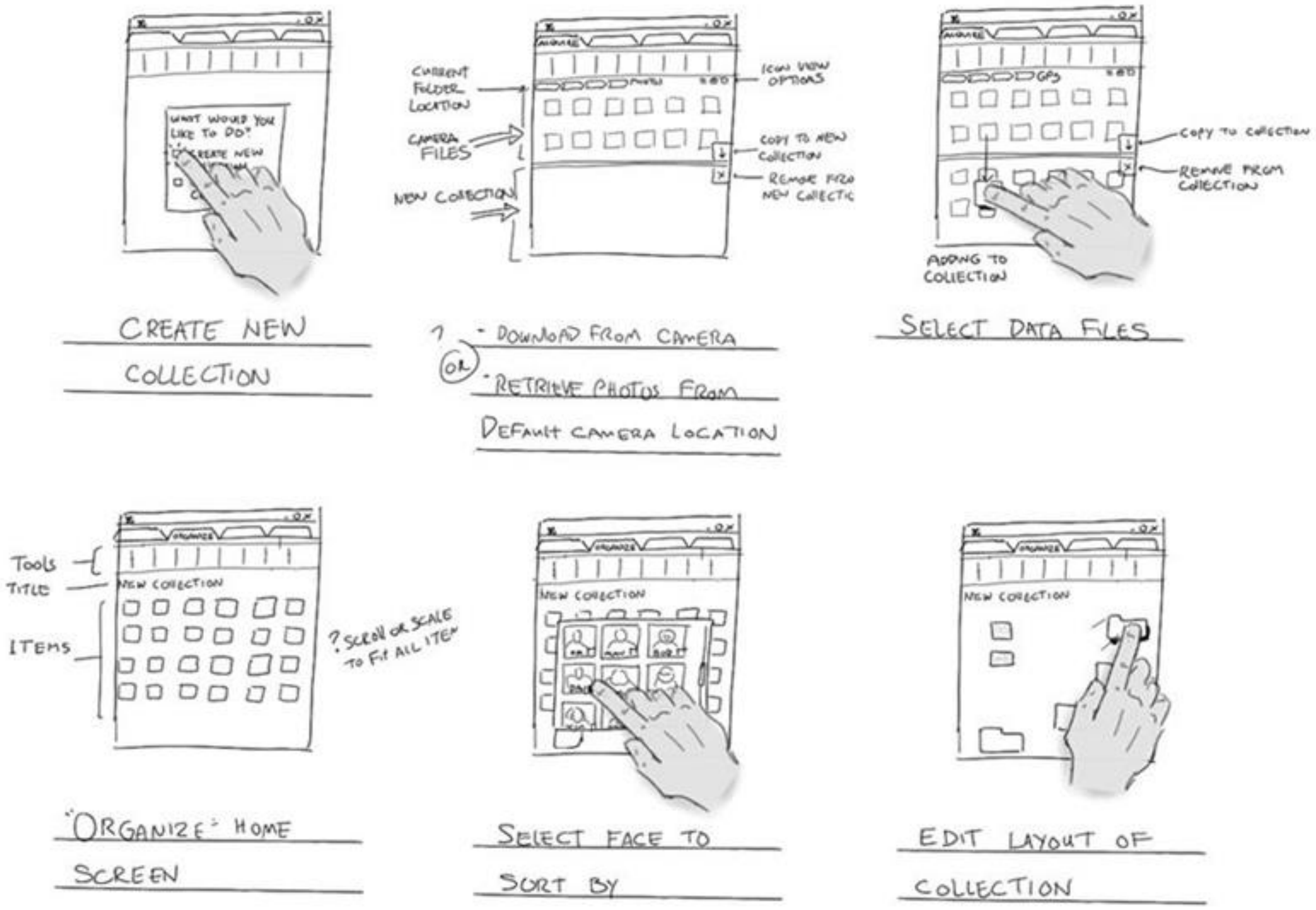
Now, be creative…

- Lots of people selling such a game; why should they buy yours?
- How about variations on the same game but with different UI/theme?

User-Interface Stage

- Communicating the UI design... how?
- Pictures with "links" saying "if this happens" the result is "this”
- Try it for the standard game.

UI Design Storyboard Example



Hangman Code Examples & Tutorials

- **Creating a Hangman Game in Java**
https://medium.com/swlh/creating-a-hangman-game-in-java-2c3088cb0d6d

- **Java for Beginners: Hangman Game in Terminal**
https://www.youtube.com/watch?v=VRN6cgv59Ak

- SedaKunda/**hangman.java**
https://gist.github.com/SedaKunda/79e1d9ddc798aec3a366919f0c14a078

- **Simple Java Hangman Assignment**
https://stackoverflow.com/questions/4988143/simple-java-hangman-assignment

- Text-based Hangman game in Java
https://codereview.stackexchange.com/questions/171369/text-based-hangman-game-in-java

User Interface

- Notice
  - Different versions of the "game"
  - Different UI features (purely text-based, i.e., console I/O, animation, music/sounds, keyboard, mouse, etc.)

Detailed Design Stage

- First design, then implement! But how do we come up with design?
- In groups again,
  - One person be the "display“
  - One the "game"
  - One the "player" (other people can act as guides and recorders)

- The display needs certain info to produce the UI... it needs to ask the "game" (model) for this.
  - Exactly what does it ask for? What info, if any, does the game need to supply? What are the data types?
- Ok, user can see display
  - What do they need to ask the model to do? Types, etc. again. Updating the display may require additional info... and so on.
- The end result should be a good set of methods and properties for the game/model (and GUI/display) classes.

Detailed Design Stage

- **class Hangman**
- constructors
  - + Hangman()
    // default max 6 incorrect tries, English alphabet,
    // chooses secretWord from fixed list.
- properties
  - secretWord : StringBuffer
  - allLetters : StringBuffer
  - usedletters : StringBuffer
  - numberOfIncorrectTries : int
  - maxAllowedIncorrectTries : int
  - knownSoFar : StringBuffer // secretWord but with chars
    // not yet found blanked out

Detailed Design Stage

- methods
  - + getAllLetters() : String
  - + getUsedLetters() : String
  - + getNumOfIncorrectTries() : int
  - + getMaxAllowedIncorrectTries : int
  - + getKnownSoFar() : String // returns partial word formed with known letters only
  - + tryThis(letter) : int // returns number of occurrences of letter in secretWord
  - + isGameOver() : boolean
  - + hasLost() : boolean
  - – chooseSecretWord() // initially use fixed list,
    // called from constructor

Tasks

- **`main`** method – plays the game by creating instance of Hangman class & calling its methods.
- constructor for **`Hangman`** class
- **`tryThis`** method
- **`chooseSecretWord`** method
- **`Hangman`** class (but with the bodies of the constructor, **`chooseSecretWord`**, & **`tryThis`** methods being empty.)