Peter Magalhaes 12-11-13
CIS 498 Senior Software Engineering Project I
Second Mid-Term Exam
Fall 2013

1. (30p) What is the role of prototyping in the requirements engineering process during the inception and the elaboration phase of the software development process? What role does user interface prototyping play in this process? How is interface prototyping connected to requirements engineering? What role if any does the prototyping play in the construction and transition phase of the software development process?

- inception phase
    - identify the stakeholders
        - identify who will be responsible for the requirements being successful or not
        - identify who is providing finances for the solution to be created and their goals
    - identify the problem
        - conversing with the customer, the company, and all other stakeholders
        - to come to an agreement on the set of requirements that are necessary for project direction
        - understand the problem, how it was created, and how the customer's process works
        - discuss a direction and negotiate an approach to possible solutions
    - designate forms of communication
        - we need to identify methods of communication with the customer and stakeholders in order for collaboration to take place
        - also need to commit to a method of communicating and collaborating within the project team
    - gather multiple views or understandings
        - get a personal view from multiple people who are experiencing the problem, or are affected by the process
        - identify, or meet, with other people outside of the developer and customer, who may have experience with the problem, or the process being used
        - meeting with people multiple times, as some pieces of information are not always mentioned the first time, as extra meeting can cause reiteration using different words

- elaboration phase

    - create analysis model
        - develop use case models, diagrams, scenarios, and other visualizations to help understand the problem and help the customer understand the possible solution being developed
        - communicate and share these models with the customer, or anyone else who is contributing to the team developing a solution to the problem
    - identify data
        - we need to know what existing software is being used, and what we can reuse, or what needs to be recreated
        - identify third party software being used, and identify compatibility issues that may arise
    - identify and agree on requirements
        - we need to identify and negotiate all the specific requirements, b both functional and behavioral requirements using state diagrams, sequence charts, data flow diagrams, etc
        - all the system features should be part of the requirements, or contribute to the functionality of a portion that satisfies a requirement
        - come to an agreement with the customer on specific functionality that is required by the customer
- role of user interface prototyping
    - to develop a mock gui in order to visually understand what is happening from the user's point of view
    - to identify a specific process and how it will function in combination with other requirements
    - to help stakeholders, customer, or anyone else contributing to the solution visually understand the steps required to perform a specific function
    - to better our understanding of what the user will see, and what operations need to happen in the background
    - to confirm our understanding of the customer's process, and cover all the possible scenarios that may happen
    - essentially to get a visual understanding, and see if there are any missing elements, or functionality that is not presented to the user
- interface connected with requirements engineering
    - a requirement may have a specific design, or need to interface to another piece of software developed by a third party, or that already exists
    - this will also contribute to the team's understanding of what software is interfacing with each other
- role of prototyping in the construction and transition phases
    - when developing diagrams and use-cases, we will be able to define classes and methods required in the construction phase

    - the prototype should ultimately be moving toward the final solution in the transition phase, and the role of prototyping early in development will create a concrete structure for the final solution
    - once a complete prototype is created, we can test it with the cutsomer, and some users who will be using the software, to find bugs or problems before the final release of the solution

2. (30p) Explain significance of software architecture in context of software design and in the wider context of software engineering process. Why is software architecture important, when is the architecture addressed for the first time in the software development process? By when in the software process do we have to define (make a commitment to) the software architecture?

- significance of architecture in context of design
    - great way to represent that software to an engineer to analyze
    - it helps the design models move toward the solution of the problem, and the requirements
    - can help the team decide on which model of architecture to be used when there are different options for a possible solution
- why is software architecture important
    - reduces risks creating during development in the construction phases by allowing the engineer to analyze the planned solution
    - its provides a model that can be addressed visually, whether or not someone understands the underlying coding language or coding functionalities
- when is architecture first addressed
    - should be addressed as soon as possible, in order to get the idea of the planned solution accurate and promoted into development
    - during design is when the architecture should be addressed, but it is better to at least start thinking about architecture during meeting with customers and stakeholders, and promote questions created by the team to inspire the customer to release more information about the problem, and required solution
- when do we commit to the software architecture
    - we should commit to a software architecture before the construction phase, when we are developing the architecture, or shortly after beginning construction of the solution if there are any problems
    - once an agreement has be made between the customer and developers on a particular direction for the architecture, it should be analyzed and tested before completely committing to a full size application

3. (40p) Justify the following basic principles of software design:

- **The Open-Closed Principle** – "A module/component should be open for extension but closed for modification"
    - this means that the component should be extendable to improve its functionality without performing actions like modifying its source code
    - once we have a concrete, error-free component, there should be no need to modify or alter the functionality of the component, which could create problems in large software applications as one method may be used by many classes
- **The Liskov Substitution Principle** – "Subclasses should be substitutable for their base classes"
    - if a created subtype object has a particular type, then the objects of these types can be used alternatively without changing any major functionality
    - this principle helps everyone understand the class hierarchy, and their interchangeability to perform alternate functionality
- **The Release Reuse Equivalency Principle** – "The granule of reuse is the granule of release"
    - this means that the components that are finally released are available to be reused again in other software
    - this provides modularity to the software solution, so each component created may be reused again in the future
- **Dependency Reversion Principle** – "Depend on abstractions. Do not depend on concretions"
    - this states that we should create a concrete abstraction for which details can relate and be expanded on technically
    - once we have an abstraction that is accurate, all the fine details and requirements or functionalities can relate back to the abstraction
- **The Common Closure Principle** – "Classes/modules that change together belong together"
    - this helps to organize particular modules/classes, so when something needs to be changed, all the other elements grouped together are required to change
    - when an alteration is required, it is easier to identify other components that need to be modified in order for the change to be successful and operate as required
- **The Common Reuse Principle** – "Classes/modules that aren't reused together should not be grouped together"
    - this is nearly identical as the closure principle, where if something requires a change, is not affected by something that is not being used, and therefore they are not grouped together
    - this helps to idealize the necessary components required for the functionality, and removes all the unnecessary components
- **The Interface Segregation Principle** – "Many client-specific interfaces are better than one general purpose interface"

- if we have one general purpose interface, that interface may not be completely modular in its code, and many client specific interfaces will have the ability to reuse portions of code or functionality
- in an example of a document editor, a general purpose interface may not be suitable for every client, as each client has different requirements and common functionalities
- we can limit the interface to show only what is applicable by the type of user, where it may be necessary to remove functionality from a general purpose software so users don't go outside their scope of work (like editing formats, or exporting multiple file types)