Case study – software engineering

Explain why programs which are developed using Evolutionary Development are likely difficult to maintain. Evolutionary development is an iterative and incremental approach to software development. Instead of creating a comprehensive artifact, such as a requirements specification, that you review and accept before creating a comprehensive design model (and so on) you instead evolve the critical development artifacts over time in an iterative manner. Instead of building and then delivering your system in a single “ big bang” release you instead deliver it incrementally over time.

Programs that are that are developed in Evolutionary Development are likely to be difficult to maintain because the specifications of evolutionary development projects are often abstract, and as the project continues, the development and validation portions of software engineering overlap one another. This usually results in the systems being poorly constructed due to a good initial specification, and on a large projects make it more difficult to integrate new systems into the evolutionary design.

Lastly the documentation for such projects is often lacking, as the designs are constantly rebuilt to the customer’s specifications. Explain how both the waterfall model of the software process and the prototyping model can be accumulated in the spiral model. Waterfall Model- Abstracts the essential activities of software development and lists them in their most primitive sequence of dependency. Real development projects (software and other) rarely follow such a model literally, mainly because the model can and is applied to itself recursively, yielding an almost fractal fabric of actual activity.

Prototyping Model- Is an attractive idea for complicated and large systems for which there is no manual process or existing system to help determining the requirements. Spiral Model – Is a software development process combining elements of both design and prototyping-in-stages, in an effort to combine advantages of top-down and bottom-up concepts.

Waterfall and prototyping model of the software process can be accumulated in the spiral process model in such a way that the spiral odel is much like the waterfall model, in that there are well defined stages, but different in that once an initial sequence is complete, t6he process start over, correcting problems and expanding ideas to better suit the customer’s needs. Also as an iteration of the spiral maybe short, a prototype can be produced that the costumer can see and work with, to help guide the engineers to more accurately construct what they seek. Describe the main activities in the software design process ant the outputs of these activities.

Using an entity- relation diagram, show possible relationships between the outputs of these activities. Once software requirements have been analyzed and specified, software design is the first of three technical activities: Design, code generation and testing- that are required to build and verify the software. Each of these activities transforms information in a manner that ultimately results in validated computer software. The Software Design process itself can be defined as a process through which requirements are translated into a representation of software.

There is a strong relationship between the analysis model and the design model, in that each of the elements of the analysis model provides

information that is required to create a design model. Data design itself is defined and summarized by Wasserman (Principles of Systematic Data Design, and Implementation” in Software Design Techniques): “ The primary activity during data design is to select logical representations of data objects (data structures) identified during the requirements definition and specification phase.

The selection process may include algorithmic analysis of alternative structures in order to determine the most efficient design or may simply involve the use of a set of modules (a “ package”) that provide the desired operations upon some representation of an object. ” A set of principles was also proposed by Wasserman that may be used to specify and design data. The systematic analysis principles applied to function and behavior should also be applied to data.

The same good principles that are followed in analysis principles should also be applied to help us develop data flow and content, identify data objects, and consider alternative data organization. All data structures and the operations to be performed on each should be identified. The design of an efficient data structure should consider the operations that will be performed on the data structure itself. A data dictionary should be established and used to define both data and program design. A data dictionary explicitly represents the relationships among data objects and the limitations on the elements of a data structure.

Survey the tool available in your local development environment and classify the tools according to the parameters (function, activity, breadth of support).

On a project using object technology and relational databases together a good strategy is to do analysis/domain/conceptual modeling before design object modeling, which in turn leads to physical data design modeling, then mapping the two models, then refactoring in conjunction with performance tuning. This is the overall order, yet you still iterate back and forth as needed. Let’s go at it from a slightly different point of view.

Depending on the nature of your project you could start with a project-level conceptual model (you may not have an enterprise model) or you may start first with traditional object modeling activities such as use case modeling. It doesn’t really matter because agile software developers will iterate to another activity as required. Third, notice how I use the term “ enterprise structural modeling” and not “ enterprise data modeling” – many organizations are choosing to use UML class models or even UML component models (Herzum and & Sims 2000; Atkinson et. l. 2002) instead of data models for structural modeling. Fourth, I’ve combined the notions of conceptual and domain modeling in one as they’re often commingled anyway (if they’re done at all). During the time that the development environment is in place, users will gain experience with the systems, displays, and products and be in a better position to determine how the system will be incorporated into their operational environment during subsequent phases of projects.

Discover ambiguities and omissions in the following statement of requirements for part of ticket issuing system: “ An automated ticket issuing system sells rail tickets. Users select their destination number. The rail ticket is issued and their credit card account charged with it cost. When the user

presses the start button, a menu display of potential destinations activated along with a message to the user to select a destination. Once a destination has been selected, users are requested to input their credit cards.

Its validity is checked and the user is then requested to input a personal identifier. When the credit transaction has been validated, the ticket issued. ” There are ambiguities and omissions in the statement in such away those cases need to be considered for invalid credit transactions, improper cards, incorrect personal ID numbers, over-the-limit transactions, debit transactions; details could be given on how the credit information is input, either by swiping a card, or manually entering the numbers.