OBJECT-ORIENTED RAPID PROTOTYPING pdf

1: Object-Oriented Rapid Prototyping | InformIT

About Features. provides a fast-track explanation of what rapid prototyping really is (as opposed to the folklore of rapid prototyping) -- why it should be object-oriented, how specification methods benefit rapid prototyping, and what kinds of advanced development tools are critical for optimal rapid protoytyping.

This process is in contrast with the s and s monolithic development cycle of building the entire program first and then working out any inconsistencies between design and implementation, which led to higher software costs and poor estimates of time and cost. Prototyping can also avoid the great expense and difficulty of having to change a finished software product. The practice of prototyping is one of the points Frederick P. Details, such as security, can typically be ignored. Develop initial prototype The initial prototype is developed that includes only user interfaces. See Horizontal Prototype , below Review The customers, including end-users, examine the prototype and provide feedback on potential additions or changes. Revise and enhance the prototype Using the feedback both the specifications and the prototype can be improved. If changes are introduced then a repeat of steps 3 and 4 may be needed. Dimensions of prototypes[edit] Horizontal prototype[edit] A common term for a user interface prototype is the horizontal prototype. It provides a broad view of an entire system or subsystem, focusing on user interaction more than low-level system functionality, such as database access. Horizontal prototypes are useful for: Confirmation of user interface requirements and system scope, Demonstration version of the system to obtain buy-in from the business, Develop preliminary estimates of development time, cost and effort. Vertical prototype[edit] A vertical prototype is an enhanced complete elaboration of a single subsystem or function. It is useful for obtaining detailed requirements for a given function, with the following benefits: Obtain information on data volumes and system interface needs, for network sizing and performance engineering, Clarify complex requirements by drilling down to actual system functionality. Types of prototyping[edit] Software prototyping has many variants. However, all of the methods are in some way based on two major forms of prototyping: Throwaway prototyping[edit] Also called close-ended prototyping. Throwaway or rapid prototyping refers to the creation of a model that will eventually be discarded rather than becoming part of the final delivered software. After preliminary requirements gathering is accomplished, a simple working model of the system is constructed to visually show the users what their requirements may look like when they are implemented into a finished system. It is also a rapid prototyping. Rapid prototyping involves creating a working model of various parts of the system at a very early stage, after a relatively short investigation. The method used in building it is usually quite informal, the most important factor being the speed with which the model is provided. The model then becomes the starting point from which users can re-examine their expectations and clarify their requirements. If the users can get quick feedback on their requirements, they may be able to refine them early in the development of the software. Making changes early in the development lifecycle is extremely cost effective since there is nothing at that point to redo. If a project is changed after a considerable amount of work has been done then small changes could require large efforts to implement since software systems have many dependencies. Speed is crucial in implementing a throwaway prototype, since with a limited budget of time and money little can be expended on a prototype that will be discarded. Another strength of throwaway prototyping is its ability to construct interfaces that the users can test. The user interface is what the user sees as the system, and by seeing it in front of them, it is much easier to grasp how the system will function. Requirements can be identified, simulated, and tested far more quickly and cheaply when issues of evolvability, maintainability, and software structure are ignored. One method of creating a low fidelity throwaway prototype is paper prototyping. The prototype is implemented using paper and pencil, and thus mimics the function of the actual product, but does not look at all like it. Another method to easily build high fidelity throwaway prototypes is to use a GUI Builder and create a click dummy, a prototype that looks like the goal system, but does not provide any functionality. The usage of storyboards , animatics or drawings is not exactly the same as throwaway prototyping, but certainly falls within the same family. These are non-functional implementations but show how the system will look. In this approach the prototype is constructed with the idea that it will be discarded

and the final system will be built from scratch. The steps in this approach are: The main goal when using evolutionary prototyping is to build a very robust prototype in a structured manner and constantly refine it. The reason for this approach is that the evolutionary prototype, when built, forms the heart of the new system, and the improvements and further requirements will then be built. When developing a system using evolutionary prototyping, the system is continually refined and rebuilt. For a system to be useful, it must evolve through use in its intended operational environment. A product is never "done;" it is always maturing as the usage environment changesâ€¦we often try to define a system using our most familiar frame of referenceâ€”where we are now. We make assumptions about the way business will be conducted and the technology base on which the business will be implemented. A plan is enacted to develop the capability, and, sooner or later, something resembling the envisioned system is delivered. Although they may not have all the features the users have planned, they may be used on an interim basis until the final system is delivered. To minimize risk, the developer does not implement poorly understood features. The partial system is sent to customer sites. As users work with the system, they detect opportunities for new features and give requests for these features to developers. Developers then take these enhancement requests along with their own and use sound configuration-management practices to change the software-requirements specification, update the design, recode and retest. At the end, the separate prototypes are merged in an overall design. By the help of incremental prototyping the time gap between user and software developer is reduced. Extreme prototyping[edit] Extreme prototyping as a development process is used especially for developing web applications. Basically, it breaks down web development into three phases, each one based on the preceding one. The first phase is a static prototype that consists mainly of HTML pages. In the second phase, the screens are programmed and fully functional using a simulated services layer. In the third phase, the services are implemented. The process is called extreme prototyping to draw attention to the second phase of the process, where a fully functional UI is developed with very little regard to the services other than their contract. Advantages of prototyping[edit] There are many advantages to using prototyping in software development â€” some tangible, some abstract. Prototyping can improve the quality of requirements and specifications provided to developers. Because changes cost exponentially more to implement as they are detected later in development, the early determination of what the user really wants can result in faster and less expensive software. Prototyping requires user involvement and allows them to see and interact with a prototype allowing them to provide better and more complete feedback and specifications. Since users know the problem domain better than anyone on the development team does, increased interaction can result in a final product that has greater tangible and intangible quality. Disadvantages of prototyping[edit] Using, or perhaps misusing, prototyping can also have disadvantages. The focus on a limited prototype can distract developers from properly analyzing the complete project. This can lead to overlooking better solutions, preparation of incomplete specifications or the conversion of limited prototypes into poorly engineered final projects that are hard to maintain. Further, since a prototype is limited in functionality it may not scale well if the prototype is used as the basis of a final deliverable, which may not be noticed if developers are too focused on building a prototype as a model. User confusion of prototype and finished system: Users can begin to think that a prototype, intended to be thrown away, is actually a final system that merely needs to be finished or polished. They are, for example, often unaware of the effort needed to add error-checking and security features which a prototype may not have. This can lead them to expect the prototype to accurately model the performance of the final system when this is not the intent of the developers. Users can also become attached to features that were included in a prototype for consideration and then removed from the specification for a final system. If users are able to require all proposed features be included in the final system this can lead to conflict. Developer misunderstanding of user objectives: Developers may assume that users share their objectives e. For example, user representatives attending Enterprise software e. PeopleSoft events may have seen demonstrations of "transaction auditing" where changes are logged and displayed in a difference grid view without being told that this feature demands additional coding and often requires more hardware to handle extra database accesses. Users might believe they can demand auditing on every field, whereas developers might think this is feature creep because they have made assumptions about the extent of user requirements. If

the developer has committed delivery before the user requirements were reviewed, developers are between a rock and a hard place, particularly if user management derives some advantage from their failure to implement requirements. Developer attachment to prototype: Developers can also become attached to prototypes they have spent a great deal of effort producing; this can lead to problems, such as attempting to convert a limited prototype into a final system when it does not have an appropriate underlying architecture. This may suggest that throwaway prototyping, rather than evolutionary prototyping, should be used. Excessive development time of the prototype: A key property to prototyping is the fact that it is supposed to be done quickly. If the developers lose sight of this fact, they very well may try to develop a prototype that is too complex. When the prototype is thrown away the precisely developed requirements that it provides may not yield a sufficient increase in productivity to make up for the time spent developing the prototype. Users can become stuck in debates over details of the prototype, holding up the development team and delaying the final product. Expense of implementing prototyping: Many companies have development methodologies in place, and changing them can mean retraining, retooling, or both. Many companies tend to just begin prototyping without bothering to retrain their workers as much as they should. A common problem with adopting prototyping technology is high expectations for productivity with insufficient effort behind the learning curve. In addition to training for the use of a prototyping technique, there is an often overlooked need for developing corporate and project specific underlying structure to support the technology. When this underlying structure is omitted, lower productivity can often result. However, prototyping is most beneficial in systems that will have many interactions with the users. It has been found that prototyping is very effective in the analysis and design of on-line systems , especially for transaction processing , where the use of screen dialogs is much more in evidence. The greater the interaction between the computer and the user, the greater the benefit is that can be obtained from building a quick system and letting the user play with it. Sometimes, the coding needed to perform the system functions may be too intensive and the potential gains that prototyping could provide are too small. It expands upon most understood definitions of a prototype. According to DSDM the prototype may be a diagram, a business process, or even a system placed into production.

OBJECT-ORIENTED RAPID PROTOTYPING pdf

2: OORP - Object-Oriented Rapid Prototyping in Technology, IT etc. by www.enganchecubano.com

*Object-Oriented Rapid Prototyping [John L. Connell, Linda Isabell Shafer] on www.enganchecubano.com *FREE* shipping on qualifying offers. Object-oriented methods and the automated tools and languages that support these methods are rapidly replacing structured methods.*

Object-oriented development is the process of turning an idea or a problem specification into an object-oriented program. That program consists of a group of objects that communicate with one another. Object-oriented analysis OOA transforms a glimmer of a product into an object-oriented model of that problem. A definition of the functional requirements of the system to be developed and a description of the classes in the problem domain are included in a typical object-oriented analysis model. Some analysis models also include a definition of the high-level system components such as subsystems and their interactions, as well as the general interactions among instances of the problem domain classes. Object-oriented design OOD entails transforming the analysis model into a feasible design. Much of design involves refining the analysis model through the introduction of classes and algorithms that define exactly how certain required features are realized. Another common design activity is the development of an overall system architecture. This entails organizing the system as a set of major building blocks, such as subsystems or components. For a concurrent system, the architecture includes the basic task or process structure. For a distributed system, it includes the organization of hardware in terms of processors and their interconnections. Once the design reaches a sufficient level of specificity, it is translated into an implementation through object-oriented programming. This task can be either relatively straightforward or rather challenging, depending upon the amount of detail in the design and perhaps on other factors, too. When completed, the object-oriented software must coexist with legacy software that is not object-oriented, often requiring that the two types of software communicate with one another. Existing data that are stored in relational databases must be converted to objects in memory when used by an object-oriented application. The staff must be trained to understand and employ object-oriented development and infrastructure. Because they are using new design methods and programming languages, staff members will make mistakes on their first projects, regardless of how much training they receive. This lack of experience also pervades project management. Project leaders who are unversed in the new technology often cannot detect architectural flaws until very late in the development process. In addition, because most estimation models rely on previous data, managers have a difficult time with estimation and planning. Given these costs, why adopt object technology at all? Many organizations take the leap to object technology for non-technical reasons, such as: It must be right. The possible technical benefits include the following: Object-oriented systems can be more flexible. Object-oriented programming languages offer features whose application promotes the extensibility and reuse of the resulting program. Object-oriented development is a relatively seamless process. Analysis and design are a continuous process with a continuous representation. Class-based models closely mimic the real world. Classes model real-world abstractions and, therefore, define a language of the problem domain. This analog with the real world makes object-oriented representations especially useful for modeling a business. Object-oriented technology provides various ways of dealing with complexity. You can decompose a system in different ways, and you can hide implementation complexity behind well-defined interfaces. Object-oriented systems are relatively easy to prototype. Reusable components and encapsulated implementations support prototyping. Greater Flexibility Given that many organizations build the same kinds of software systems again and again, a realistic goal is to avoid building every new system from scratch, aiming instead to build systems that can be extended to incorporate new capabilities, and that can be at least partially reused in other, similar contexts. Object-oriented technology is based on several concepts that, when applied to a design or program, can make the resulting product easier to extend and reuse. Five such features are abstraction, encapsulation, specialization, type promotion, and polymorphism. In general, abstraction is the process of focusing only on the essential characteristics of an entity. A class provides a powerful abstraction mechanism for object-oriented design and programming. A class defines an abstraction in the problem or solution domain. In an order processing application, for example, an Order class

defines the basic properties of each order instance. In particular, it defines both the state data and behavior algorithms that its instances have. Because it defines both state and behavior, a class provides a more powerful abstraction mechanism than a procedure or a data record does. A class also defines both an interface and an underlying implementation. That implementation is invisible to external clients of the class or of its instances. Defining such properties as private to the class is an example of encapsulation. In other words, a class encapsulates its state and behavior. Encapsulation is a mechanism to achieve information hiding. A major advantage of information hiding is that if the information is invisible to the client, you can modify its implementation without affecting the client in any way. Object-oriented programming languages also support class specialization, or the is-a-kind-of relationship between classes. The Automobile class is a specialization of the Vehicle class or, Automobile is a kind of Vehicle , the Router class specializes the Device class, and the Service Order class is a specialization of the Order class. The operational result of the specialization relationship is inheritanceâ€”the specialization or subclass inherits all of the state and behavior of the generalization or superclass. The Automobile class, for example,inherits all of the properties of the Vehicle class. Class specialization allows you to define the properties common to all vehicles in the Vehicle class and then to inherit those properties in the Vehicle specializations such as Automobile, Truck, and so forth. Therefore, a major selling point of specialization is that you can define those common properties once and inherit them where necessary. Hence, you have a single copy of those properties. With multiple copies of a piece of code, any corrections to the code must be introduced in each copy. With class specialization, however, any changes in the superclass properties are propagated to the subclasses through inheritance. Type promotion allows you to treat a specific type of object as a more general type. For example, you can refer to an Automobile instance either as an Automobile or as a more generic Vehicle. This principle is also called substitution. When a client expects to be given a reference to a generic Vehicle, you can substitute any more specific reference such as a reference to an Automobile. Type promotion allows you to write client code that refers only to generic types such as Vehicle. This implies that the client code is unaware of the specific types. Therefore, when you introduce a new specialization of Vehicle, the client code is unaffected. Instances of that new class will be treated by the client code as generic Vehicles. At this point, you may be thinking, "But I want an Automobile to behave like an Automobile, not like a generic Vehicle. Polymorphism simply means that an object still acts like what it really is, regardless of how a client refers to it. When a client tells a Vehicle to do something, the Vehicle will respond appropriately based on what it really is at execution time. If what the client is referencing is actually an Automobile, it will act like an Automobile. On the other hand, if it is a Truck, it will behave like a Truck. The combination of type promotion and polymorphism allows you to write very generic client code that is de-coupled from the existence of particular specializations. As noted previously, your client code that references only generic Vehicles is unaffected by the addition or removal of specializations of Vehicle. Nonetheless, when that code acts on a Vehicle, you still see the correct behavior based on the actual type of the Vehicle. Furthermore, the same generic code can act on any type of Vehicle. A Relatively Seamless Process The object-oriented development process itself is an attraction for many organizations. Compared to many alternative development approaches, which suffer points of discontinuity during the process, object-oriented development is a relatively seamless process. During analysis, you create data flow diagrams that describe how data is transformed by the system. When your analysis is complete and you turn your attention to design, you draw structure charts that indicate what subroutines will exist in your implementation and how those subroutines interact. In other words, you draw one type of diagram during analysis and an entirely different type of diagram during design. The fact that the two models are largely unrelated further exacerbates the problem. Except in a few rare cases, you cannot transform a data flow diagram into a beginning set of structure charts. When you start the design phase, you essentially start anew. In contrast, object-oriented development enjoys representational continuity. As you proceed through analysis and design, you continue to draw the same kinds of diagrams. During high-level analysis, you draw a class diagram, a use case diagram, and perhaps a set of interaction diagrams. Many weeks later, during low-level design, you are still drawing these same diagrams and perhaps a few others. The fact that the analysis and design representations are the same provides one form of continuity. A second form is the continuity of the

model itself; the design is an elaboration of the analysis. That is, the classes and use cases that exist in the initial analysis also appear in the final design. The analysis of an order processing application will include Order, Line Item, and Customer classes. Those classes will continue to appear in the refined analysis and design models. In fact, they will exist in the ultimate implementation! These two forms of representational continuity provide you with a more coherent development process. Mimicking the Real World For some organizations, a major advantage of object technology is the fact that classes model real-world abstractions in an intuitive way. A class can also model both the state and the behavior of its instances. For example, an Order class in an order processing application defines the state that an order has such as an order number and a set of line items , as well as the actions one could perform on an order such as canceling or committing it. A class diagram can, therefore, serve as a model of the business.

3: Object-Oriented Rapid Prototyping : John L. Connell :

1 American Institute of Aeronautics and Astronautics RAPID PROTOTYPING OF AN AIRCRAFT MODEL IN AN OBJECT-ORIENTED SIMULATION P. Sean Kenney.*

4: Object-Oriented Design/Process Concepts

We discuss the connection between rapid prototyping, object-oriented data models, formal specifications, reusable components, and engineering databases kmc/kmc 11/30/09 Call number ocm Camera Canon EOS 5D Mark II.

5: Object-Oriented Rapid Prototyping by John L. Connell (English) Paperback Book | eBay

provides a fast-track explanation of what rapid prototyping really is (as opposed to the folklore of rapid prototyping) -- why it should be object-oriented, how specification methods benefit rapid prototyping, and what kinds of advanced development tools are critical for optimal rapid protoytyping.

6: Object-Oriented Rapid Prototyping

Object-oriented methods and the automated tools and languages that support these methods are rapidly replacing structured methods, CASE tools, information engineering, and RAD approaches to improving software development productivity.

7: Software prototyping - Wikipedia

Abstract: In this paper a new approach for an overall system design is presented. It supports object-oriented system modeling for software components in embedded systems in addition to the well known time-discrete and time-continuous modeling concepts.

OBJECT-ORIENTED RAPID PROTOTYPING pdf

Small town Minnesota, A to Z Along the Inside Passage Makers of the media mind Mesmerism and hypnosis Deng xiaoping economic policies An invitation to health dianne hales 16th edition The Book of Gilding V. 4. The soldier and the werewolf A Victorian courtship When you let go of your past, its time to move forward. Nephrotic syndrome diet plan Post-soul Black cinema National Rhythms, African Roots Of the principles of morals. Kia rio 2003 manual Would you eat your cat The beautiful plants of Kenya. Popular Lyric Writing Describing culture : what it is and where it comes from Thailand; social and economic studies in development Phenomenal Physics and Astronomy On Persephones island How Elephants Lost Their Wings Chattahoochee Nature Center trail Sturgeon, T. The hurkle is a happy beast. Taming The Tabloid Heiress Matthew reilly five greatest warriors The artful astrologer. Aid and the Commonwealth, 1974 Redirect timothy wilson Mostly Mediterranean Timeless Wisdom of the Native Americans Health care history in the united states Federalism in the Making:Contemporary Canadian and German Constitutionalism, National and Transnational The Lecherous University The Watters Mou Historical and philosophical foundations. Ecological integrity and the Darwinian paradigm Alan Holland Consumer and commercial collection deskbook IAMSLIC and international scientific organizations : an approach to trans-Pacific information exchange Ge Texts and calendars II