DOWNLOAD PDF UML BY GRADY BOOCH

Chapter 1 : Grady Booch | Speaker | TED

Grady Booch (born February 27,) is an American software engineer, best known for developing the Unified Modeling Language (UML) with Ivar Jacobson and James calendrierdelascience.com is recognized internationally for his innovative work in software architecture, software engineering, and collaborative development environments.

In spite of Boochs credentials and the fact that this book won an important award, I do not find this book very convincing or clear and would certainly not recommend it as an undergraduate textbook. To me, this book has Grady Booch is a well-known and highly respected software engineer who was one of the founders of the Unified Modelling Language UML used to develop object-oriented software. To me, this book has the feel of having grown out of a collection of simplified case studies loosely tutored by Booch, prefaced by class notes and previous writings quickly cobbled together and in desperate need of masterful editing. The book is organized into three main parts Concepts Method Applications In my opinion, Applications is, hands down, the best part of the book and Concepts the worst. Having taught introductory object-oriented analysis and design courses for many years, I must admit that it is extremely hard to write a good textbook for this subject. You run the risk of falling into vague generalities on the one hand, and on the other, going into too much excruciating and painful detail. Unfortunately by postponing badly needed software examples until the third part of the book, the first two parts of the book were practically condemned to blow away into thin generalities. The first part of the book Concepts includes four chapters 1. Classes and Objects 4. Classification Chapter one starts at a very general level and runs on for some five pages on the structure of plants and animals, the structure of matter and the structure of social organizations, before turning to the complexity of software and superficially touching some key aspects of software development: This chapter reads very much like a general introduction to systems engineering and tries, unsucessfully in my opinion, to provide a motivation and a narrative of how general system ideas eventually trickle down into algorithmic and object-oriented decompositions. Too many important terms and concepts are mentioned almost in passing without anchoring them deeply enough to make them intelligible and usable in a software development context. Chapter 2 The Object Model introduces such key concepts as abstraction, encapsulation, modularity, persistence, coupling, cohesion and typing with lots of hand waving, vague definitions and the occasional conceptual error. If you are an undergraduate student unfamiliar with these terms, you will find this chapter confusing; if you have some grasp of these concepts, I would recommend you skip this chapter. By chapter 3 Classes and objects and 75 pages into the book, the authors finally seem to be ready to come to grips with more substantial stuff, but again they shy away into the realms of the abstract and some humorous if not always pertinent drawings.. One would certainly expect one the founders of UML to provide a thoughtful, clear and precise introduction to UML class diagrams and the key relationships between classes, but unfortunately this is not the case, and again the reader must be satisfied with a rather vague and slipshod introduction. Thus we move into the second part of the book Method which is divided into three chapters, one on notation, one on process and one on pragmatics. The chapter on process distinguishes between agile processes and plan-driven processes and winds up adopting -unsurprisingly- a lightweight version of RUP. The book then distinguishes, rather confusingly between a development macro-process and a micro-process. A micro-process model is presented only for architectural and component analysis and design, its activities consist of identifying object-oriented elements, defining the collaboration between elements, defining the relationship between elements and defining the semantics of the elements their attributes and behaviour. The complete dearth of examples again hampers understanding, forcing most readers to attempt to fast forward into the applications and then come back to these chapters in order to make sense of such recommendations as: The part of the book devoted to method draws to an end with a chapter on pragmatics. Very few of the pragmatics are specifically related to analysis and design, and unfortunately the book again balloons up into generality on such topics as software development management, staffing, documentation and so on. Even the sections which are clearly important for analysis and design, such as Quality Assurance and Metrics manage to be singularly unhelpful. For example, the authors state that: Our primary focus here is on product metrics

sometimes called design metrics that help the development team assess the artifacts of their analysis and design efforts. The developers may come up with a number for, say, weighted methods per class supposing one manages to come up with a reasonable weight scheme -another important omission or the depth of an inheritance tree, but is that number telling him he has a good or a bad design on his hands? Booch et al stay mum on the subject, both in this chapter and in the applications sections where few if any metrics are reported. Thus, ending what ought to be a key section on evaluating object-oriented analysis and design by stating: There is still disagreement about how object-oriented design principles contribute to software quality; consequently, there is still much debate about what constitutes an appropriate set of object-oriented metrics. So after painfully slogging through pages we finally reach the best part of the book, which describe the analysis and design of five applications: In spite of the RUP-like framework, the analysis and designs lack unity, to me they show all the signs of having been tackled by different teams. Thus some of the case studies focus on the problem domain, while others notably the vacation tracking system focuses, in the inception phase, on a specific application environment, thus it starts by stating the design will implemented using Java, Java Server Pages JSP , Tomcat,, J2EE, EJB It is somewhat of a mystery to me why the applications are presented in the aforementioned order, since the most complex system is presented first and the simplest the weather monitoring station software almost last. In my opinion, there are many moot design decisions in most, if not all the case studies. For example, in the arguably simplest case study, the weather monitoring station there is a deep and narrow inheritance tree which derives temperature sensors and pressure sensors from trend sensors. This seems to place the cart before the horse, surely sensors do not capture trends, they capture point data. And surely it is overkill to define a pressure sensor as a subclass of a trend sensor which, in turn is a subclass of a historical sensor, which in turn is a subclass of a calibrating sensor? The train traffic management system is the case study that best states non-functional requirements, while the cryptoanalysis application makes a fine class project. Both the cryptoanalysis application and the vacation tracking systems more interesting aspects include the challenge of implementing rule-based systems. A blackboard-based approach is suggested for the cryptoanalysis framework while a more ad-hoc approach is proposed for the vacation tracking system. The satellite navigation system is the most complex but most original of the applications in an exciting but, for most students, little known domain. The one exception is the very poor corresponding section for the vacation tracking system. Martin, Eric Evans and Craig Larman, amongst others.

DOWNLOAD PDF UML BY GRADY BOOCH

Chapter 2 : Grady Booch - Wikipedia

*The Unified Modeling Language User Guide (2nd Edition) [Grady Booch, James Rumbaugh, Ivar Jacobson] on calendrierdelascience.com *FREE* shipping on qualifying offers. For nearly ten years, the Unified Modeling Language (UML) has been the industry standard for visualizing, specifying.*

Hi Grady, thanks for agreeing to do this interview. Thank you for the opportunity. This was good for everyone as it meant we could all communicate using a single notation. Modelling using UML then seemed to lose the focus it had had previously. Why do you think that was? The problem therein was two-fold. As such, there was rabid innovation in the art and practice of software engineering as we moved from structured methods to object-oriented ones. So, we need to separate methodology from process, for the two are not the same. On the one hand, there was a general recognition that we needed better ways to reason about our systems and that led to this era of visual modeling languages. On the other hand, it was clear that traditional waterfall methods of the 60s and 70s were simply not right for contemporary software development, which led us to the predecessors of agile methods. Waterfall from Wyn Royce, although even Wyn recognized the need for incrementality begat the spiral model from Boehm which begat incremental and iterative approaches, which were always a part of the OOAD processes we at Rational developed. The former we standardized through the OMG, the latter we made mostly open source. It was an unusual period in time. All times are transitional! I concur that the UP was transitional, but the notion of incremental and iterative as comforting was not the reason. So, the notion of these stable points is quite legitimate. Even agile methods have them Process-wise, this is what miniwaterfalls provide. Coming back to UML - there was certainly a degree it ceased to be flavour of the day. To what degree do you think change of emphasis was justified or not justified? Was it perhaps that because UML was new, it became too much of a focus? Sic transit gloria mundi: So it is with the UML. The notation retains modest use, but the underlying concepts live on. That being said, I think that the UML eventually suffered from the standard growing to be overly complex. While I celebrate organizations who were quite successful in that use - such as Siemens, who has used the UML deeply in its telecom products - our intended use case for the UML was more modest: In practice, one should throw away most UML diagrams; in practice, the architecture of even the most complex software-intensive system can be codified in a few dozen UML diagrams, thus capturing the essential design decisions. Much more makes the UML a programming language, for which I certainly never intended it. So, to continue, in many ways XP and its successors was an outgrowth of the changing nature of software development due to the Web. We began to see a dichotomy arise: XP flourished out of the dynamics of building these things at the edge, where experimentation was key, there was no legacy of any material amount, and for which we had domains wherein there was no obvious dominant design, and thus required rapid build and scrap and rework Two general things come to mind. First, we never got the notation for collaborations right. I was trying to find the Right Way to describe patterns, and collaborations were the attempt. Second, component and deployment diagrams needed more maturing. However, my systems-orientation was not well accepted by others. The UML metamodel became grossly bloated, because of the drive to model driven development. I think that complexity was an unnecessary mistake. How would you like to have seen collaboration diagrams? I think we needed something akin to what National Instruments did in LabView for subsystems, but with a bit of special sauce to express cross-cutting concerns. I had hoped the aspect-oriented programming community could have contributed to advances here, but they seemed to have gotten lost in the weeds and forgot the forest. What would you consider to be the most important core techniques of UML, and why? And what modelling techniques have you seen most used in your experience with the wider industry? First, the very notion of objects and classes as meaningful abstractions were a core concept that the presence of the UML helped in transforming the way people approached design. Second, the presence of the UML, I think, helped lubricate the demand pull for design patterns. I was always a great fan of the design patterns Gang of Four, and I hope that the UML and the work I did in this space contributed in some small manner to making their work more visible. So in a sense UML was key to introducing the object-oriented mindset into industry. The UML - and all that surrounded it -

was simply a part of the journey. What notations would you recommend be used on agile projects today. Oh, I rather still like the UML: Was this an improvement over version 1. In an era when open source was just emerging, handing over the UML standard to another body, to put it into the wild, was absolutely the right thing. Having a proprietary language serves no one well, and by making the UML a part of the open community, it had the opportunity to flourish. As I often say, the code is the truth, but it is not the whole truth. There are design decisions and design patterns that transcend individual lines of code, and at those points, the UML adds values. The UML was never intended to replace textual languages.. You could find all these things in the code, but having them in a diagram offers a fresh and simple expression of cross-cutting concerns and essential design decisions. So models in UML can assist in conveying a higher level of thinking about the intent in the code? As I have often said, the history of software engineering is one of rising levels of abstraction and the UML was a step in that direction. We just brought them together in the UP. What were the motivations behind that at the time? The UP reflected our experience at Rational Software - and the experience of our customers - who were building ultra-large systems. We were simply documenting best practices that worked, and that had sound theoretical foundations. To clarify for readers: Consider washing an elephant. All good projects observe a regular iterative heartbeat. What bit you choose is a matter of a attacking risk, b reducing unknowns, and c delivering something executable. Honestly, everything else is just details. The dominant methodological problems that follow are generally not technical in nature, but rather social, and part of the organizational architecture and dynamics. UP was pretty large - especially if you looked to follow it with any degree of rigour. In retrospect is that something that you regret? Or do you think perhaps people got the wrong end of the stick about how UP should be used in practice? The fundamentals of good software engineering can be summarized in one sentence: Honestly, everything beyond that is details or elaboration. Note that there are really three parts here: The UP in its exquisite detail had a role All times are transitional: The four major phases of UP are inception, elaboration, construction and transition. No matter what you name something, these are indeed phases that exist in the cycles of every software-intensive system. One must have a spark of an idea, one must built it, one must grow it, and then eventually you must release it into the wild. Which aspects of UP that you think agile projects could benefit from in particular? Two things come to mind. By views, I mean the concept that one cannot fully understand a system from just one point of view; rather, each set of stakeholders has a particular set of concerns that must be considered. For example, the view of a data analysis is quite different from the view of a network engineer Indeed, every engineering process is an activity of reconciling the forces on a system, and by examining these forces from different points of view, it is possible to build a system with a reasonable separation of concerns among the needs of these stakeholder groups. XP was the right method at the right time led by charismatic - and very effective - developers. This is as it should be: XP worked, and was useful. What do you think of the practises of XP? I think that the dogma of pair programming was overrated. TDD was - and is - still key. The direct involvement of a customer is a great idea in principle but often impractical. Doing the simplest thing possible is absolutely correct, but needs to be tempered with the reality of balancing risk. Finally, the notion of continuous development is absolutely the right thing. On the subject of TDD - do you think comprehensive unit test is good thing - and is it necessary to write all the tests before the code? Or to put it another way, are you sometimes tempted to write the tests afterwards? I believe in moderation in all things, even in the edict of writing ALL tests first. I also believe in the moderation of moderation: Before XP refactoring existing code during later iterations seemed to be completely ignored as a major activity.

Chapter 3 : Grady Booch | Bulldozer00's Blog

Grady is one of the original authors of the Unified Modeling Language (UML) and was also one of the original developers of several of Rational's products. Grady has served as architect and architectural mentor for numerous complex software-intensive systems around the world in just about every domain imaginable.

Quotes[edit] Perhaps the greatest strength of an object-oriented approach to development is that it offers a mechanism that captures a model of the real world. Grady Booch Software Engineering with Ada p. Grady Booch in his talk "The Limits of Software. Diagrams for the Future. After we chatted for a while, he handed me a Fortran [manual]. Where can I find a computer? That was my first programming experience, and I must thank that anonymous IBM salesman for launching my career. It captures decisions and understanding about systems that must be constructed. It is intended for use with all development methods, lifecycle stages, application domains, and media. The modeling language is intended to unify past experience about modeling techniques and to incorporate current software best practices into a standard approach. UML includes semantic concepts, notation, and guidelines. It has static, dynamic, environmental, and organizational parts. It is intended to be supported by interactive visual modeling tools that have code generators and report writers. It is intended to support most existing object-oriented development processes. A design thus represents one point in a potential decision space. A design may be singular representing a leaf decision or it may be collective representing a set of other decisions. As a verb, design is the activity of making such decisions. Given a large set of forces, a relatively malleable set of materials, and a large landscape upon which to play, the resulting decision space may be large and complex. As such, there is a science associated with design empirical analysis can point us to optimal regions or exact points in this design space as well as an art within the degrees of freedom that range beyond an empirical decision; there are opportunities for elegance, beauty, simplicity, novelty, and cleverness. All architecture is design but not all design is architecture. Architecture represents the significant design decisions that shape a system, where significant is measured by cost of change. Grady Booch " On design " cited in: It is the mark of the professional software engineer to know that no such panacea exist. Model Driven Architecture is a style of enterprise application development and integration, based on using automated tools to build system independent models and transform them into efficient implementations. Attributed to Grady Booch in: Attributed to Booch in: Only a few were allowed to commune directly with the machine; all others would give their punched card offerings to the anointed, who would in turn genuflect before their card readers and perform their rituals amid the flashing of lights, the clicking of relays, and the whirring of fans and motors. If the offering was well-received, the anointed would call the communicants forward and in solemn silence hand them printed manuscripts, whose signs and symbols would be studied with fevered brow. Grady Booch " The Computing Priesthood " on ibm. November 14, Object-oriented design: A physician , a civil engineer , and a computer scientist were arguing about what was the oldest profession in the world. The physician remarked, "Well, in the Bible, it says that God created Eve from a rib taken out of Adam. This clearly required surgery, and so I can rightly claim that mine is the oldest profession in the world. This was the first and certainly the most spectacular application of civil engineering. Therefore, fair doctor, you are wrong: The idea that the world could be viewed either in terms of objects or processes was a Greek innovation, and in the seventeenth century, we find Descartes observing that humans naturally apply an object-oriented view of the world. In the twentieth century, Rand expanded upon these themes in her philosophy of objectivist epistemology. More recently, Minsky has proposed a model of human intelligence in which he considers the mind to be organized as a society of otherwise mindless agents. Minsky argues that only through the cooperative behavior of these agents do we find what we call intelligence. Different classes would bear no relationship with one another, since the developer of each provides methods in whatever manner he Chooses. Any consistency across classes is the result of discipline on the part of the programmers. Inheritance makes it possible to define new software in the Same way we introduce any concept to a newcomer, by comparing it with something that is already familiar. An operation is some action one object performs upon another in order to elicit a reaction. When we classify, we seek to group things that have

a common structure or exhibit a common behavior. Managing the Object-Oriented Project. In a quality object-oriented software system, you will find many classes that speak the language of the domain expert p. Journal of Database Management. Does this technology scale? Is it the sole technology worth considering? Is there some better technology we should be using in the future? Possibly, but I am clueless as to what that might be. It is dangerous to make predictions, especially in a discipline that changes so rapidly, but one thing I can say with confidence is that I have seen the future, and it is object-oriented. Lecture September The entire history of software engineering is that of the rise in levels of abstraction. Executable UML is the next logical, and perhaps inevitable, evolutionary step in the ever-rising level of abstraction at which programmers express software solutions. Rather than elaborate an analysis product into a design product and then write code, application developers of the future will use tools to translate abstract application constructs into executable entities. And the code generated from an Executable UML model will be as uninteresting and typically unexamined as the assembler pass of a third generation language compile is today. This shift is made possible by the confluence of four factors: Mellor and Marc J. Balcer define this profile in their book Executable UML: These VEEs, which exist today in a somewhat incipient stage, will someday soon reduce low-level system architectures to near-commodity status.

Chapter 4 : Booch, Rumbaugh & Jacobson, Unified Modeling Language User Guide, The, 2nd Edition | Pe

It was developed by Grady Booch, Ivar Jacobson and James Rumbaugh at Rational Software in , with further development led by them through [1] In UML was adopted as a standard by the Object Management Group (OMG), and has been managed by this organization ever since.

Widely recognized for these and many contributions in the field, he is a popular speaker at technology conferences around the world. James Rumbaugh is one of the leading object-oriented methodologists. He has been working on object-oriented methodology and tools for many years. He developed the DSM object-oriented programming language, the state tree model of control, the OMT object modeling notation, and the Object Modeling Tool graphic editor. Rumbaugh received his Ph. Rumbaugh was one of the inventors of data flow computer architecture. His career has dealt with semantics of computation, tools for programming productivity, and applications using complex algorithms and data structures. Rumbaugh has published journal articles on his work and has spoken at leading object-oriented conferences. He writes a regular column for the Journal of Object-Oriented Programming. Rumbaugh is the lead author of the recent best-selling book Object-Oriented Modeling and Design, published by Prentice Hall. His latest book, OMT Insights: He and his colleagues developed the OMT methodology described in the book based on real-world applications at GE, and they have worked to extend the original methodology. He has taught courses based on the methodology to different audiences around the world, ranging from one-hour seminars to intensive several-day training courses. He has a B. During his career at GE, he worked on a variety of problems, including the design of one of the first time-sharing operating systems, early work in interactive graphics, algorithms for computed tomography, use of parallel machines for fast image generation, VLSI chip design, and finally, object-oriented technology. Jim developed OMTool, an interactive graphical editor for manipulation of object model diagrams. The editor is commercially available. In addition, he led a five-year programming effort producing production-quality software. In addition, he performed personal research. He also led a team of four programmers in implementation. Jim developed and implemented the object-oriented language DSM, combining object-oriented concepts with database concepts and distributed it within GE for use on production applications. The language was heavily used at Calma Corporation and was extensively extended based on user feedback with a preliminary version. He implemented user-interface applications based on this system, including a configuration-management tool and a user-interface generation tool. Jim developed the concept of state trees, a structured extension of finite state machines incorporating a new model of object-oriented control. Later, it was used in the OMTool object editor. Jim also developed the Flow Graph System, a generic interactive graphic system for controlling a network of design engineering jobs, including management of multiple versions of data and coordination of information flow among applications. He received a patent on the underlying concepts. In addition, Jim developed algorithms for the reconstruction of images for computerized tomography using fewer input points and with reduced noise in the reconstructed images. He also developed algorithms for display of three-dimensional images in real time using array processors, and he developed Parallax, a language for programming pipelined array processors. He was one of the three amigos who originally developed the Unified Modeling Language. He is the principal author of five best-selling books on these methods and technologies, in addition to being the coauthor of the two leading books on the Unified Modeling Language. Ivar is a founder of Jaczone AB, where he and his daughter and cofounder, Agneta Jacobson, are developing a ground-breaking new product that includes intelligent agents to support software development. Ivar also founded Ivar Jacobson Consulting IJC with the goal of promoting good software development practices throughout teams worldwide.

DOWNLOAD PDF UML BY GRADY BOOCH

Chapter 5 : Booch, Rumbaugh & Jacobson, Unified Modeling Language User Guide, The | Pearson

In The Unified Modeling Language User Guide, the original developers of the UML--Grady Booch, James Rumbaugh, and Ivar Jacobson--provide a tutorial to the core aspects of the language in a.

History of object-oriented methods and notation Before UML 1. The timeline see image shows the highlights of the history of object-oriented modeling methods and notation. It is originally based on the notations of the Booch method , the object-modeling technique OMT and object-oriented software engineering OOSE , which it has integrated into a single language. They were soon assisted in their efforts by Ivar Jacobson , the creator of the object-oriented software engineering OOSE method, who joined them at Rational in During the same month the UML Partners formed a group, designed to define the exact meaning of language constructs, chaired by Cris Kobryn and administered by Ed Eykholt, to finalize the specification and integrate it with other standardization efforts. The result of this work, UML 1. Recent researchers Feinerer, [13] Dullea et al. Hartmann [15] investigates this situation and shows how and why different transformations fail. The Superstructure that defines the notation and semantics for diagrams and their model elements The Infrastructure that defines the core metamodel on which the Superstructure is based The Object Constraint Language OCL for defining rules for model elements The UML Diagram Interchange that defines how UML 2 diagram layouts are exchanged The current versions of these standards are [19]: UML Superstructure version 2. It continues to be updated and improved by the revision task force, who resolve any issues with the language. Modeling[edit] It is important to distinguish between the UML model and the set of diagrams of a system. The set of diagrams need not completely cover the model and deleting a diagram does not change the model. The model may also contain documentation that drives the model elements and diagrams such as written use cases. UML diagrams represent two different views of a system model: It includes class diagrams and composite structure diagrams. Dynamic or behavioral view: This view includes sequence diagrams , activity diagrams and state machine diagrams. Use cases are a way of specifying required usages of a system. Typically, they are used to capture the requirements of a system, that is, what a system is supposed to do.

Chapter 6 : Books by Grady Booch (Author of Object-Oriented Analysis and Design with Applications)

Grady Booch () in interview "Grady Booch polishes his crystal ball", IBM The Uniï¬•ed Modeling Language (UML) is a general-purpose visual modeling language that is used to specify, visualize, construct, and document the artifacts of a software system.

Chapter 7 : IBM Meet the experts - United States

Grady Booch discusses the growth of software engineering as a discipline with Mark Collins-Cope, the pair covers topics ranging from UML and Unified Process to Programming Languages and the future.

Chapter 8 : Grady Booch - Wikiquote

Grady Booch has 24 books on Goodreads with ratings. Grady Booch's most popular book is Design Patterns: Elements of Reusable Object-Oriented Software.

Chapter 9 : Object-Oriented Analysis and Design with Applications by Grady Booch

Grady Booch, James Rumbaugh, and Ivar Jacobson are the original designers of the Unified Modeling Language and three of the most widely known names in the field of software engineering.