Chapter 4 Software Development Methodologies Programmers subscribe to their chosen methodology of worship! Software development methodologies are frameworks used to plan and control the process of developing software-intensive systems. Over the past several decades, a wide variety of frameworks have evolved each having specific strengths and weaknesses. There is no single best system development strat- egy; good options exist, and no one methodology is good for all types of projects. Forget about silver bullets and one size fits all. As the Software Project Manager, you should have an in-depth understanding of the alternative methodologies and their strengths and weaknesses; then you can proceed with the methodology that best fits your project. Even better, you can select specific features from the methodologies that are applicable to your project and create your own hybrid meth- odology tailored to the needs of your project. Chapter 4 provides a top-level overview of the most pop- ular software development approaches, methodologies, mod- els and standards and is discussed in the following sections: Software Development Process Models: Covers the most commonly used Software Development Process Models including a brief description of those models and their differences (4.1). Software Analysis and Design Models: Covers the most commonly used Software Analysis and Design Methodologies including a brief description of each (4.2). Managing Agile Software Development Projects: Agile is an umbrella term for multiple project management methodologies. This section is focused on Agile/Scrum which is an approach to software development that focuses on iterative goals set by the Product Owner through a backlog developed by the Scrum Team, facilitated by the Scrum Master (4.3). Schedules and Activity Networks: Describes the impor- tance of a formal program Integrated Master Plan (IMP) coupled with a related Integrated Master Schedule (IMS) (4.4). Software Standards: Covers the need for and advan- tages of software standards to provide consistency during system development plus typical software product and testing levels (4.5). As the Software Project Manager you must be involved with the selection of the development methodology because, once a decision is made, it will have a long-term impact to your project. The objective of Chapter 4 is to provide an introduction to that basic understanding providing you with a foundation for that decision. 4.1 Software Development Process Models A Software Development Life Cycle Model should almost always be used to describe, organize, monitor and control soft- ware development activities. There is considerable confusion regarding the differences between some of the development models. Table 4.1 contains, in alphabetical order, a brief descrip- tion of the most commonly used Software Development Process Models. To clarify the intent of each model, they are discussed in more detail at the indicated locations in the Guidebook. Managing the Agile methodology is covered in a separate Section 4.3 since it is important enough to warrant its own section (actually, it is worth more than a section, but this Guidebook is intended to be a compendium and not a huge tome). Your project must select the strategy (or strategies) appropriate to the system you are developing with empha- sis on the availability of requirements. The process selected should be defined in your Software Development Plan (SDP) as described in Subsections 1.5.1 and 5.1.1. More than one 50

Project Management of Large Software-Intensive Systems Table 4.1 Software-Process Model Agile Model Evolutionary Model Incremental Model Iterative Model Prototyping Spiral Model Unified Model Waterfall Model Development Process Models Brief Description The term Agile covers a number of methods and approaches for software development. It is focused on frequent delivery and testing of small portions of the full system. Agile methods are based on the principles of human interaction, where solutions evolve through collaboration of small self- organizing, cross-functional teams. Agile is conceived to be flexible and more quickly able to respond to changes than traditional models (see 4.3). With this model, the software product is developed in a series of builds (blocks or stages) with increasing functionality. The requirements are defined for each evolutionary build as that build is developed. This is a build-a-little, test-a-little development process model that can provide an early operational capability for a portion of the entire system, and it is highly amenable to systems with evolving requirements (see 4.1.1). This model requires that all of the requirements are defined up front; the software product is then developed in a series of builds with cumulative increasing functionality. A portion of the software product is built and testedone small increment at a time. This is also a build-a-little, test-a-little approach that can provide an early operational capability for a portion of the entire see 4.1.2). Not really a software development model but more of a quality improvement approach where a fully developed and delivered systems is periodically updated and improved with each new release of the product. After a system is developed using one of the other development models, the iterative approach is used to improve its quality (see 4.1.3). This development approach involves building an early experimental portion of a system to better understand the requirements and interfaces, to test throughput speeds, develop environment testing, etc. Normally the product produced is built fast, without sufficient documentation, and not designed to be maintainable, so it normally cannot be used as the final product, but there are exceptions (see 4.1.4). The Spiral Model is a risk-driven software development process that has two main features: (1) A cyclic approach that grows a systems functionality and implementation incrementally while focusing on decreasing its degree of risk; and (2) A set of anchor point milestones for insuring stakeholder commitment to acceptable system solutions. Implementations using this model are often done in conjunction with the Evolutionary Model (see 4.1.5). A variation of the Spiral Model is the Unified Process exemplified by the IBM Rational Unified Process (RUP). RUP is an iterative software development framework. However, it is not a single prescriptive process but an adaptable process framework intended to be tailored by selecting elements of the process applicable to each user. It involves an underlying Object-Oriented Model using the Unified Modeling Language (see 4.1.6). A linear sequential software development model that requires all functionality and design requirements to be defined up front and each development activity to be completed before the next activity begins, although some overlap is allowed. The entire software product is not available until the last testing activities are completed (see 4.1.7). Software Development Life Cycle Model may be needed for different types and applications of software used.

4.1.1 Evolutionary Software Acquisition Strategy As defined in the dictionary, the word evolution refers to a process of continuous change from a lower to a higher, more complex, or better state. In that context, evolutionary software development could be considered a generic strategy, applicable to other development methodologies where the functionality of the software product evolves or is refined during the course of each increment or iteration. Key to the success of evolutionary acquisition is continuous customer involvement in the articulation, validation and prioritiza- tion of system requirements over the life of the development process. An example development scenario following the Evolutionary Model is shown in Figure 4.1. Software Development Methodologies Evolutionary Software Development Modelexample. The figure shows three builds (B1-B2-B3) where each build adds more functionality. The requirements are defined or updated for each build. In this example, the requirements analysis for Build-2 starts during the Build-1 design, and the Build-3 requirements analysis starts during the Build-2 design. Software Item Qualification Testing (SIQT) takes place at the end of Build-3 followed by progressive integra- tions with Hardware Items (HI), other Software Items (SI) and then with the other subsystems. The Evolutionary Model is time-tested; consider it seriously for medium to large soft- ware-intensive system developments. NOTE: Increment versus Iterate. Before discussing the Incremental Model and the Iterative Model, their differences should be noted because they are easily confused, and sometimes the two terms are used interchangeably. According to (Cockburn, 2008), increment fundamentally means add onto whereas iterate generally means re-do or rework. They are fun- damentally different models, each serves a different purpose, and they need to be managed differently. Cockburn states: The development process, feature set, and product quality all need constant improvement. Use an incremental strategy, with reflection, to improve the first two. Use an iterative strat- egy, with reflection, to improve the third. Please note that the word iterate, iterative or iteration with a small case i refers to the common meaning of the word: repetition. A capital I is used when referencing the model name. 4.1.2 The Incremental Model The Incremental Model is a method of developing software where the product is developed in a series of builds with increasing functionality. A portion of the software product is built, tested and integratedone small increment at a time. This is a build-a-little, test-a-little approach that can provide an early operational capability for a portion of the entire system. The product is defined as finished when it satisfies all of the requirements that were defined up front. The Incremental Model combines the elements of the Sequential Model (Waterfall; see 4.1.7) with the iterative philosophy of prototyping (see 4.1.4). The biggest problem with this approach is that it presumes you can define all of the software requirements up front. Assuming there will be no changes during development is unrealistic unless there is an edict to freeze the requirements for the current version. A vari- ation of this model could include the provision for some feed- back and changes during the build process. Regression Testing is an important element of the incremental build model. A big advantage of the Incremental Model is that there is a working model of a portion of the system at an early stage of development making it easier to find functional or design flaws. Finding issues at an early stage of development is extremely cost-effective (see Subsection 6.5.1). Aside from needing requirements to be defined up front, a disadvantage of the Incremental Model is that it is typically applicable only to large software development projects because it may be hard to break a small software system into even smaller serviceable increments. 4.1.3 The Iterative Model The Iterative Model can be defined as delivering an entire system and then changing the functionality of the system as needed during each subsequent release of the product. This model involves a cyclical process. A cornerstone of the Iterative Model is the fact that it is often very difficult to know upfront exactly what the customer really wants, so the process builds in as many learning opportunities as possible. The learning you gain from the iterations provides construc- tive information to the next iteration. This learning can come from end-users, Testers or the Developers themselves. 52 Project Management of Large Software-Intensive Systems When the Iterative Model is used, the first delivery is likely to be a core product. The subsequent iterations are the supporting functionalities or the add-on features that the customers want. The product is designed, implemented and tested as a series of incremental deliveries until it has all the functionality required by the users. For example, a shrink wrap company may want to get a product to market within a prescribed time frame, regardless of its inferior quality, and then rely on future releases to fix the bugs and improve its quality. They may call this approach a development model because that is the iterative approach they follow. It can be argued that this is not a development model because the subsequent upgraded improvements could be considered the maintenance phase. Another perspective of the Iterative Model can be explained with the analogy of a portrait painter who provides the customer initially with a very simple sketch; the customer decides the head is facing in the wrong direction so at this point it is easy for the painter to make that adjustment. The painter then adds a little more detail and presents it again to the customer for review. The customer makes recommended changes and the iteration starts over again and continues until the customer approves what will be the final version. For a software project where requirements are unknown, this approach may work, however, for a large or mega system this analogy does not generally apply. The key to successful use of an iterative Software Development Life Cycle is rigorous validation of the new requirements, and verification and testing of each version of the software product against those requirements. As the soft- ware evolves through successive release cycles, tests have to be repeated and extended to verify each version of the soft- ware. Software project managers often consider each itera- tion a separate project. 4.1.4 Prototyping Prototyping is the process of building an early experimental model of a defined system that provides partial function- ality of the final product. Prototyping allows customers to evaluate development approaches and try them out before implementation and to better understand the requirements. Developing prototypes is an extra cost element but provides useful benefits especially for systems having a high level of user interactions such as online systems where users need to fill out forms or go through various screens before data is processed. A prototype can be used very effectively to give the exact look and feel before the actual software is developed. The major software prototyping types used widely are: analysis to build a prototype. Once the actual require- ments are understood, the prototype is discarded, and the actual system is developed with a clearer under- standing of user requirements. Evolutionary Prototyping: Evolutionary prototyping, also called breadboard prototyping, is based on building actual functional prototypes with minimal functional- ity in the beginning but built up incrementally to even- tually deliver the entire system with full functionality. Integration Prototyping: Refers to building independent multiple functional prototypes of the various subsys- tems and then integrating all the available prototype subsystems to form the complete system. The trap with the last two types is the tendency to delay or avoid developing the documentation needed by the final product and the difficulty, and cost, of recreating the docu- mentation later. Prototyping addresses the inability of many customers to specify their exact information needs and the difficulty of system analysts to understand the users environment, by providing the customer with a tentative system for experi- mental purposes at the earliest possible time. The major prob- lem with prototyping is that too often a prototype becomes a quick but inefficient approach to meeting customer require- ments. The code will most often need major reconstruction, and most Programmers do not want to throw away their code so you may end up with patchwork code that is difficult to maintain. Another problem with prototyping is the frequent inter- active interface with the customer during prototyping may lead to requirements scope creep. Every time you believe you are finished there can be new improvements or new func- tionality proposed to make it better (see Gold Plating 5.1.5). You can avoid this if you plan a specific number of iterations and issue a cut-off date for adding new functionality to the system. 4.1.5 The Spiral Model The Spiral Model is a risk-driven Evolutionary Software Development Model that couples the iterative nature of pro- totyping with the controlled and systematic aspects of the Linear Sequential Model where the software is developed in a series of incremental releases. Early releases typically produce a simple prototype, however, during later iterations more complex functionalities are added. The Spiral Model is a risk-driven Software Development Process Model that has two main features: Throwaway/Rapid Prototyping: Throwaway prototyp- ing is also called rapid or close-ended prototyping. This type of prototyping needs minimal requirement A cyclic approach that grows a systems functionality and implementation incrementally while focusing on decreasing its degree of risk

Introduction to LaTeX

Author's Name

December 29, 2020

Abstract

The abstract text goes here.

1 Introduction

Here is the text of your introduction.

$$\alpha = \sqrt{\beta} \tag{1}$$

1.1 Subsection Heading Here

Write your subsection text here.

2 Conclusion

Write your conclusion here.