Software Development Processes

张启豪 13211244
Zhejiang Normal University
Software Project Management

Abstract

This short report gives a brief and concise explanation to why Software Development Processes is important and valuable. The report has been formatted, so whether or not you are currently a software project manager, actively working on a project team, or completely in the dark about this mysterious field. This report has something for you. This is an ambitious report, but not unrealistic, and we know that software development processes is a little bit of an art and involves a lot of practice. There is so much more to know than this report is able to present - however, it gives a taste of the subject and a high-level view of core components. It's mot that we necessarily excluded anything from this report, but it would be unrealistic to try and cover every possible aspect in a couple of pages.
Keywords: software engineering, development processes

1 Introduction

In software engineering, a software development methodology(also known as a system development methodology, software development life cycle, software development process,software process) is a splitting of software development work into distinct phases (or stages) containing activities with the intent of better planning and management. It is often considered a subset of the systems development life cycle. The methodology may include the pre-definition of specific deliverables and artifacts that are created and completed by a project team to develop or maintain an application.

Common methodologies

include waterfall, prototyping,it erative and incremental development,spiral development, rapid application development, extreme programming and various types of agile methodology. Some people consider a life-cycle "model" a more general term for a category of methodologies and a software development "process" a more specific term to refer to a specific process chosen by a specific organization. For example, there are many specific software development processes that fit the spiral life-cycle model.

2 History

The software development methodology (also known as SDM) framework didn't emerge until the 1960s. According to Elliott (2004) the systems development life cycle(SDLC) can be considered to be the oldest formalized methodology framework for building information systems. The main idea of the SDLC has been "to pursue the development of information systems in a very deliberate, structured and methodical way, requiring each stage of the life cycle--from inception of the idea to delivery of the final system--to be carried out rigidly and sequentially" within the context of the framework being applied. The main target of this methodology framework in the 1960s was "to develop large scale functional business systems in an age of large scale business conglomerates. Information systems activities revolved around heavy data processing and number crunching routines".

Methodologies, processes, and frameworks range from specific proscriptive steps that can be used directly by an organization in day-to-day work, to flexible frameworks that an organization uses to generate a custom set of steps tailored to the needs of a specific project or group. In some cases a "sponsor" or "maintenance" organization distributes an official set of documents that describe the process.

3 Approaches

Several software development approaches have been used since the origin of information technology, in two main categories. Typically an approach or a combination of approaches is chosen by management or a development team.

"Traditional" methodologies such as waterfall that have distinct phases are sometimes known as software development life cycle (SDLC) methodologies, though this term could also be used more generally to refer to any methodology. A "life cycle" approach with distinct phases is in contrast to Agile approaches which define a process of iteration, but where design, construction, and deployment of different pieces can occur simultaneously.

Prototyping

Software prototyping, is the development approach of activities during software development, the creation of prototypes, i.e., incomplete versions of the software program being developed.

The basic principles are:

Not a standalone, complete development methodology, but rather an approach to handle selected parts of a larger, more traditional development methodology (i.e. incremental, spiral, or rapid application development (RAD)).

Attempts to reduce inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process.

User is involved throughout the development process, which increases the likelihood of user acceptance of the final implementation.

Small-scale mock-ups of the system are developed following an iterative modification process until the prototype evolves to meet the users' requirements.

While most prototypes are developed with the expectation that they will be discarded, it is possible in some cases to evolve from prototype to working system.

A basic understanding of the fundamental business problem is necessary to avoid solving the wrong problems.

4 Formal Methods

Formal methods are mathematical approaches to solving software (and hardware) problems at the requirements, specification, and design levels. Formal methods are most likely to be applied to safety-critical or security-critical software and systems, such as avionics software. Software safety assurance standards, such asDO-178B, DO-178C, and Common Criteria demand formal methods at the highest levels of categorization.

For sequential software, examples of formal methods include the B-Method, the specification languages used in automated theorem proving, RAISE, and the Z notation.

Formalization of software development is creeping in, in other places, with the application of Object Constraint Language (and specializations such as Java Modeling Language) and especially with model-driven architecture allowing execution of designs, if not specifications.

For concurrent software and systems, Petri nets, process algebra, and finite state machines (which are based on automata theory - see also virtual finite state machine or event driven finite state machine) allow executable software specification and can be used to build up and validate application behavior.

Another emerging trend in software development is to write a specification in some form of logic—usually a variation of first-order logic (FOL)—and then to directly execute the logic as though it were a program. The OWL language, based on Description Logic (DL), is an example. There is also work on mapping some version of English (or another natural language) automatically to and from logic, and executing the logic directly. Examples are Attempto Controlled English, and Internet Business Logic, which do not seek to control the vocabulary or syntax. A feature of systems that support bidirectional English-logic mapping and direct execution of the logic is that they can be made to explain their results, in English, at the business or scientific level.

References

1. Barry Boehm (1996., "A Spiral Model of Software Development and Enhancement". In: *ACM SIGSOFT Software Engineering Notes* (ACM) 11(4):14-24, August 1986
2. Jump up^ Richard H. Thayer, Barry W. Boehm (1986). *Tutorial: software engineering project management*. Computer Society Press of the IEEE. p.130
3. Jump up^ Barry W. Boehm (2000). *Software cost estimation with Cocomo II: Volume 1*.
4. ^ Jump up to:[a] [b] Whitten, Jeffrey L.; Lonnie D. Bentley, Kevin C. Dittman. (2003). *Systems Analysis and Design Methods*. 6th edition. ISBN 0-256-19906-X.
5. Jump up^ Kent Beck, Extreme Programming, 2000.