Computer Science Journal of Moldova, vol.**12**, no.1(34), 2004

Software Development Processes

V. Iacoban

Abstract

For many years now, software process improvement (SPI) has been recognized as an effective way for companies to improve the quality of the software they produce and to increase the efficiency of the development process. Much work has gone into developing and selling improvement paradigms, assessment methods, modeling languages, tools and technologies. This paper will list many of existing processes, frameworks, paradigms, presenting the advantages and drawbacks of each of them. Synthesizing, a requirements list will be prepared to define a software process improvement approach. Ideally, the approach should be practical, applicable, efficient, and allowing to integrate existing software developing technologies.

1 Introduction

Software organizations in the world employ nearly 7 million engineers and generate annual revenue of more than $600 billion, an amount that has been growing at an annual rate exceeding 25% for the past three years. About half of this revenue is generated by the software products industry, which builds general–purpose software products, and roughly half is generated by the software service industry, which builds customized software products for clients. The software industry today is viewed as one of the most promising industry segments and one holding tremendous future potential.

If we consider developing a software product as a project, then the software industry constantly focuses on project execution. Assuming that the average software project consumes about 7 person–years of effort (during which a software product consisting of 20,000 to

80,000 lines of code can be built), then the software industry, with its more than 7 million engineers, executes in excess of 1 million software projects per year! Clearly, executing software projects efficiently is of paramount importance to the software industry as a whole.

The processes used for executing a software project clearly have a major effect on the quality of the software produced and the productivity achieved in the project. Consequently, there is a need to evaluate processes used in an organization for executing software projects and to improve them.

Software process improvement involves applying defined, systematic and repeatable approaches to software development. Few software development company directors would argue that software process improvements would not improve their company's profitability but, in most cases, they cannot afford to embark on high-cost, large-scale style software improvement where tangible improvement may not be seen for years.

In addition, there are many approaches and technologies for software process improvement that have already been developed and proved to be effective. The approach should allow companies to utilize for example, ISO 9001, CMM, Trillium, SPICE, or BOOTSTRAP or any other software process improvement technology if and when they believe them to be relevant to their improvement goals.

Next sections will describe in more details existing software improvement technologies and frameworks presenting its' advantages and drawbacks. In conclusion, last section will give a list of requirements which any manager of a software development company would like to be implemented in their development process.

Processes describe how things are done – specifically, what should be done, what is needed to do it, how it should be done and who should do it. Everyone has processes and everyone naturally does process improvement, evidenced by the fact that we get faster and better at doing things we do often and repetitively. Manufacturing has benefited from explicit process improvement for a long time. The goal now is for software development to leverage process improvement to increase quality and productivity. To do this many approaches and technologies

have been developed. Presented below are just a few approaches that are central to development of software development processes.

2 Capability Maturity Model (CMM)

One of the most famous approaches to software process improvement is the Software Engineering Institute's Capability Maturity Model (CMM). Originally commissioned by the US Department of Defence (DoD) as a way to ascertain which software/system providers were capable of delivering quality software on time and within budget, the creators of CMM deemed this capability to be directly related to "process maturity". That is, how advanced and adhered to are the company's software processes? The CMM defines "levels" of maturity to which maturity groups (e.g. companies, departments, groups) can be certified. Groups must achieve a specified level of maturity before they can be considered for DoD contracts (many other software commissioning groups also use CMM certification to select developers for contracts). To determine their level of maturity, groups undergo process assessments, in which assessors investigate the process of the group and how closely they are followed. Each level defines strict criteria to which the group must adhere in order to gain certification.

2.1 Immature Versus Mature Software Organizations

Setting sensible goals for process improvement requires an understanding of the difference between immature and mature software organizations. In an immature software organization, software processes are generally improvised by practitioners and their management during the course of the project. Even if a software process has been specified, it is not rigorously followed or enforced. The immature software organization is reactionary, and managers are usually focused on solving immediate crises (better known as fire fighting). Schedules and budgets are routinely exceeded because they are not based on realistic estimates. When hard deadlines are imposed, product functionality and quality are often compromised to meet the schedule.

In an immature organization, there is no objective basis for judging product quality or for solving product or process problems. Therefore, product quality is difficult to predict. Activities intended to enhance quality such as reviews and testing are often curtailed or eliminated when projects fall behind schedule.

On the other hand, a mature software organization possesses an organization-wide ability for managing software development and maintenance processes. The software process is accurately communicated to both existing staff and new employees, and work activities are carried out according to the planned process. The processes mandated are usable and consistent with the way the work actually gets done. These defined processes are updated when necessary, and improvements are developed through controlled pilot-tests and/or cost benefit analyzes. Roles and responsibilities within the defined process are clear throughout the project and across the organization.

In a mature organization, managers monitor the quality of the software products and the process that produced them. There is an objective, quantitative basis for judging product quality and analyzing problems with the product and process. Schedules and budgets are based on historical performance and are realistic; the expected results for cost, schedule, functionality, and quality of the product are usually achieved. Generally, a disciplined process is consistently followed because all of the participants understand the value of doing so, and the necessary infrastructure exists to support the process.

2.2 Fundamental Concepts Underlying Process Maturity

A **software process** can be defined as a set of activities, methods, practices, and transformations that people use to develop and maintain software and associated products (e.g., project plans, design documents, code, test cases, and user manuals). As an organization matures, the software process becomes better defined and more consistently implemented throughout the organization.

Software process capability describes the range of expected re-

sults that can be achieved by following a software process. The software process capability of an organization provides one means of predicting the most likely outcomes to be expected from the next software project the organization undertakes.

Software process performance represents the actual results achieved by following a software process. Thus, software process performance focuses on the results achieved, while software process capability focuses on results expected.

Software process maturity is the extent to which a specific process is explicitly defined, managed, measured, controlled, and effective. Maturity implies a potential for growth in capability and indicates both the richness of an organization's software process and the consistency with which it is applied in projects throughout the organization.

As a software organization gains in software process maturity, it institutionalizes its software process via policies, standards, and organizational structures. Institutionalization entails building an infrastructure and a corporate culture that supports the methods, practices, and procedures of the business so that they endure after those who originally defined them have gone.

2.3 The Five Levels of Software Process Maturity

A **maturity level** is a well-defined evolutionary plateau toward achieving a mature software process. Each maturity level comprises a set of process goals that, when satisfied, stabilize an important component of the software process. Achieving each level of the maturity framework establishes a different component in the software process, resulting in an increase in the process capability of the organization.

Maturity Levels 2 through 5 can be characterized through the activities performed by the organization to establish or improve the software process, by activities performed on each project, and by the resulting process capability across projects. A behavioral characterization of Level 1 is included to establish a base of comparison for process improvements at higher maturity levels.

Level 1 – The Initial Level

At the Initial Level, the organization typically does not provide a stable environment for developing and maintaining software. Such organizations frequently have difficulty making commitments that the staff can meet with an orderly engineering process, resulting in a series of crises. During a crisis, projects typically abandon planned procedures and revert to coding and testing. Success depends entirely on having an exceptional manager and a seasoned and effective software team. Occasionally, capable and forceful software managers can withstand the pressures to take shortcuts in the software process; but when they leave the project, their stabilizing influence leaves with them. Even a strong engineering process cannot overcome the instability created by the absence of sound management practices.

In spite of this ad hoc, even chaotic, process, Level 1 organizations frequently develop products that work, even though they may be over the budget and schedule. Success in Level 1 organizations depends on the competence and heroics of the people in the organization and cannot be repeated unless the same competent individuals are assigned to the next project. Thus, at Level 1, capability is a characteristic of the individuals, not of the organization.

Level 2 - The Repeatable Level

At the Repeatable Level, policies for managing a software project and procedures to implement those policies are established. Planning and managing new projects is based on experience with similar projects. Process capability is enhanced by establishing basic process management discipline on a project by project basis. An effective process can be characterized as one which is practiced, documented, enforced, trained, measured, and able to improve.

Projects in Level 2 organizations have installed basic software management controls. Realistic project commitments are based on the results observed on previous projects and on the requirements of the current project. The software managers for a project track software costs, schedules, and functionality; problems in meeting commitments

are identified when they arise. Software requirements and the work products developed to satisfy them are base-lined, and their integrity is controlled. Software project standards are defined, and the organization ensures they are faithfully followed. The software project works with its subcontractors, if any, to establish a customer-supplier relationship.

Processes may differ between projects in a Level 2 organization. The organizational requirement for achieving Level 2 is that there are policies that guide the projects in establishing the appropriate management processes.

The software process capability of Level 2 organizations can be summarized as disciplined because planning and tracking of the software project is stable and earlier successes can be repeated. The project's process is under the effective control of a project management system, following realistic plans based on the performance of previous projects.

Level 3 - The Defined Level

At the Defined Level, the standard process for developing and maintaining software across the organization is documented, including both software engineering and management processes, and these processes are integrated into a coherent whole. This standard process is referred to throughout the CMM as the organization's standard software process. Processes established at Level 3 are used (and changed, as appropriate) to help the software managers and technical staff perform more effectively. The organization exploits effective software engineering practices when standardizing its software processes. There is a group that is responsible for the organization's software process activities, e.g., a software engineering process group. An organization-wide training program is implemented to ensure that the staff and managers have the knowledge and skills required to fulfill their assigned roles.

Projects tailor the organization's standard software process to develop their own defined software process, which accounts for the unique characteristics of the project. This tailored process is referred to in the CMM as the project's defined software process. A defined software

process contains a coherent, integrated set of well-defined software engineering and management processes. A well-defined process can be characterized as including readiness criteria, inputs, standards and procedures for performing the work, verification mechanisms (such as peer reviews), outputs, and completion criteria. Because the software process is well defined, management has good insight into technical progress on all projects.

The software process capability of Level 3 organizations can be summarized as standard and consistent because both software engineering and management activities are stable and repeatable. Within established product lines, cost, schedule, and functionality are under control, and software quality is tracked. This process capability is based on a common, organization-wide understanding of the activities, roles, and responsibilities in a defined software process.

Level 4 - The Managed Level

At the Managed Level, the organization sets quantitative quality goals for both software products and processes. Productivity and quality are measured for important software process activities across all projects as part of an organizational measurement program. An organization-wide software process database is used to collect and analyze the data available from the projects' defined software processes. Software processes are instrumented with well-defined and consistent measurements at Level 4. These measurements establish the quantitative foundation for evaluation the projects' software processes and products.

Projects achieve control over their products and processes by narrowing the variation in their process performance to fall within acceptable quantitative boundaries. Meaningful variations in process performance can be distinguished from random variation (noise), particularly within established product lines. The risks involved in moving up the learning curve of a new application domain are known and carefully managed.

The software process capability of Level 4 organizations can be summarized as being quantifiable and predictable because the process is

measured and operates within measurable limits. This level of process capability allows an organization to predict trends in process and product quality within the quantitative bounds of these limits. Because the process is both stable and measured, when some exceptional circumstance occurs, the "special cause" of the variation can be identified and addressed. When the known limits of the process are exceeded, action is taken to correct the situation. Software products are of predictably high quality.

Level 5 - The Optimizing Level

At the Optimizing Level, the entire organization is focused on continuous process improvement. The organization has the means to identify weaknesses and strengthen the process pro-actively, with the goal of preventing the occurrence of defects. Data on the effectiveness of the software process is used to perform cost benefit analyzes of new technologies and proposed changes to the organization's software process. Innovations that exploit the best software engineering practices are identified and transferred throughout the organization.

Software project teams in Level 5 organizations analyze defects to determine their causes. Software processes are evaluated to prevent known types of defects from recurring, and lessons learned are disseminated to other projects.

There is chronic waste, in the form of rework, in any system simply due to random variation. Waste is unacceptable; organized efforts to remove waste result in changing the system, e.g., improving the process by changing "common causes" of inefficiency to prevent the waste from occurring. While this is true of all the maturity levels, it is the focus of Level 5.

The software process capability of Level 5 organizations can be characterized as continuously improving because Level 5 organizations are continuously striving to improve the range of their process capability, thereby improving the process performance of their projects. Improvement occurs both by incremental advancements in the existing process and by innovations using new technologies and methods. Technology

and process improvements are planned and managed as ordinary business activities.

2.4 Process Capability and the Prediction of Performance

These levels are designed to guide the process improvement, but they do not offer detailed instructions on how to achieve this improvement (they do not, for example, provide actual process models that can be adopted). This idea of levels offers a phased approach, but each level is so comprehensive it often takes groups years to move from level to level in CMM.

Achieving higher levels of software process maturity is incremental and requires a long-term commitment to continuous process improvement. Software organizations may take ten years or more to build the foundation for, and a culture oriented toward, continuous process improvement. Although a decade-long process improvement program is foreign to most companies, this level of effort is required to produce mature software organizations.

The CMM is not a silver bullet and does not address all of the issues that are important for successful projects. For example, the CMM does not currently address expertise in particular application domains, advocate specific software technologies, or suggest how to select, hire, motivate, and retain competent people. Although these issues are crucial to a project's success, they have not been integrated into the CMM.

3 Rational Unified Process (RUP)

The Rational Unified Process is a software engineering process. It provides a disciplined approach to assigning tasks and responsibilities within a development organization. The Rational Unified Process is also a process framework that can be adapted and extended to suit the needs of an adopting organization. The Rational Unified Process

captures many of the best practices in modern software development in a form that is suitable for a wide range of projects and organizations.

A process describes *who* is doing *what*, *how*, and *when*. The Rational Unified Process is represented using four primary modeling elements:

Workers – the *who*;

Activities – the *how*;

Artifacts – the *what*;

Workflows – the *when*;

3.1 Workers

The central concept in the process is that of a worker. A worker defines the behavior and responsibilities of an individual or a group of individuals working together as a team. The behavior is expressed in terms of activities the worker performs, and each worker is associated with a set of cohesive activities. "Cohesive" in this meaning defines those activities best performed by one person. The responsibilities of each worker are usually expressed in relation to certain artifacts that the worker creates, modifies, or controls.

It's helpful to think of a worker as a "hat" that an individual can wear during the project. One person may wear many hats. This distinction is important because it is natural to think of a worker as the individual or the team, but in the Rational Unified Process the term worker refers to the roles that define how the individuals should do the work. A worker performs one or more roles and is the owner of a set of artifacts. Another way to think of a worker is as a part in a play-a part that can be performed by many actors.

Workers are not individuals; instead, they describe how individuals should behave in the business and the responsibilities of each individual. Individual members of the software development organization wear different hats, or play different parts or roles.

3.2 Activities

Workers have activities, which define the work they perform. An activity is a unit of work that an individual in that role may be asked to perform, and that produces a meaningful result in the context of the project. The activity has a clear purpose, usually expressed in terms of creating or updating artifacts, such as a model, a class, or a plan. Every activity is assigned to a specific worker.

The granularity of an activity is generally a few hours to a few days. It usually involves one worker and affects one or only a small number of artifacts. An activity should be usable as an element of planning and progress; if it is too small, it will be neglected, and if it is too large, progress will have to be expressed in terms of the activity's parts.

Activities may be repeated several times on the same artifact, especially, from one iteration to another as the system is refined and expanded. Repeated activities may be performed by the same worker but not necessarily by the same individual.

In object-oriented terms, the worker is an active object, and the activities that the worker performs are operations performed by that object.

3.3 Artifacts

Activities have input and output artifacts. An artifact is a piece of information that is produced, modified, or used by a process. Artifacts are the tangible products of the project: the things the project produces or uses while working toward the final product. Artifacts are used as input by workers to perform an activity and they are the result or output of such activities. In object-oriented design terms, just as activities are operations on an active object (the worker), artifacts are the parameters of these activities.

Artifacts can also be composed of other artifacts. For example, the design model contains many classes; the software development plan contains several other plans: a staffing plan, a phase plan, a metrics plan, iteration plans, and so on.

Artifacts are very likely to be subject to version control and configuration management. Sometimes, this can be achieved only by versioning the container artifact when it is not possible to do it for the elementary, contained artifacts. For example, you may control the versions of a whole design model or design package and not of the individual classes they contain.

Typically, artifacts are not documents. Many processes place an excessive focus on documents and in particular on paper documents. The Rational Unified Process discourages the systematic production of paper documents. The most efficient and pragmatic approach to managing project artifacts is to maintain the artifacts within the appropriate tool used to create and manage them. When necessary, you can generate documents (snapshots) from these tools on a just-in-time basis.

Artifacts are the responsibility of a single worker, to promote the idea that every piece of information must be the responsibility of a specific person. Even though one person may "own" the artifact, many people may use this artifact, perhaps even updating it if they have been given permission.

3.4 Workflows

A mere enumeration of all workers, activities, and artifacts does not quite constitute a process. We need a way to describe meaningful sequences of activities that produce some valuable result and to show interactions between workers. A workflow is a sequence of activities that produces a result of observable value. In UML terms, a workflow can be expressed as a sequence diagram, a collaboration diagram, or an activity diagram.

3.5 Phases

From a management perspective, the software lifecycle of the Rational Unified Process (RUP) is decomposed over time into four sequential phases, each concluded by a major milestone; each phase is essentially a

span of time between two major milestones. All phases are not identical in terms of schedule and effort.

Phase: Inception

The overriding goal of the inception phase is to achieve concurrence among all stakeholders on the lifecycle objectives for the project. The inception phase is of significance primarily for new development efforts, in which there is significant business and requirement's risks, which must be addressed before the project can proceed. For projects focused on enhancements to an existing system, the inception phase is shorter, but is still focused on ensuring that the project is both worth doing and possible to be done.

At the end of the inception phase comes the first major project milestone or **Lifecycle Objectives Milestone**. At this point, the lifecycle objectives of the project are examined, and it is decided wether to proceed with the project or to cancel it.

Phase: Elaboration

The goal of the elaboration phase is to baseline the architecture of the system to provide a stable basis for the bulk of the design and implementation effort in the construction phase. The architecture evolves out of a consideration of the most significant requirements (those that have a great impact on the architecture of the system) and an assessment of risk. The stability of the architecture is evaluated through one or more architectural prototypes.

At the end of the elaboration phase comes the second important project milestone, the **Lifecycle Architecture Milestone**. At this point, the detailed system objectives and scope, the choice of architecture, and the resolution of the major risks are examined.

Phase: Construction

The goal of the construction phase is to clarify the remaining requirements and complete the development of the system based upon the

base-lined architecture. The construction phase is, in some sense, a manufacturing process, where emphasis is placed on managing resources and controlling operations to optimize costs, schedules, and quality. In this sense the management mindset undergoes a transition from the development of intellectual property during inception and elaboration, to the development of deploy-able products during construction and transition.

At the **Initial Operational Capability Milestone**, the product is ready to be handed over to the Transition Team. All functionality has been developed and all alpha testing (if any) has been completed. In addition to the software, a user manual has been developed, and there is a description of the current release.

Phase: Transition

The focus of the Transition Phase is to ensure that software is available for its end users. The Transition Phase can span several iterations, and includes testing the product in preparation for release, and making minor adjustments based on user feedback. At this point in the lifecycle, user feedback should focus mainly on fine-tuning the product, configuring, installing and usability issues, all the major structural issues should have been worked out much earlier in the project lifecycle.

By the end of the Transition Phase lifecycle objectives should have been met and the project should be in a position to be closed out. In some cases, the end of the current life cycle may coincide with the start of another lifecycle on the same product, leading to the next generation or version of the product. For other projects, the end of Transition may coincide with a complete delivery of the artifacts to a third party who may be responsible for operations, maintenance and enhancements of the delivered system.

At the end of the transition phase comes the fourth important project milestone, the **Product Release Milestone**. At this point, it is decided if the objectives were met, and if another development cycle should be started. In some cases this milestone may coincide with the end of the inception phase for the next cycle.

At the Product Release Milestone, the product is in production and the post-release maintenance cycle begins. This may involve starting a new cycle, or some additional maintenance release.

Looking from the practical point of view, the Rational Unified Process was designed as a development process to complement object-oriented development using Unified Modeling Language. Currently offered as an electronic (web-based) guide RUP incorporates "best practice" lessons from object-oriented development projects. It does not define an improvement paradigm other than purchase, install and use the RUP. The RUP can be valuable for companies engaging in object-oriented style development but it does have the following drawbacks:

1. It defines a process rather than an improvement paradigm i.e. it does not define how the process should be introduced, monitored and evolved to obtain maximum benefit;

2. It is a predefined process which may or may not "fit" a company's culture;

3. It is a predefined process which is being used by many companies, so after it has been installed and followed, there is little scope for evolving it to gain maximum competitive advantage.

One of the more recent examples is Extreme Programming (XP) which specifies very short, iterative cycles of development consisting of requirements definition, design and implementation (along with rework). Adopting a development method is sure to improve your process, however they tend to be focused on requirements, design and implementation issues, not necessarily management, documentation and quality processes that can have a large effect on productivity and quality. They also do not offer an improvement paradigm to install, monitor and improve the process.

4 Conclusion

There are many standards for software development that can be applied to effectively improve software processes. Generally, they fall into

two categories. Firstly, the CMM style improvement paradigms such as SPICE, Bootstrap and Tickit. These approaches meet the same problems as CMM when applied to small scale software development. Secondly, comprehensive standards for all or particular parts of the development process such as ISO 9000 / 9001 or IEEE. These kinds of standards can be very helpful in defining a process, but do not provide improvement paradigms.

As well as improvement paradigms and process standards or methods, many technologies to support process improvement have been developed. These include languages and tools for documenting, simulating, communicating and executing processes, methods for carrying out process assessments to determine process maturity of companies and process measures for monitoring improvement in projects. Many of these technologies are well advanced and should be utilized fully in any software process improvement paradigm.

To sum up, the problem for software developing companies is how to use all of this technology, experience and knowledge to effect process improvement in their particular organizations. Obviously, a different paradigm for improvement is needed to that of the large scale maturity based approaches. The important requirements for this paradigm are that it should:

1. Be effective and produce good results (e.g. shorter cycle times, earlier error detection, less post-release defects, good return on investment etc.)

2. Be incremental so that it can be initialized with a relatively small investment and then incrementally extended. Increments should be short and ideally match project cycles.

3. Give fast, tangible results so that its continuation can be justified.

4. Utilize existing technologies such as those outlined above to get maximum advantage from the work that has already been done in this field.

Having such a process improvement mechanism in place would help

to take full advantage of existing technologies for software development processes.

References

[1] Pankaj Jalote, "*CMM in Practice*", Addison-Wesley Pub Co; 1st edition (October 22, 1999) .

[2] L. Scott, R. Jeffery, L. Carvalho, J. D'Ambra and P. Rutherford, "*Practical Software Process Improvement - The IMPACT Approach*", Proceedings 2001, Australian Software Engineering Conference, pp. 182–189, IEEE Computer Society Press, 2001.

[3] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, Charles V. Weber, "*The Capability Maturity Model for Software*", Software Engineering Institute.

[4] Mark C. Paulk, "*Extreme Programming from a CMM Perspective*", IEEE Software, November/December 2001.

[5] Gary Pollice, "*Using the Rational Unified Process for Small Projects: Expanding Upon eXtreme Programming*", A Rational Software Whitepaper.

[6] Scott W. Ambler, Larry C. Constantine, "*Best Practices in Implementing Unified Process*", CMP Books.

Victor Iacoban,

Institute of Mathematics and Computer Science,
Academy of Sciences of Moldova,
str.Academiei,5, Chisinau, MD 2028,
Republic of Moldova.
E–mail: *victor_iacoban@yahoo.com*

Received December 25, 2003