

UPGRADE is the European Journal for the Informatics Professional, published bimonthly at <<http://www.upgrade-cepis.org>>

UPGRADE is the anchor point for UPENET (UPGRADE European NETWORK), the network of CEPIs member societies' publications, that currently includes the following ones:

- Mondo Digitale, digital journal from the Italian CEPIs society AICA
- Novática, journal from the Spanish CEPIs society ATI
- Piroforiki, journal from the Cyprus CEPIs society CCS
- Pro Dialog, journal from the Polish CEPIs society PTI-PIPS

Publisher

UPGRADE is published on behalf of CEPIs (Council of European Professional Informatics Societies, <<http://www.cepis.org/>>) by Novática <<http://www.ati.es/novatica/>>, journal of the Spanish CEPIs society ATI (Asociación de Técnicos de Informática <<http://www.ati.es/>>).

UPGRADE is also published in Spanish (full issue printed, some articles online) by Novática, and in Italian (abstracts and some articles online) by the Italian CEPIs society ALSI <<http://www.alsi.it/>> and the Italian IT portal Tecnoteca <<http://www.tecnoteca.it/>>.

UPGRADE was created in October 2000 by CEPIs and was first published by Novática and INFORMATIK/INFORMATIQUE, bimonthly journal of SVI/FSI (Swiss Federation of Professional Informatics Societies, <<http://www.svifsi.ch/>>).

Editorial Team

Chief Editor: Rafael Fernández Calvo, Spain, <rfoalvo@ati.es>
Associate Editors:

- François Louis Nicolet, Switzerland, <nicolet@acm.org>
- Roberto Carniel, Italy, <carniel@dgt.uniud.it>
- Zakaria Maamar, Arab Emirates, <Zakaria.Maamar@zu.ac.ae>
- Soraya Kouadri Mostéfaoui, Switzerland, <soraya.kouadrimostefaoui@unifr.ch>

Editorial Board

Prof. Wolfried Stucky, CEPIs Past President
Prof. Nello Scarabottolo, CEPIs Vice President
Fernando Piera Gómez and Rafael Fernández Calvo, ATI (Spain)
François Louis Nicolet, SI (Switzerland)
Roberto Carniel, ALSI – Tecnoteca (Italy)

UPENET Advisory Board

Franco Filippazzi (Mondo Digitale, Italy)
Rafael Fernández Calvo (Novática, Spain)
Panicos Masouras (Piroforiki, Cyprus)
Andrzej Marciniak (Pro Dialog, Poland)

English Editors: Mike Andersson, Richard Butchart, David Cash, Arthur Cook, Tracey Darch, Laura Davies, Nick Dunn, Rodney Fennemore, Hilary Green, Roger Harris, Michael Hird, Jim Holder, Alasdair MacLeod, Pat Moody, Adam David Moss, Phil Parkin, Brian Robson.

Cover page designed by Antonio Crespo Foix, © ATI 2004

Layout: Pascale Schürmann

Editorial correspondence: see "Editorial Team" above

Advertising correspondence: <novatica@ati.es>

Upgrade Newsletter available at

<<http://www.upgrade-cepis.org/pages/editinfo.html#newsletter>>

Copyright

© Novática 2004 (for the monograph and the cover page)

© CEPIs 2004 (for the sections MOSAIC and UPENET)

All rights reserved. Abstracting is permitted with credit to the source. For copying, reprint, or republication permission, contact the Editorial Team.

The opinions expressed by the authors are their exclusive responsibility.

ISSN 1684-5285

Next issue (December 2004):
"Cryptography"

(The full schedule of UPGRADE is available at our website)

- 2 Editorial
Four Years of UPGRADE

SPT, Software Process Technology

Guest Editors: Francisco Ruiz-González and Gerardo Canfora

Joint monograph with Novática*

- 3 Presentation
Software Process Technology: Improving Software Project Management and Product Quality – *Francisco Ruiz-González and Gerardo Canfora*
- 6 Software Process: Characteristics, Technology and Environments – *Francisco Ruiz-González and Gerardo Canfora*
- 11 Key Issues and New Challenges in Software Process Technology – *Jean-Claude Derniame and Flavio Oquendo*
- 17 A Taxonomy of Software Engineering Environment Services: The Upcoming ISO/IEC Standard 15940 – *Dan Hyung Lee and Juan Garbajosa-Sopeña*
- 22 Open Source and Free Software: A New Model for The Software Development Process? – *Alfonso Fuggetta*
- 27 Applying The Basic Principles of Model Engineering to The Field of Process Engineering – *Jean Bézivin and Erwan Breton*
- 34 Software Process Modelling Languages Based on UML – *Pere Botella i López, Xavier Franch-Gutiérrez, and Josep M. Ribó-Balust*
- 40 Supporting the Software Process in A Process-centred Software Engineering Environment – *Hans-Ulrich Kobialka*
- 47 Managing Distributed Projects in GENESIS – *Lerina Aversano, Andrea De Lucia, Matteo Gaeta, Pierluigi Ritrovato, and Maria-Luisa Villani*
- 53 Software Process Measurement – *Félix García-Rubio, Francisco Ruiz-González, and Mario Piattini-Velthuis*
- 59 Process Diversity and how Practitioners Can Manage It – *Danilo Caivano and Corrado Aaron Visaggio*

MOSAIC

- 67 Data Architecture
A Disquisition on The Performance Behaviour of Binary Search Tree Data Structures – *Dominique A. Heger*
- 75 News & Events: News from CEPIs and EUCIP; SCI 2005 (Call for Papers)

UPENET (UPGRADE European NETWORK)

- 77 From **Novática** (Spain):
IT and Disabilities
Braille and The Pleasure of Reading: We Blind People Want to Continue Reading with Our Fingers – *Carmen Bonet-Borrás*
- 84 From **Piroforiki** (Cyprus):
Information Technology in Today's Organizations
Is the IT Productivity Paradox Resolved? – *Kyriakos E. Georgiou*

* This monograph will be also published in Spanish (full issue printed; summary, abstracts and some articles online) by **Novática**, journal of the Spanish CEPIs society ATI (*Asociación de Técnicos de Informática*) at <<http://www.ati.es/novatica/>>, and in Italian (online edition only, containing summary abstracts and some articles) by the Italian CEPIs society ALSI (*Associazione nazionale Laureati in Scienze dell'informazione e Informatica*) and the Italian IT portal Tecnoteca at <<http://www.tecnoteca.it/>>.

Software Process: Characteristics, Technology and Environments

Francisco Ruiz-González and Gerardo Canfora

In this introductory article we present the concept of Software Process (SP) and the properties that characterize and distinguish this process from other types of processes (e.g. typical industrial production processes). We go on to justify the interest in having Software Process Technology (SPT) to enable us to automate and to integrate production and management processes in order to carry out software projects. We also present Software Engineering Environments (SEE), collections of integrated tools whose purpose is to provide support to the abovementioned processes. We conclude by summarising the problem of how to integrate the tools making up an environment and how to create a process-oriented environment.

Keywords: Software Engineering Environment, Software Process, Software Process Technology, Process-Oriented, Tools Integration.

1 Characteristics of Software Processes

The definition of *Software Process* (SP) complements the concept of software life-cycle in the sense that it defines the skeleton and philosophy for carrying out an SP, but it is not in itself sufficient to guide and control a development and/or maintenance project. An SP is a “*coherent set of policies, organizational structures, technologies, procedures and artifacts that are needed to conceive, develop, deploy, and maintain a software product*” [3].

The special nature of SPs can be defined as follows:

- a) They are complex.
- b) They are not typical production processes, since they are exception driven, are strongly affected by highly unpredictable circumstances, and each process has peculiarities that distinguishes it from all others.
- c) They are not ‘pure’ engineering processes either, since we do not know the appropriate abstractions, (there is no experimental science behind them), they depend too much on too many people, their design and production are not clearly differentiated, and their budgets, schedules and quality cannot be programmed reliably enough.
- d) They are not (entirely) creative processes because some parts can be described in detail while some procedures are previously enforced.
- e) They are finding-based and depend on communication, coordination and cooperation within predefined frameworks. Their delivery generates new requirements, the cost of changing software is not usually recognised, and their success depends on user involvement and the coordination of many different roles (sales, technical development, customer, etc.).

The necessity for creative human participation and the absence of repetitive actions means that neither the development nor the maintenance of the software is a production process. However, there are some similarities between the two

Francisco Ruiz-González has a PhD in Computer Science from the *Universidad de Castilla-La Mancha* (UCLM), Spain, and an MSc in Chemistry-Physics from the *Universidad Complutense de Madrid*, Spain. He is a full time Associate Professor of the Dept. of Computer Science at UCLM in Ciudad Real, Spain. He was the Dean of the Faculty of Computer Science between 1993 and 2000. Previously, he was the Director of Computer Services at the aforementioned university (1985–1989) and he has also worked in private companies as an analyst-programmer and project manager. He is a member of the Alarcos Research Group, <<http://alarcos.inf-cr.uclm.es/english/>>. His current research interests include software process technology and modelling, software maintenance, and methodologies for software projects planning and management. He has also worked in the fields of GIS (Geographical Information Systems), educational software systems and deductive databases. He has written eight books and fourteen chapters on the abovementioned topics and he has published 90 papers in Spanish and international journals and conferences. He has sat on nine programme committees and seven organizing committees and he belongs to several scientific and professional associations: ACM, IEEE-CS, ATI, AEC, AENOR, ISO JTC1/SC7, EASST, AENUI and ACTA. <Francisco.RuizG@uclm.es>

Gerardo Canfora is a full Professor of Computer Science at the Faculty of Engineering and the Director of the Research Centre on Software Technology (RCOST) of the *Università degli Studi del Sannio*, Benevento, Italy. He served on the programme and organizing committees of a number of international conferences. He was a programme co-chair of the 1997 International Workshop on Program Comprehension, the 2001 International Conference on Software Maintenance, and the 2004 European Conference on Software Maintenance and Reengineering. In 2003 he was the general chair of the European Conference on Software Maintenance and Reengineering. His research interests include software maintenance and evolution, programme comprehension and reverse engineering, software process improvement, knowledge management, and service oriented software engineering. On these topics he has published more than 100 articles in international journals and conferences. He is an associate editor of the *IEEE Transactions on Software Engineering* and serves on the editorial board of the *Journal of Software Maintenance and Evolution: Research and Practice*. <canfora@unisannio.it>

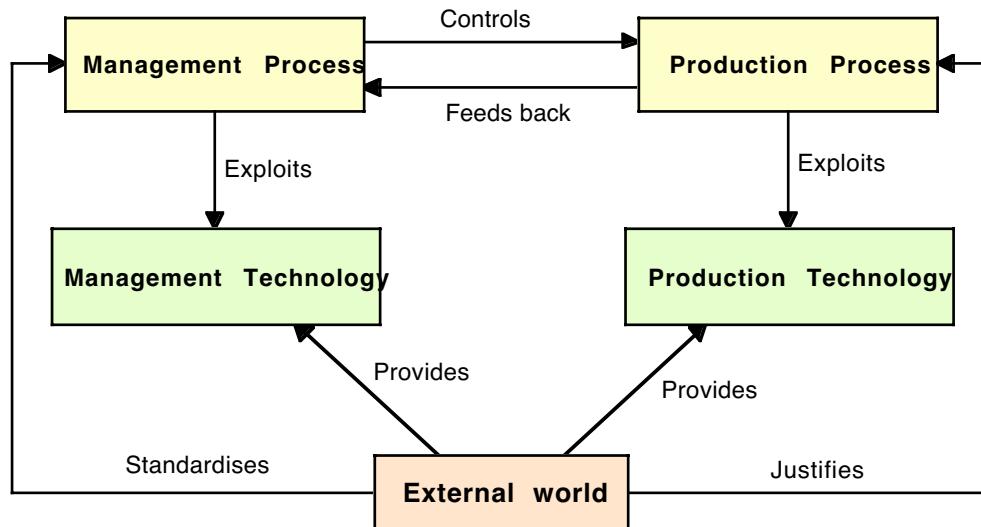


Figure 1: Production Process vs Management Process.

kinds of processes which are useful for understanding software processes in a broader perspective. Like production processes, software processes consist of two inter-related sub-processes; the production process and the management process [5]. The production process relates to the actual production and maintenance of the product, while the management process provides the necessary resources for the production process, and controls it. This control is possible when the production process feeds back information about its situation to the management process. These relationships are represented in Figure 1, as are the relationships between processes and the external environment. The request for a product comes from outside (the external world); in other words, the external environment justifies the production process's existence. Management also has to conform to the current standards of the external environment; in other words, the external environment also has an indirect influence on the production process. In short, production and management processes make use of technologies originating from the external environment.

2 Software Process Technology

The essence of SPT is that it allows the integration of production and management technologies in a new work environment, known as *Process-centred Software Engineering Environment (PSEE)*, which provides support to management and production processes in an integrated manner. Figure 2 shows the impact of this new technology and how PSEE implements, controls and improves the flow of information by which the management process controls the production process. The main objective of SPT is to control the inherent complexity of the SP via an in depth understanding of the process itself, and by means of the automated support provided by a PSEE. An essential factor in achieving this objective is computerized support; in other words, having a process model and the proper means of defining, modifying, analysing and enacting it [1].

Following on from the previous definition, SPT makes use of a wide range of areas and concepts:

1. **Software development and maintenance technologies** which provide the necessary tools and infrastructure to make it possible and economically feasible to create and maintain complex software products that meet present and future needs.
2. **Methods and techniques** for software development and maintenance which provide the essential methodological support required to make efficient use of the abovementioned technologies and successfully perform development and maintenance activities.
3. **Organizational behaviour**, in other words, the science of organizations and people, is useful because software projects are generally carried out by groups that need to be coordinated and directed within an effective organizational structure.
4. **Marketing and economy**, since software development and maintenance projects are not executed in isolation, and, as is the case with any other product, software must be oriented towards the needs of real customers and users.

To sum up, when developing and maintaining software it is necessary to pay attention to the complex relationships that arise between the various organizational, cultural, technological and economical factors.

3 Software Engineering Environments

Although the use of tools in helping developers to produce software has existed in one way or another since the early days of computing, the concept of *Software Engineering Environment (SEE)*, is a relatively recent development. SEE is defined as "a set of tools providing full or partial automated support to software engineering activities". Normally these activities are carried out within the framework of a software project and refer to aspects such as specification, development, reengineering and maintenance of software systems. SEE has

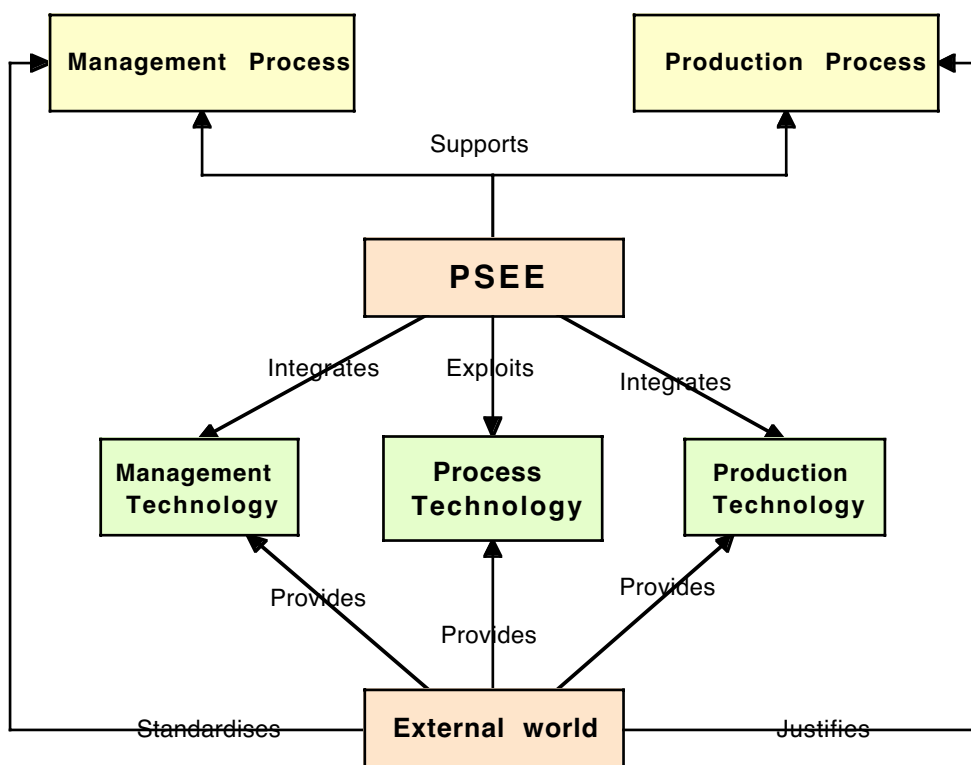


Figure 2: Impact of Software Process Technology.

also been known by other names: IPSE (*Integrated Project Support Environment*), ISEE (*Integrated Software Engineering Environment*), CASE tools coalition, Federated CASE tools, or ISF (*Integrated Software Factory*).

The term SEE can be applied to a wide range of systems: from a set of a few tools running on the same system, to a completely integrated environment able to manage and control all data, processes and activities of a software product life-cycle. Thanks to the automation of activities (partial or total), SEE can produce significant benefits for an organization: reduction in costs (high productivity), improved management and improved quality of the end product. For example, the automation of repetitive activities, such as the performance of test cases, not only improves productivity but also helps to ensure the completeness and consistency of testing activities.

SEE normally deals with information related to:

- Software under development or maintenance (specifications, design information, source code, tests data, project plans...)
- Project resources (costs, computing resources, personnel, responsibilities and duties...)
- Organizational aspects (organizational policies, standards and methods used...)

SEE gives support to human activities by means of a set of services that describe the environment's capabilities. These services provide the correspondence between a chosen set of software life-cycle related processes and their automation by the use of tools. In most cases a tool's functionality is related to one or more services.

Interest was first aroused in SEE in the early 90s when the first reference models were proposed and the first taxonomy of the services to be included was produced [8]. But it was not until the beginning of the 21st century that environments were developed which actually tried to meet the ambitious objectives that the definition of SEEs entails [6].

3.1 Integration

The concept that most differentiates an SEE from a simple set of tools working on a computer under the same operating system, is the degree of integration that it provides. The concept of integration applied to an SEE can mean many related but different things:

- The degree to which different tools can communicate effectively between one another within the framework of an SEE.
- A measure of the relationships between SEE components.
- The simplicity, interoperability, portability, scalability, productivity, etc., produced by the seamless interaction of SEE components.

Sharing the same object management system (repository manager) instead of having a separate file system for each tool is an important feature of integration, but not the only one. SEEs must have a series of interfaces enabling cooperation between tools from different manufacturers. Integration can be said to involve the three following aspects:

- A set of services.** Most of the services described later are applicable to integration. For example, using a common object management system with common schemes enables the tools to share objects; using global presentation charac-

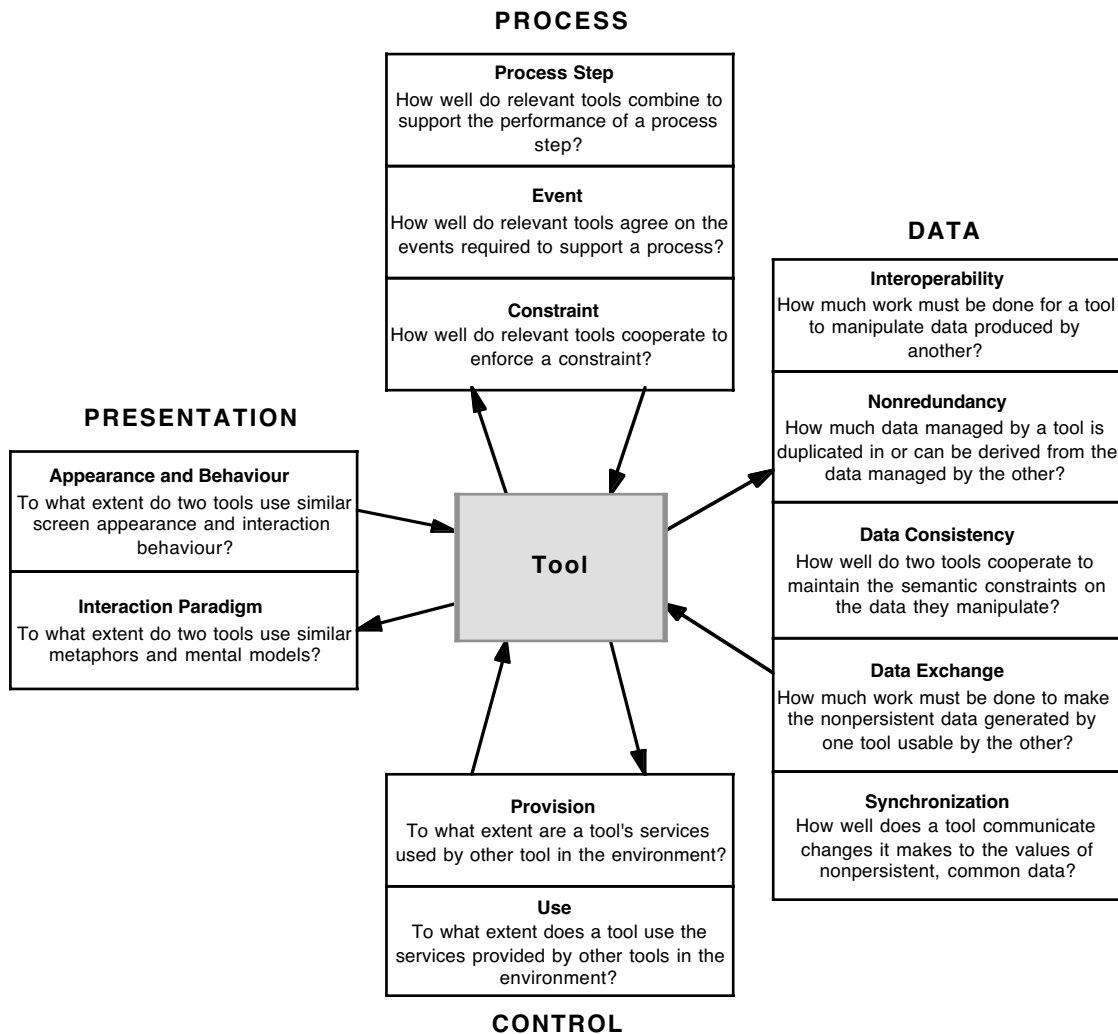


Figure 3: Properties of Tool Integration in SEE.

teristics in the user interface enables all tools to have a similar “visual aspect”; and management process and communication services are necessary for the tools to communicate with each other.

- **A new dimension for each service.** Having common services permits integration but does not make it obligatory (tool developers are not obliged to use them). This new dimension indicates the degree to which a service can help to increase integration.
- **A policy.** It is also necessary to implement policies so that developers of tools, frameworks and platforms can use integration services efficiently. An example of this are the “style guides” intended for tool constructors.

According to [7], the need for integration in an EIS involves several different dimensions (see Figure 3):

- **Data.** Data integration is the ability to share information in the SEE. The degree of data integration can be high (tools use a common database with a common scheme), medium (common data formats) or low (uses transformation mecha-

nisms). Another characteristic that data integration can include is composition.

- **Control.** Control integration is the ability to flexibly combine the functionalities provided in an environment. These combinations may correspond to project preferences and be driven by the underlying software processes.
- **Presentation.** Presentation integration is the ability to interact with environment functionalities with similar screen appearance and similar modes of interaction.
- **Processes.** Process integration is the ability to access environment functionalities using a pre-defined SP enacted with automated support.

3.2 Process Orientation

We have already mentioned the importance which process-oriented SEEs (or PSEE)s have in SPT. In fact, the principal role of an SEE is to provide support to order to enact the SP effectively. This point of view is gaining ground as software development and maintenance processes have become intellectual activities of an ever more complex and laborious nature,

with a great potential for improvement to quality and productivity, based on discipline, management, and the help of PSEEs and other computing technologies. Many organizations have problems defining and performing the steps which transform user requirements into a software product in such a way that they are replicable, measurable in terms of quality objectives, and adaptable or improvable, so the use of an SEE to implement a defined process during the performance of a software project can provide considerable short term benefits.

In a PSEE, process management services contribute to the effective support of SPs, providing end-user oriented facilities in order to define and use processes that can replace the undisciplined, difficult to control, and tedious invocation of individual tools. Garg and Jazayeri [4] believe that process support in PSEE is based on the following functionalities:

- *Process definition.* A process engineer use the PSEE to define a process to be followed by one or more projects.
- *Process analysis.* A process model within a PSEE can be analysed to verify its consistency, completeness, and correctness.
- *Process presentation.* A PSEE includes support for the graphical display of SPs (activity flows) and products (structured diagrams).
- *Process simulation.* The PSEE supports the use of simulations to be able to evaluate the suitability of a process before committing full resources to it.
- *Process automation.* Once a process has been defined, activities which do not require human intervention can be identified and automated by the PSEE.
- *Process monitoring.* The PSEE monitors the execution of a process and records the history of the activities carried out. This process history can be used later for future process developments and improvements.
- *Process change support.* The PSEE allows an organization to change its process definitions without interrupting its work.
- *Openness.* The PSEE provides tools to exchange data and metadata with other non-integrated tools or with other PSEEs.
- *Multi-User support.* Typically, software engineering projects are worked on by teams of people with different roles. The PSEE must therefore provide services to all the people working together on a process.
- *Process guidance.* Software engineers use the PSEE to carry out various process steps. The PSEE must provide help in choosing among possible next steps based on the modelled process and the current state.
- *Task-specific user interface.* Based on the modelled process, the PSEE can scope the user interface to the needs of each task, thereby preventing an excess of information from being presented to the user.

It is becoming ever more common for software product development and maintenance to be carried out with the collaboration of various companies or organisations. As a result, in recent years there has been increasing interest in the study of

the problems that can arise when several separate and distinct PSEEs are required to collaborate with one another, and, more specifically, in the need for interoperability between the processes supported by those PSEEs. Among the various proposals made to address this problem, perhaps the most interesting is that put forward by a number of authors who, using the metaphor of international alliances, support the idea of forming federations of PSEEs, whereby each organization would manage its own processes (just as each country has its own laws), and inter-organizational processes would act in a similar way to international treaties between countries. In the relevant bibliography, two types of conceptual architectures have been proposed for PSEE federations: control based, favouring centralization given the existence of common process models; and state based in which there is a workspace in which the common state is stored [2].

4 Conclusions

In this article we have presented the key aspects of Software Process Technology: the object of interest (software process and their characteristics); the interest in and justification for giving automatic support to these processes; and the requirements, functionality and characteristics of integration and process-orientation that the collection of tools need to have to achieve these goals.

References

- [1] J. C. Derniame, B. A. Kaba, and D. Wastell, D. (eds.). *Software Process: Principles, Methodology and Technology*. LNCS 1500, Springer-Verlag, 1999.
- [2] J. Estublier, P. Y. Cunin, and N. Belkhatir. *Architectures for Process Support System Interoperability*. Proceedings of the Fifth International Conference on the Software Process (ICSP'98), 15–17 Junio, Chicago (Estados Unidos), pp. 137–147, 1998.
- [3] A. Fuggetta. *Software Process: A Roadmap*. 22nd International Conference on Software Engineering (ICSE'2000), Future of Software Engineering Track, June 4–11, Limerick (Irlanda), ACM, 2000.
- [4] P. K. Garg and M. Jazayeri. *Process-centred Software Engineering Environments: A Grand Tour*. In Fuggetta, A. y Wolf, A. (eds.); *Software Process*. John Wiley & Sons, 1996.
- [5] R. McLeod Jr. *Management Information Systems*. McMillan Publishing, New York, 1990.
- [6] H. Ossher, W. Harrison, and P. Tarr. *Software Engineering Tools and Environments: a Roadmap*. International Conference on Software Engineering (ICSE) – Future of SE Track. Limerick (Irlanda), pp. 261–277, 2000.
- [7] I. Thomas and B.A. Nejmeh. *Definitions of Tool Integration for Environments*. *IEEE Software*, 9(2), pp. 29–35, 1992.
- [8] M. V. Zelkowitz. *Software Engineering Environment Capabilities*. *Journal of Systems and Software*. Elsevier Science, 35(1), pp. 3–14, 1996.