17

Software Factories*

Ivan Aaen
Peter Bøtcher
Lars Mathiassen

Abstract. Software factory efforts and related ideas have been around for thirty years. The term signals a commitment to long-term, integrated efforts—above the level of individual projects—to enhance software operations. This is not only a powerful, but a necessary idea taking the challenges involved in professionalizing software operations into account. The term factory has, however, the controversial connotation that software development and maintenance is comparable to mass-production of industrial products, and arguably this is not the case.

The paper discusses contrasting views by providing a survey of software factory concepts, by highlighting variations and differences between them, and by discussing the relative strengths and weaknesses of their approaches to professionalize software operations. Based on a comparison of four well-known factory, or factory-like, concepts the paper identifies useful contributions and possible illusions related to the idea of a software factory.

Introduction

Software organizations experience considerable pressures to professionalize their operation. Clients and customers ask for transparent development processes and faster feed-back, they require higher productivity, and, last but not least, they want better quality in the delivered services and products. For more than thirty years, soft-

* Published as: The Software Factory: Contributions and Illusions. I. Aaen, P. Bøttcher & L. Mathiassen. In: Proceedings of the Twentieth Information Systems Research Seminar in Scandinavia, Oslo, 1997.

ware engineer's have been occupied with this challenge trying to come up with innovations to enhance software operations. Throughout this period, the idea of the software factory has kept emerging as a sort of ultimate response to the needs of our profession.

A historical interpretation of the software factory concept and experience is provided by M. A. Cusumano (1989). R. W. Bemer was probably the earliest proponent suggesting in 1968 that General Electric develop a software factory to enhance programmer productivity through standardized tools, a computer-based interface, and a database with historical data for financial and management controls (Bemer 1969). Approximately at the same time, M. D. McIlroy of AT&T proposed a different factory-like concept emphasizing systematic reusability of code when developing new programs (McIlroy 1969).

The first company in the world to label a software organization as a factory was Hitachi in 1969, while the second software factory was established by one US leader in the custom software field, System Development Corporation, during 1975–1976. According to Cusumano, a software factory boom then followed, in particular in Japan where NEC, Toshiba, and Fujitsu launched their own factory efforts during 1976–1977. More recently, Mitsubishi and Nippon Telephone and Telegraph have initiated factory-like efforts, and in 1985 MITI—Japan's Ministry of International Trade and Industry—started the national SIGMA project as a cooperative effort to develop an infrastructure from which to produce high-quality software in great quantity (Cusumano 1989).

The term factory signals a commitment to long-term, integrated efforts—above the level of individual projects—to enhance software operations. This is not only a powerful, but also a necessary idea taking the challenges involved in professionalizing software operations into account. But for many the term factory has at the same time the controversial connotation that software development and maintenance is comparable to mass-production of industrial products, and arguably this is not the case. This can easily lead to illusions with respect to the kinds of interventions that can, in fact, improve software operations. It is not surprising, therefore, that some software professionals like the concept while others do not.

The term factory can be used to denote either one or more buildings with facilities for manufacturing or the seat of some kind

of production. To many people the concept of a factory also implies a particular way of organizing work with considerable job specialization, formalization of behavior and standardization of work processes. In this paper we will not adopt this historically based connotation. Rather we use the term without assumptions regarding particular ways to standardize, formalize, specialize, or achieve functional grouping. The factory is an organization inhabited by people engaged in a common effort, work is organized one way or the other, standardization is used for coordination and formalization, and systematization is important, but there will be several options for the design of a particular software factory. This paper investigates how existing approaches to the software factory has chosen among such options by fitting each approach into one of Henry Mintzberg's five basic organizational structures (Mintzberg 1983): the simple structure (organic, centralized, direct supervision); the ad-hocracy (organic, decentralized, mutual adjustment); the machine bureaucracy (bureaucratic, centralized, standardized processes); the professional bureaucracy (bureaucratic, decentralized, standardized skills); and the divisionalized form (decomposed based on standardized output).

In the paper, we discuss and evaluate contrasting views on software factories. We do that by presenting a selection of different software factory concepts, by highlighting variations and differences in the underlying approaches, and by discussing the relative strengths and weaknesses of different approaches to professionalize software operations. The goal is to clarify useful contributions and possible illusions related to the idea of a software factory.

The approach taken is basically that of a literature survey. We have selected four well-known factory-like approaches to professionalize software operations. The four approaches cover Japanese, European, and North American initiatives, some are mainly tool oriented while others are more process oriented, and together they cover the range from quite early to more recent initiatives. The four approaches are: a Japanese approach to the industrialized software organization (Matsumoto 1981, 1987), a European approach to the generic software factory (Fernström 1991, 1992), a North-American approach to the experience-based component factory (Basili 1989, 1993; Basili *et al.* 1992), and, finally, a North-American approach to the mature software organization (Nilsson 1990; Paulk *et al.* 1993a).

The paper is divided into two major sections. In the first section we present and interpret the four approaches. The second section is devoted to a comparison and discussion of the four approaches. We outline the important similarities and differences, and we discuss their relative strengths and weaknesses. The paper is concluded with a discussion of the contributions and illusions related to software factories.

I. Software factory approaches

In order to compare software factory approaches we need to know the contents of each approach as well as the context in which it was conceived. The context gives an understanding of the scope and main focus of an approach. Under this heading we outline the setting in which the approach was developed. The contents of each approach is described in a fairly top down manner starting at the top with the objective of the approach and the strategy advocated for achieving the objective. In the middle we describe the organizational design of the software factory in question, and at the bottom we describe the implementation of this organization. Thus the description of each approach falls under five headings:.

Context. In which setting was the approach developed?

Objective. Which goals are pursued and which specific problems are sought to be solved?

Strategy. How should these goals be achieved?

Organization. What is the proposed design of the software factory?

Implementation of improvement. How should the improvement efforts be implemented?

Under the last of these headings we characterize similarities and differences between the studied approaches by referring to the common elements involved in implementing software factories identified by Cusumano in his study of Japanese initiatives (Cusumano 1991): commitment to process improvement; product-process focus and segmentation; process-quality analysis and control; tailored and centralized process R&D; skills standardization and leverage; dynamic standardization; systematic reusability, computer-aided tools and, integration; incremental product/variety improvement.

I.1. The industrialized software organization (Japan)

As a representative for the Japanese software factory approach we have chosen Toshiba's software factory concept, described in (Matsumoto 1981, 1987). This concept is used to denote Toshiba's software division and its procedures as they were in 1981 and 1987 respectively.

Context. The software produced in the software factory is primarily for control systems, nuclear reactors, turbines etc. The establishment of the software factory in 1981 is motivated by a wish for software of higher quality in terms of minimizing the number of defects in the software. The focus on software quality is matched by a focus on productivity to ensure that the quality efforts do not weaken competitiveness via increased costs.

Objective. To meet these constraints the objective is to increase software quality and to improve productivity. Later in 1987 the objectives of quality and productivity are maintained with the additional objective of creating an environment in which design, programming, test, installation, and maintenance can be performed in a unified manner.

Strategy. The strategy in 1981 includes three elements. The first is to design buildings that support the software development process, the second is to build a Software Work Bench (SWB) which is an integrated software support for the activities in the software development process, and the third is to establish an organization that controls and monitors the software development process. Later in 1987 these strategies are maintained, but more initiatives are added: properly designed work spaces; software tools, user interfaces, and tool maintenance facilities; standardized baseline management system for design review, inspection, and configuration management; standardized technical methodologies and disciplines; education program; project progress management system; cost management system; productivity management system; quality assurance system with standardized quality metrics; quality circle activities; documentation support; existing software library with maintenance support; technical data library; career development system.

Organization. The organization of the Toshiba software factory is determined by some of the elements in the software factory: Software work bench, project management, reusability, measuring productivity, measuring quality, and quality circles.

The term software work bench is used to denote an integrated system for supporting all workers in the factory. The system consists of a number of subsystems that collectively offer support for: programming, debugging, project files, program generation, and program reuse; program test; requirements specification, software design description, and documentation; maintaining software in operation at customer sites; project management; quality assurance; software configuration control and reuse.

The software factory has a standardized waterfall model for system development. The project management strategy used is called look-forward-management. The idea is to calculate costs from equations containing data from the organizational history. All projects are decomposed into unit workloads. A unit workload is defined to be an activity to complete a software configuration by one person. Progress in the projects is managed based on daily or weekly status reports on these unit workloads. The use of frequent progress reports enables tracking of actual versus expected progress before items are complete. Therefore corrective actions can be taken during the process. This in combination with the use of organizational data to estimate the expected time and resources form the background for the term look-forward-management.

Quality and productivity is measured in the software factory. There are quality factors defined for each baseline of the life-cycle model. Software quality is understood in terms of reliability, and the quality measures express the number of expected faults left in the code after test, and the expected time between failures. These measures are estimated from the number of faults found in test. Productivity is also measured for every step in the life-cycle model. The software factory uses two types of measures: cost-based measures and capability-based measures. The cost based measures are cost per person-month, profit per person-month, and cost per equivalent assembler source lines. The capability measures are used in progress management, work assignment, and education planning. On the project and factory level equivalent assembler source lines per person-month, specification pages per person-month, and test items per person-month are created. On the person level a personal spectrum is created which includes productivity and fault rate. For instance, the number of pages produced by an analyst per hour is the productivity measure for an analyst. The number of faults per page is the fault rate. In the programming phase the productivity

measure is the number of equivalent assembler source lines produced per hour.

Reuse of software is considered to be the single most critical issue in improving quality and productivity. The primary source of reuse is reliable and documented software. To promote reuse an organization for reuse has been designed. The key elements are the software reusing parts steering committee, the software reusing parts manufacturing department, and the software reusing parts center. The steering committee gathers, selects, and authorizes needs for creating, updating, and discarding reusable modules, and the manufacturing department processes these needs. The parts center is the place where all the reusable parts are kept and available to the projects. To help the search for reusable parts, keywords are assigned to describe the functionality of the reusable parts and search mechanisms are offered.

Quality circles are volunteers' groups. They meet to discuss and exchange ideas on quality improvement, productivity, improvement of methodology etc. Each year both a factory-wide and a company-wide quality circle conference are held. The best groups are rewarded by being invited to these conferences where various awards are presented to honor excellent activities.

The dominant traits of the organizational design are: a determined effort to make the operating work routine, simple, and repetitive and to standardize work processes. Responsibilities and routines are standardized and there is a clearly defined hierarchy of authority combined with an elaborate administrative structure. All in all this organizational design corresponds to what Mintzberg terms a *Machine Bureaucracy* (Mintzberg 1983).

Implementation of improvement. Although this approach puts a heavy emphasis on infrastructures (buildings and tools), the implementation efforts in this approach are comprehensive: all of Cusumano's nine implementation elements can be identified here.

I.2. The generic software factory (Europe)

The second idea to be considered is a European approach to the generic software factory (Fernström 1991; Fernström *et al.* 1992; Nilsson 1990). The approach is funded under the Eureka program and is called the Eureka Software Factory.

Context. The participants in the project are large European companies, computer manufacturers, software houses, research in-

stitutes, and universities. The project was designed with a 10 year horizon (1987–1996) with 2400 man-years of work financed by industry (50%) and national government (50%). Most of the development work was performed in sub-projects.

In this project the aim is to provide the technology, standards, organizational support, and other necessary infrastructures in order that software factories be constructed and tailored from components marketed by independent suppliers. The software factory concept denotes a combination of software tools and software processes to be installed in an organization. A software factory thus consists of both computerized and non-computerized parts.

The main focus in the project is on the factory support environment i.e. the computerized support part of a software factory. The project defines a communication-centered CASE-architecture to be combined with specific support for describing and enacting software-engineering activities. Together this is intended to help builders of software factories in their effort to integrate CASE products into software process models. In this respect the Eureka Software Factory can be viewed as a Generic Software Factory to produce software factories.

Objective. The objective is to produce an architecture and a framework for ISDEs—Integrated Software Development Environments—basic building blocks, general components, and environments for different application areas (like business applications, real time applications, telecommunication systems, and embedded systems). Based on this it should be possible to "compose" support environments tailored to the tasks to be performed by users in specific projects. The main principle is to adapt the factory support environment to the organization rather than the other way around.

Strategy. The Generic Software Factory develops components and production environments that are part of software factories together with standards and guidelines for software components. The component standards ensure that the different components of an ISDE can communicate through a common software bus. This bus handles communication, data conversion, configuration, and other services. The bus makes it possible to select and combine components that conform to specific needs in an organization or a project. Composing a user environment thus consists of selection, combination, and set-up of a number of components.

A software factory is produced as shown in figure 1. The Generic level defines a reference architecture for software factories with standards for Eureka Software Factory compliant components. The Component Base level denotes the base of available components for building factory support environments. The number of components compliant with the Eureka Software Factory standards is continuously extended. The Factory Model level describes specific software factories in terms of models of the processes to be supported by the factory and the characteristics of the components integrated into the support environments. Finally, the Software Factory level denotes the customized software factory instance put into place within the software development organization.

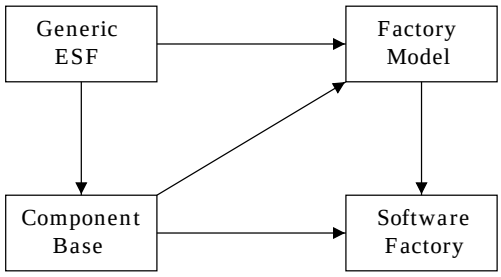

Figure 1. Producing an instance of a software factory.

Organization. Process modeling plays an important role in the definition of the requirements for the factory support environment. The users are placed in work contexts that form parts of explicit models of the software production process. The work contexts are intended to support the work of individuals, to increase predictability of the process, and to bridge between the computerized and non-computerized parts of tasks.

The software processes describe the working procedures supported in the factory. Part of this description deals with the customization of the factory support environment to fit the organization and projects. The process models cover the relationships between roles, tasks, activities and tools, and the task descriptions serve as templates for specific tasks to be executed during a project.

The dominant traits of the organizational design are: a determined effort to formalize the operating work, to link automated and non-automated work into coherent processes and thereby to stan-

dardize the work processes. The organizational design consists of an elaborate process design focusing on tasks and tools, and the aim is to embed the software process in tools. Like Japan's Industrialized Factory this organizational design corresponds to Mintzberg's *Machine Bureaucracy* (Mintzberg 1983). However these two approaches differ significantly in strategy: the Industrialized Factory seeks to enforce standardization mainly via work procedures, whereas the Generic Software Factory uses automated tools.

Implementation of improvement. Management of the improvement effort is not explicitly described in the Eureka Software Factory. Rather, adopting the factory support environment from the Generic Software Factory to the organization is viewed as an atomic process. The main emphasis is on establishing technological infrastructures, and of Cusumano's nine implementation elements only a few play a prominent role in this approach: product-process focus and segmentation; tailored and centralized process R&D; systematic reusability; computer-aided tools and integration.

I.3. The experience-based component factory (US)

The third idea to be considered is a North-American approach to the experience-based component factory (Basili 1989, 1993; Basili *et al.* 1992).

Context. The Experience-based Component Factory is developed at the Software Engineering Laboratory which has existed since 1976 as a consortium between NASA/Goddard Space Flight Center, University of Maryland, and Computer Science Corporation (Basili 1993). Its goals are to "(1) understand the software process in a production environment, (2) determine the impact of available technologies, and (3) infuse identified/refined methods back into the development process". The approach has been to experiment with new technologies in a production environment, to extract and apply experiences and data from the experiments, and to measure the impact with respect to cost, reliability, quality etc.

Objective. It is assumed that significant changes are needed in the way software is produced. Software organizations need to increase both quality and productivity, and a solution is summarized in three goals: "improve the effectiveness of the process, reduce the amount of rework, and reuse life-cycle products" (Basili *et al.* 1992).

Strategy. The strategy, continuos improvements based on reuse of prior experiences and flexible automation, consists of three

key elements: an improvement paradigm, a dedicated experience organization, and a contingency approach.

An improvement paradigm (plan, execute, analyze, synthesize) is applied to integrate efforts on the organizational and project level (Basili *et al.* 1992): Projects are planned explicating goals and utilizing packaged experiences from previous projects; projects are then executed and controlled by measurement; subsequently the results are analyzed and compared with the planned goals; finally, experiences are synthesized and packaged in the form of new or updated models to be used in future projects.

A dedicated experience organization is formed because reuse of experiences requires separate resources to create and maintain reusable objects (Basili 1993). The activities are thus divided into two separate logical and physical organizations: the project organization whose focus is delivery of software supported by packaged reusable experiences, and an experience factory whose focus is to support projects by providing reusable experiences.

A contingency approach is taken because "every environment has its characteristics and pursues its goals by means different from any other one" (Basili *et al.* 1992), and because "the organization must be able to change its configuration" (Basili *et al.* 1992). The architecture of the Experience-based Component Factory is therefore described on different levels of abstraction: the reference level (representing agents with specified functions), the conceptual level (representing flows of data and control between agents), and the implementation level (defining the actual technical and organizational implementation). The reference architecture is used to characterize established initiatives and to describe possible, alternative approaches, e.g. a clustered architecture in which every development takes place in the project organization versus a detached architecture in which no development, but only design and integration, takes place in the project organization.

Organization. The organization of the Experience-based Component Factory is illustrated in figure 2. The project organization is primarily responsible for the planning and development activities; the focus is on problem solving. The experience factory is primarily responsible for the learning and technology transfer activities; focus is on understanding solutions and packaging experiences for reuse. This organization "recognizes the fact that improving software process and product requires the continual accu-

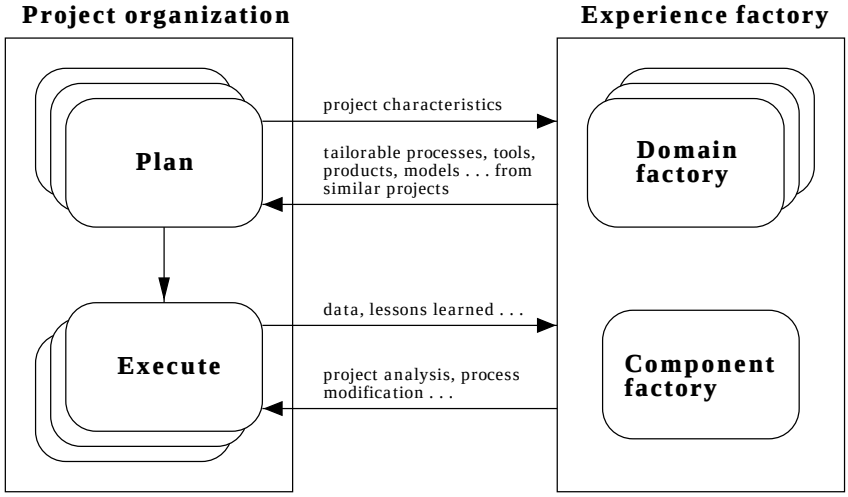

Figure 2. The organization of the Experience-based Component Factory (Basili 1993).

mulation of evaluated experiences (learning), in a form that can be effectively understood and modified (experience models), stored in a repository of integrated experience models (experience base), that can be accessed/modified to meet the needs of the current project (reuse)" (Basili 1993).

The experience factory is divided into several sub-organizations, each dedicated to a particular kind of experience. A first division results in a number of domain factories whose purpose it is to identify, collect, organize, and provide experiences related to a specific application domain, e.g. production control systems or financial systems. A further subdivision of the experience factory is the component factory whose purpose it is to develop and package reusable software components. A software component is any product in the software life-cycle, such as: code components, designs, collections of code components and designs, and documents in general. A reusable software component is a collection made of software components packaged with everything that is needed to reuse and maintain it. This includes the code, its functional specification, its context, a full set of test cases, a classification according to a certain taxonomy, and a reuser's manual (Basili *et al.* 1992).

The dominant traits of the organizational design are: project-organized units are responsible for planning and development.

These units receive support from specialist units. Focus is on learning, training, and technology transfer. In other words focus is on skills and collective experience rather than procedures as the preferred way to coordinate work. All in all this organizational design corresponds to what Mintzberg terms a *Professional Bureaucracy* (Mintzberg 1983).

Implementation of improvement. Practical use of the Experience-based Component Factory concept requires an incremental management approach. The starting point is the present operation in the software organization. A unique experience factory is designed and implemented as an instantiation of the reference architecture reflecting the specific characteristics of the organization in question. The factory is used to collect data about strengths and weaknesses, to set baselines for improvements, to establish experiments with new techniques and methods, and to collect experiences to be reused in new projects. In this way, the organization starts to understand the relationship between certain process characteristics and product qualities. As the software processes are changed new baselines can be established identifying new possible improvements (Basili 1993). This approach puts a heavy emphasis on continuous improvement and points to a wide range of implementation efforts in order to facilitate this. Although all of Cusumano's implementation elements can be identified, the main focus here is on software process.

I.4. The mature software organization (US)

Finally, we will review a software factory-like concept, the mature software organization, defined by the Capability Maturity Model (CMM).

Context. The CMM was initiated by the US Department of Defense's need to evaluate software contractors. There had been too many failing systems and software project disasters. To evaluate software contractors a questionnaire was developed, and based on experiences from using this questionnaire and based on industry feedback and involvement the Software Engineering Institute developed the CMM, see figure 3 (Paulk *et al.* 1993a).

Objective. Based on the initial experiences from the contractor the objective is to create a framework for software process improvement to achieve a predictable, reliable, and self-improving software development process that produces software of high quality

(Paulk *et al.* 1993a). Predictable meaning cost estimates and schedule commitments can be met, reliable meaning the capability of the process is known, and self improving meaning that there is a constant focus on improving the process, and that the knowledge and abilities to improve are established. All this should be achieved through attention to process (improvement) and not methods or tools.

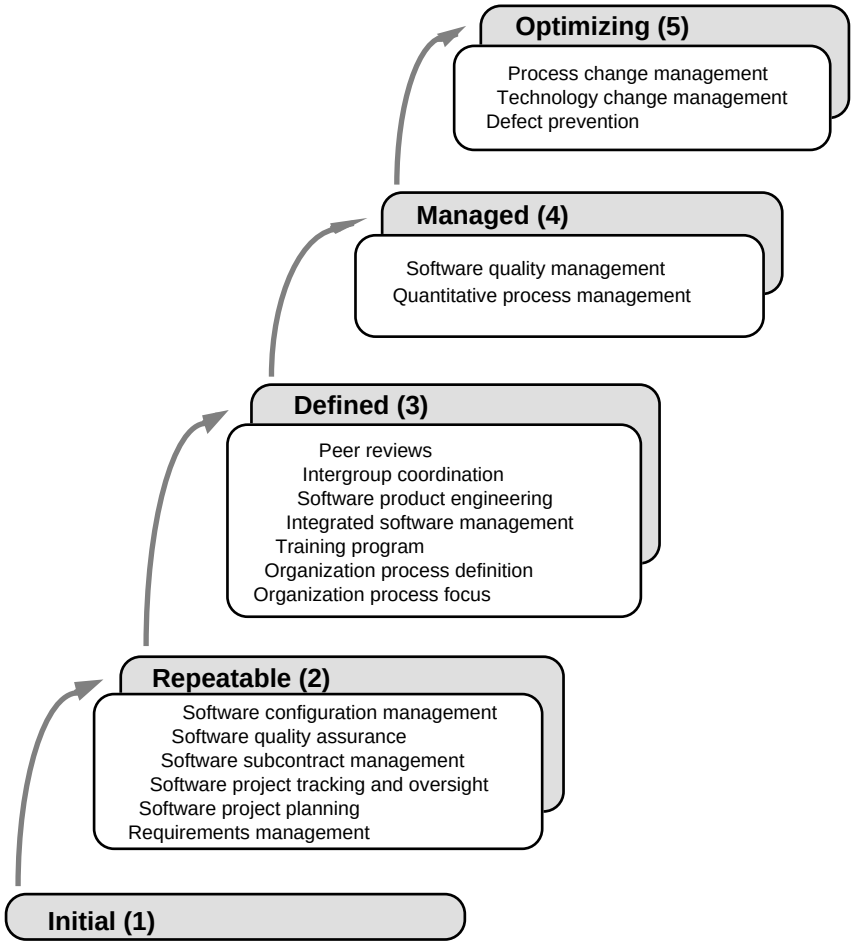

Figure 3. The Key Practice Areas in the Capability Maturity Model (Paulk et al. 1993a).

Strategy. The strategy in CMM is to do stepwise improvements in the software organization. The model defines which processes are key in software development and defines the order of the improvement of the processes. CMM has five maturity levels that describe the development from a chaotic and problematic process to a managed and optimizing process. Each level characterizes the extent to which an organization's processes are explicitly defined, managed, measured, controlled, and performed effectively.

Each maturity level is composed of key process areas. For example, one of the key process areas for level 2 is software project planning. There are 18 key process areas in all, each of these is defined and discussed in great detail, and this essentially defines the standards needed to reach a specific level of maturity (Paulk *et al.* 1993b). A key process area is a cluster of related activities that when performed together collectively achieve a set of goals important for establishing process capability at that maturity level. It is defined in terms of goals for performing the key process area, commitments that must be in place in order to be able to perform the key process area, abilities that must be established for the area, activities that are included in an area, measurement and analysis that must be taken to determine the status and effectiveness of the activities, and finally verification of implementations ensures that the activities are performed in compliance with the process that has been established for the area. The specifications of the key process areas are generic which allows flexibility in actual implementations in the organizations. This means that when areas are assessed, implemented, or improved there is a need for professional judgment and an understanding of what is appropriate for the specific organization. A key practice area can be implemented in an organization and be compliant with the CMM, but it may still be possible to improve the area and change the implementation.

Organization. The ideal organization as described by CMM has a set of characteristics. In this organization software is developed in a disciplined manner in which planning and tracking of software projects is stable and earlier successes are repeated. At the project level, the projects' processes are under the effective control of a project management system, following realistic plans based on the performance of previous projects. The software process is described and consistent and both software engineering and management

activities are stable and repeatable. Schedule, and functionality are under control, and software quality is tracked.

The software process and the software product quality are predictable because they are measured and kept within measurable limits. Trends in the process and product quality can be predicted within the quantitative bounds of these limits and when the limits are exceeded, action is taken.

At the organizational level, the process capability is based on a common, organization-wide understanding of the activities, roles, and responsibilities in a defined software process. The entire organization is focused on continuous process improvement and has the means to identify weaknesses and strengthen the process proactively, with the goal of preventing the occurrence of defects. Data on the effectiveness of the software process is used to perform cost benefit analyses 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.

No matter on which CMM maturity level an organization is, continuos improvement is a key concept in the CMM software factory. When the organization is committed to use the CMM and all levels in the organization have been informed about the strategy chosen, an appraisal of the organization must be conducted in order to find out which processes are the most important to improve. This appraisal involves questionnaires and interviews conducted in collaboration between consultants and an internal appraisal team. The result of the appraisal is a number of recommendation on what to improve and based on these recommendations management will decide which areas to address.

To find ways to improve and to implement the improvements, most CMM improvement efforts are organized in software engineering process groups (Herbsleb *et al.* 1994). These groups manage the improvement effort, they keep track of the improvement activities, and they support and facilitate ad-hoc process area working groups. The ad-hoc groups consist of software engineers and others involved in the software development projects and each group work on implementing improvements in a single process area. A software engineering process group is often mostly staffed by software project members who work in the group for a period of, say, two years and then return to work in software development projects (Fowler *et al.*

1990). When the organization believes that the committed process problem areas have improved (typically after 18 to 36 month) a new appraisal is conducted in order to find new areas of improvement (Hayes *et al.* 1995; Paulk *et al.* 1993b).

The dominant traits of the organizational design have much in common with the Experience-based Component Factory: project-organized units are responsible for planning and development, and these units receive support from specialist units. The support units have primary responsibility for training, change management, quality assurance, and for the definition of a standard software process in the organization. Furthermore the support units are responsible for learning from project experiences. Projects have primary responsibility for development, management, coordination, reviews, defect prevention. This responsibility extends to adapting the standard software process to the project. Like the Experience-based Component Factory focus is here on skills and collective experience rather than procedures as the preferred way to coordinate work. Like the Experience-based Component Factory this organizational design corresponds to Mintzberg's *Professional Bureaucracy* (Mintzberg 1983). However these two approaches differ with respect to the definition of specialized units: The Experience-based Component Factory assigns detailed responsibilities to specialized units, whereas the Mature Software Organization identifies a great number of objectives and commitments to organizational functions rather than named specialist units.

Implementation of improvement. Key elements in managing the improvement process are continuos improvement, top management commitment, identifying resistance in the organization, and letting the people closest to the processes in question improve those processes. In this approach we find a heavy emphasis on stepwise improvement. The implementation efforts cover a wide spectrum and all of Cusumano's implementation elements can be identified here. Like the Experience-based Component Factory the main focus of this approach is on software process.

II. Comparison and discussion

In the next paragraphs the four software factory approaches are compared and discussed. Table 1 (a & b) summarizes the comparison of the approaches.

	Industrial-ized Factory (Japan)	**Generic Factory (Europe)**	**Experience-based Com-ponent Fac-tory (US)**	**Mature Soft-ware Organi-zation (US)**
I. Factory	Yes	Yes	Yes	No
Generic concept	No	Yes	Yes	Yes
II. Objective	Increased quality and productivity of development and maintenance.	Tailor-made integrated software development environments.	Better process effectiveness, less rework, and more reuse.	An effective, predictable, reliable and self-improving process.
III. Strategy	*Infrastructural* Combining physical, organizational and tool-based infrastructures.	*Tool-driven* Standardization of components and customization of processes and components.	*Continuos* Improvement based on experience and flexible automation.	*Stepwise* Improvement by moving up process maturity levels.
Metrics for improvement	Yes	No	No	Yes
Customized project model	No	Yes	Yes	Yes
Reuse of life-cycle products	Yes	No	Yes	No
Technology focus	Yes	Yes	No	No

Table 1(a). Similarities and differences between the four approaches.

II.1. Similarities and differences

Whereas the Mature Software Organization does not mention software factory as a term, and the Experience-based Component Factory uses the term component factory, both the Generic Software Factory approach and the Industrialized Software Factory approach explicitly use the term software factory. The term has different meanings in what it refers to, since the Mature Software Organization, the Generic Software Factory, and the Experience-based Component Factory all describe a generic concept, whereas the Industrialized Software Factory uses the term to refer to a specific instance of a software factory: the Toshiba software factory.

	Industrial-ized Factory (Japan)	**Generic Factory (Europe)**	**Experience-based Component Factory (US)**	**Mature Software Organization (US)**
IV. Organization	• Standardized life-cycle model • Dedicated organizational units.	• Process model with roles, tasks, activities, and tools. • Task descriptions follow from process model.	• Separate project organization for problem solving • Separate experience factory for learning and technology transfer.	• No specific organizational model. • Organizational responsibilities specified
Structure	Machine bureaucracy based on standard work procedures	Machine bureaucracy based on automated tools	Professional bureaucracy with named specialized units	Professional bureaucracy with identified organizational functions
Reuse of process components	Projects are standardized	Yes	Yes	Yes
Competency focus	Yes	No	Yes	Yes
V. Implementation of improvement	(Quality circles)	Not present	Incremental management.	Groups and commitment processes
Dedicated improvement organization	(Yes)	No	Yes	Yes

Table 1(b). Similarities and differences between the four approaches.

The software factories have different objectives. The Mature Software Organization and the Generic Software Factory aim at creating a framework for improvement (based on different strategies), whereas the Experience-based Component Factory has the specific objective to improve process effectiveness, reduce rework and reuse life-cycle products, and the Industrialized Software Factory has the objective to create tailor-made integrated software development environments. All four software factories are, however, unifying processes, tools, or architectures, even though only the In-

dustrialized Software Factory mentions this explicitly as an objective.

Another difference in the objectives is that the Industrialized Software Factory and the Generic Software Factory explicitly aims to improve tools and software development environments, whereas the Experience-based Component Factory and the Mature Software Organization both focus on development and improvement processes.

Concerning the proposed strategy we make three key observations. Firstly, the Mature Software Organization and the Experience-based Component Factory both rely on continuos improvement. Secondly, the Industrialized Software Factory emphasizes physical infrastructure, organizational measures, and an integrated tool-box for the developers. Thirdly, the Generic Software Factory proposes to develop components and production environments as part of software factories together with standards and guidelines for software components.

The Industrialized Software Factory and the Mature Software Organization have metrics to track or support process improvement, whereas the Generic Software Factory and the Experience-based Component Factory have no metrics for software process improvement. The Generic Software Factory, the Experience-based Component Factory, and the Mature Software Organization all use customized project models, whereas the Industrialized Software Factory does not, since all projects are standardized. Reuse of life-cycle products are built into the strategy of the Experience-based Component Factory and the Industrialized Software Factory, but is not part of the strategy of the Generic Software Factory or the Mature Software Organization. Finally, the Industrialized Software Factory and the Generic Software Factory have a strong focus on technology. The Experience-based Component Factory and the Mature Software Organization do not focus on technology, but have a strong focus on processes.

The Industrialized Software Factory, the Experience-based Component Factory, and the Mature Software Organization all describe organizational units of the software factory. The Industrialized Software Factory has dedicated organizational units for reuse and quality improvements. The Experience-based Component Factory also have an organizational unit handling reuse, in addition to other dedicated organizational learning units. The Mature Software

organization has the software engineering process group as a key organizational unit in the software factory together with ad-hoc groups to improve specific key processes. The Generic Software Factory, however, does not recommend any organizational units within the software factory, since all handling of changes and improvements are placed in the generic software factory servicing all other software factories.

The Industrialized Software Factory, the Experience-based Component Factory, and the Mature Software Organization all have a competency focus, acknowledging the need for addressing people issues in a software factory, whereas the Generic Software Factory does not address this issue. In the Industrialized Software Factory all projects are standardized, but the other three factory approaches compensate for differences between projects by having reuse of process components build into the organization of the software factory.

The Experience-based Component Factory and the Mature Software Organization are the two factory approaches that put the most emphasis on the implementation of improvement, and the most emphasis on describing a dedicated improvement organization. In the Experience-based Component Factory an incremental management approach is recommended, starting by understanding the current state-of-the-art in the organization. In the Mature Software Organization strong emphasis is put on management issues both in the process of establishing the software factory and in organization of the software factory. Software engineering process groups and commitment processes are used and key to the Mature Software Organization management concept. The Industrialized Software Factory has quality circles, but the Generic Software Factory does not suggest any guides or thoughts related to managing the process of creating a software factory.

II.2. Strengths and weaknesses

This schematic highlighting of similarities and differences elucidates some key distinguishing features between the four approaches as presented in figure 4. The four approaches fall into two groups, one focusing on the creation of an infrastructure to support the software process (the Industrialized Software Organization and the Generic Software Factory), and another focusing on developing optimal software processes based on experience (the Experience-based

Component Factory and the Mature Software Organization). The key distinguishing features in the first group are: the Industrialized Software Organization provides a specific model of the physical infrastructure and related organizational features; the Generic Software Factory provides a model that can be used to generate a technological infrastructure customized to meet the requirements of a specific software process. The key distinguishing features in the second group are: the Experience-based Component Factory relies on a continuous improvement strategy based on situational learning; the Mature Software Organization relies on a stepwise improvement strategy based on general process requirements.

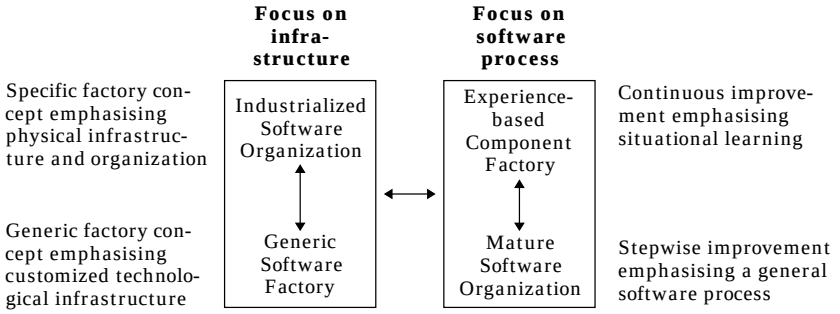

Figure 4. The key distinguishing features between the software factories.

Based on this understanding we can evaluate the most important strengths and weaknesses of each of the four approaches, see table 2. The major contributions from the Industrialized Software Factory approach are first of all the highlighting of how dependent software processes are upon having elaborate infrastructures, and the insistence of viewing these infrastructure as a whole consisting of physical, organizational, and technological elements. On the other hand, this approach may nurture the illusion that complex organizational issues can be managed through structural intervention. The limited emphasis on the dependency upon the life-cycle model and the cultural environment of the factory may furthermore lead to the illusion that the approach has quite general applicability.

The Generic Software Factory approach points to the importance of powerful and tailorable technological infrastructures. The elaborate focus on workbench features forms a major contribution to

Approach	Strengths	Weaknesses
Industrialized Software Factory	• Contemporary software processes depend on elaborate infrastructures • A holistic view of infrastructure covering physical infrastructure, organization and technology	• Managing complex organizational issues primarily through structural intervention • Applicability dependent on life-cycle model and cultural environment
Generic Software Factory	• Contemporary software processes depend on elaborate infrastructures • Elaborate focus on workbench features • Tailorable to specific process models	• Managing complex organizational issues primarily through structural intervention • Narrow focus on technology at the expense of organizational and cultural conditions
Experience-based Component Factory	• Professional praxis depends primarily on process features • Focus on reusable components • Incremental learning based on experiences	• Limited attention to infrastructural support • Conservative • Limited attention to implementation of improvement
Mature Software Organization	• Professional praxis depends primarily on process features • Provides managerial guidance on prioritizing initiatives for any software process • Stepwise enhancement adding new process areas and stabilizing existing ones	• Limited attention to infrastructural support • Abstract

Table 2. Strengths and weaknesses of the four approaches.

the development of technological infrastructures for software factories. The strong and narrow focus on a technical solution and the weak emphasis on managing its actual implementation in specific settings do, however, support the illusion that technology is not only a necessary but also a sufficient means for building and implementing successful software factories for specific project requirements.

The Experience-based Component Factory highlights the importance of process features for professional praxis and contributes specifically to the organization of how to build reusable components and how to facilitate incremental learning based on the experiences gained in the organization. This approach, however, gives only limited attention to infrastructural support and it is conservative in

the sense of mainly dealing with the existing organization with little focus on future demands and opportunities. This may give an illusionary belief that organization and learning is what matters when building software factories, and that incremental learning in itself will prepare the organization to deal with changing conditions.

Finally, the Mature Software Organization approach also highlights the importance of process features. But in this approach focus is on managerial guidance. The guidelines for incremental enhancement of the software process forms an important contribution to the development of a professional praxis in software factories. Like the Experience-based Component Factory this approach pays little attention to infrastructural support and, specifically for this approach, recommendations are given in an abstract and general form with few guidelines for evaluating the effectiveness of a specific set of software processes. This may give way to the idealistic illusion that process improvement is mainly a matter of managing commitments within an organization.

Conclusion

The paper has discussed contrasting views on software factories by providing a survey of software factory concepts, by highlighting variations and differences in the underlying approaches, and by discussing relative strengths and weaknesses of different approaches to professionalize software operations. Based on a comparison of four well-known factory, or factory-like, approaches to professionalize software operations useful contributions and possible illusions related to the idea of a software factory have been identified.

Although very different by nature and in contents the four approaches provide important contributions to professionalize and mature software organizations. Building an effective and efficient software organization requires a powerful combination of suitable infrastructures, process features, and managerial guidelines. The four approaches are important contributions towards this goal. At the same time the approaches individually may lead to unfortunate illusions. Learning from the relative strengths and weaknesses between the approaches may help us avoid becoming victims of these illusions.

On a more general level, all four concepts signal a commitment to efforts to enhance software operations above the level of individ-

ual projects, and—even though the approaches are limited in scope and emphasis—they all integrate a variety of means into an ambitious strategy. Still, the term software factory has the connotation that software development and maintenance is comparable to mass-production of industrial products, and arguably this is not the case. In this respect, the mature software organization offers a rhetoric, which signals the same level of commitment while being more in line with the characteristics of our profession.

We agree with Cusumano that the challenge for software management is to find ways to "improve *organizational* skills—not just in one project but across a stream of projects. To accomplish this, however, and still meet the demands of customers, competitors, and the technology itself, requires firms to balance two seemingly contradictory ends: efficiency and flexibility" (Cusumano 1991, p. 5). The inherent complexities involved in developing and maintaining software suggest that the appropriate organizational form for a software operation is the professional bureaucracy in which professional competence is viewed as more important than standardized procedures and advanced technologies. Software managers are therefore advised to view the professional bureaucracy as the ideal and dominant form while elements of the machine bureaucracy and other organizational forms (Mintzberg 1983) should be treated as supplements to cope with variations, to increase efficiency whenever industrialized procedures are feasible, and to allow for greater flexibility in unique situations where existing procedures and experiences are insufficient. For this reason any long-term management commitment to improve software operations should fundamentally be based on approaches focusing on software processes, see figure 4, and view approaches focusing on infrastructure as supplementary strategies that can help develop environments in which processes and professionals are better supported.

Acknowledgments

Part of this work was funded by the Danish National Center for IT Research, the Danish Natural Science Research Council (Grant No. 9400911), and the Danish Academy of Technical Sciences (Grant no. EF 516). We also wish to thank our colleagues Lars Bendix, Lars Bo Eriksen, Birgitte Krogh, and Peter Axel Nielsen for valuable discussions.

References

Basili, V. R. (1989): The Experience Factory: Packaging Software Experience. *Proceedings of the 14th Annual Software Engineering Workshop, NASA Goddard Space Flight Center, Greenbelt MD 20771.*

Basili, V. R., G. Caldiera & G. Canone (1992): A Reference Architecture for the Component Factory. *ACM Transactions on Software Engineering and Methodology.*

Basili, V. R. (1993): The Experience Factory and its Relationship to Other Improvement Paradigms, *4th European Software Engineering Conference - ESEC '93.* Springer-Verlag.

Bemer, R. W. (1969): Position papers for Panel Discussion: The Economics of Program Production. In A. J. H. Morrell (Ed.): *Information Processing 68.* Amsterdam: North-Holland.

Cusumano, M. A. (1991): *Japan's Software Factories.* Oxford University Press.

Cusumano, M. A. (1989): The Software Factory: A Historical Interpretation. *IEEE Software,* March.

Fernström, C. (1991): The Eureka Software Factory: Concepts and Accomplishments. In A. Lamsweerde *et al.* (Eds.): *Proceedings of the 3rd European Software Engineering Conference.* Lecture Notes in Computer Science No. 550: Springer-Verlag.

Fernström, C., K-H. Närfelt & L. Ohlsson (1992): Software Factory Principles, Architecture, and Experiments. *IEEE Software,* March.

Fowler, P. & S. Rifkin (1990): *Software Engineering Process Group Guide* (CMU/SEI-90-TR-24). Software Engineering Institute, Carnegie Mellon University.

Hayes, W. & D. Zubrow (1995): *Moving on Up: Data and Experience Doing CMM-Based Software Process Improvement* (CMU/SEI-95-TR-008). Software Engineering Institute, Carnegie Mellon University.

Herbsleb, J., A. Carleton *et al.* (1984): *Benefits of CMM-Based Software Process Improvement: Initial Results* (CMU/SEI-94-TR-013). Software Engineering Institute, Carnegie Mellon University.

Matsumoto, Y. (1981): SWB System: A Software Factory. In H. Hunke (Ed.): *Software-Engineering Environments.* Amsterdam: North-Holland.

Matsumoto, Y. (1987): A Software Factory: An Overall Approach to Software Production, In P. Freeman (Ed.): *Software Reusability, IEEE.*

McIlroy, M. D. (1969): Mass-Produced Software Components. In *Software Engineering: Reports on a Conference Sponsored by NATO Science Committee.* Brussels.

Mintzberg, H. (1983): *Structures in Fives: Designing Effective Organizations.* Prentice-Hall.

Nilsson, E. G. (1990): CASE Tools and Software Factories. In B. Steinholz *et al.* (Eds.): *Lecture Notes on Computer Science.* Berlin: Springer-Verlag.

Paulk, M. C., B. Curtis, M. B. Chrissis & C. V. Weber (1993a): *Capability Maturity Model for Software (Version 1.1)* (SEI/CMU-93-TR-24). Software Engineering Institute, Carnegie Mellon University.

Paulk, M. C., C. V. Weber, S. M. Garcia, M. B. Chrissis & M. Bush (1993b): *Key Practices of the Capability Maturity Model, Version 1.1* (CMU/SEI-93-TR-25). Software Engineering Institute, Carnegie Mellon University.