International Journal of Engineering Science Invention
ISSN (Online): 2319 – 6734, ISSN (Print): 2319 – 6726
www.ijesi.org ||Volume 5 Issue 12|| December 2016 || PP. 01-06

A new perspective on software factory for the development of mobile applications

Magdi Amer[1], Noha Elayoty[2]
[1]*Faculty of Computer & Information Systems, Umm Al-Qura University, Saudi Arabia*
[2]*Project Management Consultant, Canada*

ABSTRACT: *Software factory concept was the result of the desire to apply the concepts of industrial factories to software production. Currently, Software factories' main concern is process standardization and product lines. This paper attempts to expand the concept of software factory to cover the development of mobile applications. A new modular approach is suggested and a roadmap for implementing software factory is presented.*

KEYWORDS *Software Factory, Development of Mobile Applications, Software Engineering*

I. INTRODUCTION

The concept of software factory was invented by pioneers [1, 2, 3] that believed that software production should use techniques similar to those techniques used in industrial product factories.

The goal of a Software factory is to transform software production from a "craft or job-shop mode, to more systematically organized modes of engineering and manufacturing" [3]. The first time the term software factory was devised was by R.W. Bemer in 1969 [2, 3]. He defined the software factory as "a programming environment residing upon and controlled by a computer. Program construction, checkout and usage should be done entirely within this environment and by using the tools contained in the environment" [1].

The first software factory established in the United States was by System Development Corporation (SDC) in 1975. SDC built a software factory consisting of an integrated set of tools and programs, procedures and management policies for program design and implementation and a matrix organization, separating high-level system design from program development [3]. According to [4], the main feature that defines the software factory is the standardization of tools, frameworks, processes, project monitoring techniques and software quality measurements.

The term Software Factory was used in Japan since 1969 with a different meaning. The entire teams responsible for the architecture, design, code and test were considered part of the 'factory', while phases such as requirement and maintenance were considered outside the scope of the factory [5]. In this context, software factory was used to indicate the transformation of software development from an unstructured service to a product with a guaranteed level of quality and the increase of software productivity and reliability improvement through process standardization and control [2, 6]. The use of software factory in this context is a generalization of process standardization models such as ISO9001 [7] and Capability Maturity Model Integration (CMMI) [8] [5, 9]. Overtime, software factory focused more on product line. Product lines are applications that share features, functionality and architecture. In this context, software factory is defined as the set of development processes and software assets for developing instances of product line [10, 11].

The work on software factory motivated the work on standardization of software development environment, tools and processes. Although this was an important step in the right path, the implementation of the factory model in the software industry failed to capture all the benefits of applying the factory model in the manufacturing industry.

The goal of this paper is to propose a new perspective on software factory, discuss its impact and present a roadmap on its implementation.

II. THE NEED FOR A NEW PERSPECTIVE ON SOFTWARE FACTORY

The three components that define software factory are: people, processes and tools.

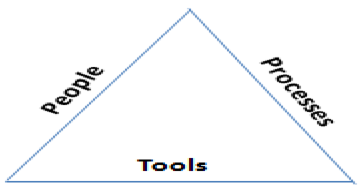

Fig. 1: The three components of a software factory

Process Standardization is a main component of the software factory. Several techniques for process standardization and definition, such as ISO9001 and CMMI, have been defined. The software standardization must be applied to all processes, including technical documentation, specification, management, testing, monitoring and so on. Process definition must be extended to cover all aspect of using the software factory, including the internal processes within a software factory and the processes that define the interaction between the project team and the software factories.

The second component of the software factory is the set of tools. 'Tools' in this context refers to the set of software assets. This includes software architecture, frameworks, prototypes, software components, standards, best practices, pitfalls and design patterns. It represents the translation of the accumulated expertise of the organization and is a continuous effort from the leaders of the software factory to build on the previous experience and to investigate new emerging technologies.

The third component of the software factory is people. This component has been neglected previously in software factory implementation, although it is one of the most important aspects of the software factory. In fact, the first description of factories and their benefits was provided by Adam Smith [12]. In his book, Smith explained the benefit of division of labor using the example of pin production. To teach a single worker all the steps required to produce a pin would require a long period of training, thus increasing the amount of overhead and time to market. Moreover, the level of expertise of the worker in some tasks will be lower than others. The result is a product with low quality and a low product production rate. Dividing the pin production into several tasks and training each worker only in the tasks that he or she will be performing would reduce the time and cost of the training considerably. Because the worker will only perform a limited number of tasks within the cycle of pin production, the worker will become an expert in these tasks, producing higher quality products with a faster rate. Tools and machines can be used to automate the execution of some steps of the pin production cycle. Only the worker responsible for these steps will need to be trained on using the machines that automates these steps. The workers operating the machines need more experience and professional training while the rest of the factory workers will only require basic training for their manual tasks.

The concept explained above of having a worker that is specialized in a specific area to reduce the training cost and to increase the productivity and the quality of the product has not been applied previously to the software factory model. The importance of such a concept has become significant with the increased demand for mobile applications, which resulted in the appearance of new types of problems. Mobile Development projects are characterized by the small project time span and the large variety in the specification, requirements and technologies from one project to another. Nevertheless, these projects share a common set of limitations and technical challenges, such as security risks, limited processing power, low network reliability, small screen and limited input capabilities [13, 14]. Managing these projects using classical approach will not make them cost effective and a more suitable managing approach was needed. In fact, there are some efforts in finding methodologies for the development of mobile applications [15], but these efforts are limited and "methodology oriented issues still remain to be properly addressed" [16].

Due to the characteristics of mobile applications' development, it is a real challenge to build a profitable business unit specialized in mobile development. In fact, mobile application tools, techniques and design patterns are quite different from classical development projects, such as web development and desktop development, making the training required to master this field rather long. Moreover, the sizes of mobile applications' projects are generally small with short time to deliver, which makes the training of programmers on the project usually unfeasible. Furthermore, there are large variations in the requirements, features and technologies from one mobile project to another and the advancements in the field occur with high pace, which will require a new set of training before the programmer can contribute to the new project. Due to the short duration of mobile projects, the overhead of such training will not make the project development profitable. Using classical management approach, where a new programmer is trained in all aspect of the field before joining a project as a junior programmer and taking several months before being productive with minimum supervision, is no longer feasible.

In a non-factory approach, ensuring the quality of software is done through training. The amount of training required to be taken by a programmer to write high quality codes is very large. A programmer that went through all this training may have a misconception about some of the topics in the training and the result will be a code with lower quality, performance issues or security issues. We are suffering from the symptom explained by Adam Smith, which is the variation in the level of expertise of a worker in different tasks will result in a product with low quality and a lower production rate. For projects with a long time span, there are usually enough resources, time and senior team members to detect errors, to help junior members to improve their coding skills and to rectify these errors. This is not the case for mobile applications' development and there is a need to find a different way for managing these projects.

The new perspective of the software factory may be a valid solution to the problem explained above. Using the software factory approach, the project is divided into a set of components each with a limited scope, such as asynchronous communication with a backend, building an interface for a specific screen resolution and orientation, data persistence on the mobile or handling unreliable network communication issues. Training materials for each of these components are created by the software factory team, each training material limited to a few days. This training material only focuses on how to build the specific component and does not try to train the programmer in the entire field of mobile development. In fact, a member in the software factory can fit perfectly in a role to implement a specific type of component after a limited amount of training with minimum supervision.

Programmers currently waiting for long term assignments will be targeted by these trainings. They will start being productive after a few days. They will be only capable of implementing the specific components they have been trained for, without having to understand the entire programming concepts and tools related to mobile applications' development. Their work within the software factory is usually for a short period of time.

This member will be repeating the same simple task over and over again for different projects. The senior experts in the software factory will be responsible for assessing ensuring the quality of the code produced by junior members. They will also be responsible for assessing new technologies, and devising templates and defining processes to improve the quality of the code and to adapt new coding techniques and technologies. A junior member of the software factory may be trained in multiple tasks to increase his or her utilization and to build his or her career path to become an expert in the field. However, this is a long term plan and this member can be productive after a few days of training on a specific role.

On the other hand, senior members in the mobile software factory are assigned permanently in the factory. They will be periodically targeted with training for sharpening their knowledge and introducing them to latest technology innovations in the field of mobile programming. Their participation in the development of mobile applications will mainly focus on the architectural design of the application, the supervision of the work of the component developers, the integration of these components and the implementation of business logic of the applications.

In the next section, a roadmap for the implementation of software factory is presented.

III. SOFTWARE FACTORY ROADMAP

The software factory's goal is to apply manufacturing techniques and principles to Software Development to use the benefits of traditional manufacturing.

The first step in building software factory is to identify the areas in which the software factory can be implemented based on the company's areas of expertise and its past experience. Example of potential areas where software factory may provide cost-benefits are database layer, backend, validation, code review, unit testing, user interface, security or performance.

The choice on the area to implement as a software factory depends on the past experience of the company as well as the expected business growth of the company. In fact, such a decision should take into consideration the cost of building a software factory and the savings realized due to the higher software quality, lower risk and lower cost of implementing the code components manufactured in the software factory.

Software factory addresses the problem of traditional application development where applications are developed and delivered without taking advantage of the accumulative knowledge gained and the assets produced from developing similar applications. Many approaches, such as training, documentation and frameworks, are used to address this problem; however, using these approaches to consistently apply the valuable knowledge previously gained during development of multiple applications can be an inefficient and error-prone process.

Developing applications using a suitable software factory model can provide many benefits, such as improved productivity, quality, software asset reuse, automation, standardized processes, standardized code style, standardized architecture and standardized process to produce a product.

Projects are the responsibility of the project team and are lead by the project manager. The project manager will delegate the tasks that fall under the area of expertise of the software factories to them. The project manager and the software factory lead will agree on the specification and the acceptance criteria of these components, using templates that were created and standardized by the software factories. The rest of the traditional tasks and components will be implemented by the project team.

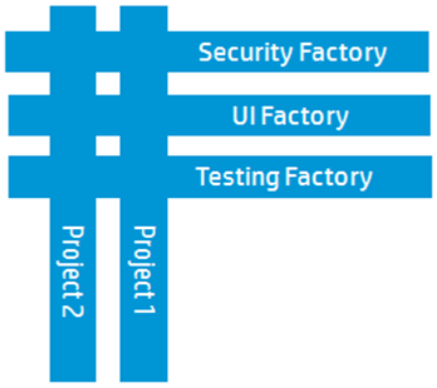

Fig. 2: Project relation with software factories

The first step (Need Analysis) in implementing the software factory is to decide what are the components, tasks or software layers that are eligible to be implemented using a software factory. A good source for discovering potential candidates is the experience factories. Experience factories are repositories for storing information about software projects, both quantitative and qualitative experiences, in the form of products, processes, tools, lessons learned reports, standards, policies, training materials and management tools [17, 18] . Building software factory can only be done based on previous experiences of building systems for a particular domain as the factory development is based on the assets and best practices resulting from these experiences [10].

Experience factory should be used to identify the strength and the weakness of the company. Both the strength areas and weak points have a high potential from benefiting from the software factory model. Building software factories in strength areas and successful project will allow the experiences gained in them to propagate to all the software projects of the company. It will also allow the company to be more competitive in its areas of expertise. Building software factories in weakness areas will allow spreading the techniques and solutions to overcome these weaknesses across the organization.

The second step (Financial Benefit Estimation) is to estimate the cost of building the software factory and the financial return on investment that the software factory has the potential to achieve. In fact, building a software factory will require experts to work for a period of time to establish the software factory, as will be explained in the following paragraphs. During this period, the salaries of these experts and the resources that they consume will represent a cost that the company will have to spend upfront. On the other hand, the software factory will generate revenue as the code will be produced by specialized workers in the software factory under the supervision of the experts. This means that the code will have higher quality, higher reliability and lower risk of logical, security or performance errors. Hence, the cost of risk mitigation will be greatly reduced. The software factory will also increase the company's ability to reuse code and adapt components of previous projects. The decision to build a software factory is a business decision and should only be carried if the expected return on investment justifies the initial cost.

The third step (Initiation) is to build the core team of the software factory. The core team is composed of the experts in the field of the software factory. Their task is to build the templates, define processes and build prototypes, tools and measurement techniques. The core team will also build templates for estimation, sizing, proposals and documentation. They will also build generic customizable components for the reoccurring tasks.

The software factory should act as an internal subcontractor that will be responsible for the implementation of components within the scope of the software factory. Thus, special care should be given for defining templates and processes for passing customer requirements and technical requirements from the software development team. Processes, templates and tools should also be defined for accepting the components produced by the software factory and evaluating its quality and level of abiding to the specification.

The core team will categorize the service of the software factory in several knowledge sub-areas. The core team will assess the expected work load that will be delegated to the software factory and will calculate the estimated number of members that need to join the software factory team. The core team will access the pool of experts in the organization and select candidates that will participate in the software factory based on these estimations. They will prepare the training program to level up these candidates to cover any gap in their knowledge and expertise. Using the concepts of factory as explained by Adam Smith [12], each new member will receive training only in the sub-area to which he or she is assigned. There is no need to train the team member in every sub-area of the software factory, thus realizing the goals of reducing the time and cost of training new members. For example, if the company is building a software factory for mobile developments, sub-areas will be identified such as UI development, mobile to backend communication, data persistence and security. A member will specialize in one of these sub-areas and will receive the related training in this sub-area only. For other management aspects such as availability, usability and rotation policies, a member may receive training in other sub-areas over time and start playing more than one role within the factory.

The forth step (Production) is to use the software factory within the company to contribute to proposals and project development. To ensure the success of this phase, the company must implement processes to facilitate and ensure the use of software factories within the company. During this phase, project managers will delegate the task of developing some software assets to the software factories after agreeing on the specifications, service level agreement and the acceptance criteria. Software factories will also contribute to project proposals in the service areas the factory provides. The software factory will review the architecture, time estimation and cost estimation of the components for which the software factory will be responsible. The factory will also collect this information to estimate the amount of work that will be needed from the factory.

During this phase, the members of the software factory will increase depending on the projected work load. Software factory leaders will be responsible for training junior members and helping senior members to gain multiple skills within the domain of interest of the software factory to increase their utilization and to maximize the benefit of their accumulated experience.

It should be noted that the implementation of the software factory will be opposed by project managers, who will not like giving up control to a centralized facility. It is the role of top management to enforce the success of the software factory by enforcing its use and mitigating potential friction between the project team and the software factory team [2].

The fifth step (Maturity) is to investigate new technologies and tools and to automate some tasks to achieve higher reliability and lower production cost. Software factory will also maintain Key Performance Indicators (KPIs) to measure different perspectives such as the productivity, quality, cost reduction and risk mitigation that resulted from the use of the software factory. During this phase, the financial performance of the software factory is evaluated and corrective actions may be taken to ensure that the software factory is having a positive financial impact. A decision to terminate the factory may be taken otherwise, especially when changes in the technology make the domain of the software factory obsolete.

When deciding to build a software factory, the factory may be organized vertically as a function of the business area such as mobile development or web development. Software factories may also be organized horizontally as a function of the software development phase or layer, such as a user interface or testing. A company may decide to use only horizontal software factories, only vertical software factories or a combination of both.

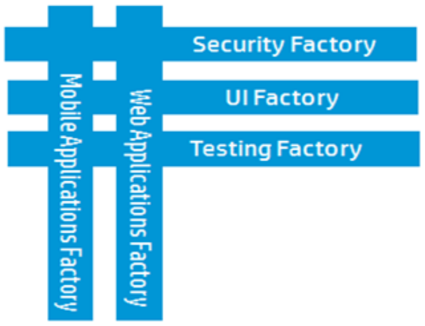

Fig. 3: Software Factories organization

Fig. 3 shows an example of two different ways to organize the software factory. Using the vertical approach, the mobility factory is responsible for the development of the mobile user interface, the core mobile applications' code and the testing of the mobile applications. Similarly, the web factory is responsible for the development of the web interface, the core web application code and the testing of the web application.

Another possible organization is using the horizontal approach. In this case, a software factory for the user interface will be built containing both mobile user interface and web user interface, another for core software development will be used for both web and mobile coding and a third testing software factory will be used for testing on web and mobile applications.

The decision of the company to adopt which approach is a business decision that may vary from one company to another. Nevertheless, no matter whether the decision of the company was to use a vertical or horizontal approach, an entity in the opposite orientation must be created to ensure collaboration and experience sharing.

For example, let’s suppose that a company has decided to adopt the vertical approach. Experience in building user interface or testing mobile applications and web applications must be shared. Tools and scripts for automating some of the web user interface tasks may be ported to the mobile user interface to increase quality and reduce cost. Pitfalls and best practices in on area can probably be adapted to the other area. A vertical entity will be needed to help transfer the experience of software development from one area to another.

Similarly, if a company has decided to adopt the horizontal approach, an entity responsible for the mobile applications and another for the web application development will be needed. For example, the technical knowhow gained in the development of the core mobile or web applications must be shared with the user interface factory and testing factory as this will probably help improve their techniques and processes. Knowledge in the user interface development and pitfalls discovered during testing should also affect the development of the core software. Horizontal entities will be needed to ensure the knowledge and experience sharing.

IV. CONCLUSION

The concept of factory has been the main driver for the industrial revolution due to the economical impact of using factories in product development [12].

There has been a lot of effort to implement the concept of factory in the software development. Previous efforts have been focusing on the process and tool aspects of the factory and have neglected applying the factory concept on people.

In this paper, a roadmap to the implementation of the software factory for mobile applications’ development was proposed with the explanation on how it can be implemented on the three areas of the software factory; people, tools and processes.

The impact of the implementation of the software factory in mobile applications’ development was proposed as a way of coping with the special challenges facing mobile applications’ development while keeping the development process cost effective.

REFERENCES

Journal Papers:

[1] R.W. Bemer, Position Paper for Panel Discussion on the Economics of Program Production, Information Processing 68, Amsterdam: North-Holland Publishing Company, 1969

[2] M.A. Cusumano, The Software Factory: A Historical Interpretation, *IEEE Software, Volume:6, Issue:2*, IEEE, 1989, pages: 23-30.

[3] M.A. Cusumano, Factory Concepts and Practices in Software Development, in *Annals of the History of Computing, Volume:13 , Issue: 1*, IEEE, 1991, pages 2-32.

[4] H. Bratman and T. Court, Elements of the Software Factory: Standards, Procedures, and Tools, *Software Engineering Techniques*, Infotech International Ltd., Berkshire, England, 1977, pages 117-143.

[5] H.P. Siy, J.D. Herbsleb, A. Mockus, M. Krishnan, G.T. Tucker, Making the Software Factory Work: Lessons from a Decade of Experience, *Proceedings of the Seventh International Software Metrics Symposium*, IEEE, 2001, pages 317-326.

[6] M.A. Cusumano, *Japan's Software Factories*, Oxford University Press, New York, 1991, 528 pages.

[7] The International Organization for Standardization (ISO), ISO 9000 quality management, http://www.iso.org/iso/home/standards/management-standards/iso_9000.htm, (accessed 2016/11/1).

[8] The Capability Maturity Model Integration Institute (CMMI), The Capability Maturity Model Integration (CMMI), http://cmmiinstitute.com/build-organizational-capability, (accessed 2016/11/1).

[9] C. Li, H. Li, M. Li, A software factory model based on ISO9000 and CMM for Chinese small organizations, *Proceedings of the Second Asia-Pacific Conference on Quality Software*, IEEE, 2001, pages 288-292.

[10] C. Wienands, G. Lenz, *Practical Software Factories in .NET*, Apress, 2006, 214 pages

[11] J.V. Gurp, J. Bosch, M. Svahnberg; On the Notion of Variability in Software Product Lines, *Proceedings of Working IEEE/IFIP Conference on Software Architecture*, IEEE, 2001, pages 45-54.

[12] Adam Smith, *An Inquiry into the Nature and Causes of the Wealth of Nations*, 1776

[13] Y.J. Jeong, J.H. Lee, G.S. Shin, Development Process of Mobile Application SW Based on Agile Methodology, *Proceedings of the 10th International Conference on Advanced Communication Technology*, IEEE, 2008, pages 362-366.

[14] V. Rahimian, R. Ramsin, Designing an agile methodology for mobile software development: A hybrid method engineering approach, *Proceedings of the Second International Conference on Research Challenges in Information Science*, IEEE, 2008.

[15] D.M. Mahmud, N.A.S. Abdullah, Reviews on agile methods in mobile application development process, *Proceedings of 9th Malaysian Software Engineering Conference*, IEEE, 2015, pages 616-165.

[16] Z. Stapić, M. Mijač, V. Strahonja, Methodologies for development of mobile applications, *Proceedings of the 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, IEEE, 2016, pages 688-692

[17] M. Hanafiah , R. Abdullah , J. Din, M.A.A. Murad, Towards developing lessons learned and experience based factory in software development, *Proceedings of the 4th International Conference on Software Engineering and Computer Systems (ICSECS)*, IEEE, 2015, pages 102-106.

[18] J.Rech, E.Ras, Aggregation of experiences in experience factories into software patterns, *ACM SIGSOFT Software Engineering Notes archive, Volume 36 Issue 2*, ACM New York, NY, USA, 2011, pages 1-4.