Software Factories Relevance to Digital Games: A Practical Discussion

André W. B. Furtado André L. M. Santos

Federal University of Pernambuco, Informatics Center, Brazil

Abstract

Software factories are focused on promoting automation, enforcing predictability and stimulating reuse, in order to turn craftsmanship into a manufacturing process. Digital games development, on the other hand, is an inherently creative discipline, a peculiar domain heavily characterized by innovation and dynamism. Given such a context, this poster provides a substantiated and practical discussion, based on software factory fundamentals, on how deeply digital game development is subject of industrialization by means of software factories.

Keywords: digital games development, software factories.

Authors' contact:
{awbf,alms}@cin.ufpe.br

1. Introduction

Software factories are focused on the transition of software development from craftsmanship to manufacturing. In short, a software factory can be defined as a software product line that configures extensible development tools and processes with packaged content and guidance, carefully designed for building and maintaining variants of an archetypal product by adapting, assembling and configuring framework-based components [Greenfield et al. 2004].

On the other hand, digital games development is an inherently creative discipline, perhaps more than any other Computer Science domain. Uniqueness and innovation are intrinsic attributes of successful game titles. This raises a discussion about how deep digital game development is subject of industrialization by means of software factories. Such a discussion is carried thorough this poster by analyzing the challenges and implications of software factories under the digital games development point-of-view.

The remainder of this poster is organized as follows. Section 2 establishes the differences between economies of scale and scope, which are key to understand software factories. Section 3 analyses digital games development in the context of the challenges and expected major implications of software factories. Section 4, finally, concludes about the presented research.

2. Economies of Scale and Scope

Software factories integrate critical innovations to define a highly automated approach to software development that exploits *economies of scope*, in contrast to *economies of scale*.

Economies of scale occur when multiple identical instances of a single design are produced collectively, rather than individually, as illustrated in Figure 1 [Greenfield 2004]. They arise in the production of things like machine screws, when production assets like machine tools are used to produce multiple identical product instances. A design is created, along with initial instances (prototypes), by a resource-intensive process, called development, performed by engineers. Many additional instances (copies) are then produced by another process, called production, performed by machines and/or low-cost labor, in order to satisfy market demand.

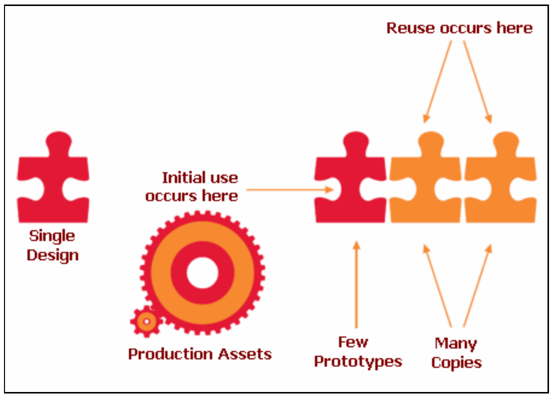

Figure 1: Economies of Scale [Greenfield 2004]

Economies of scope arise when multiple similar but distinct designs and prototypes are produced collectively, rather than individually, as illustrated in Figure 2 [Greenfield 2004]. In automobile manufacturing, for example, multiple similar but distinct automobile designs are often developed by composing existing designs for subcomponents, such as the chassis, body, interior and drive train, and variants or models are often created by varying features, such as engine and trim level, in existing designs. In other words, economies of scope reduce the cost of solving multiple similar but distinct problems in a given domain by collectively solving their common sub-problems and then assembling, adapting and configuring the resulting partial solutions to solve the top-level problems. Economies of scope can be complemented by economies of scale, as illustrated by the copies of each prototype shown in Figure 2.

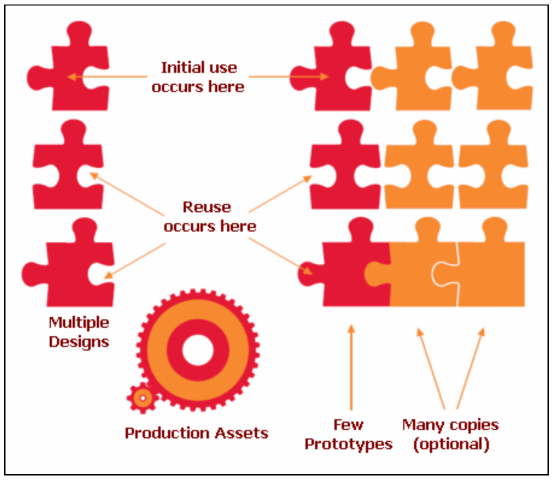

Figure 2: Economies of Scope [Greenfield 2004]

Economies of scale arise in production, while economies of scope arise in development. In heavy industries, producing copies consumes expensive resources, while in software it consumes inexpensive digital and paper media. Another difference is that software is more expensive to consume because of customization. Economies of scale are lost when high levels of customization are present.

Production in physical industries, however, has been naively compared with development in software [Greenfield 2004]. It makes no sense to look for economies of scale in development of any kind, whether of software or of physical goods. We can, however, expect the industrialization of software development to exploit economies of scope. This is where software factories are situated.

3. Factory Challenges and Implications vs. Digital Games Development

In order to check how software factories challenges and implications affect digital games development, it is important to identify the scenarios where software development may exhibit economies of scale and when it may exhibit economies of scope. Such distinction follows below [Greenfield 2004]:

- In markets like the consumer desktop, where copies of products like operating systems and productivity applications are mass produced, software exhibits economies of scale.
- In markets like the enterprise, where business applications developed for competitive advantage are seldom, if ever, mass produced, software exhibits only economies of scope.

At a first glance, one could state that the first scenario (economies of scale) is the single case where digital games development is inserted. Therefore, since software factories are expected to exploit economies of scope, game development would not be so benefited from them. Game development, however, behaves similarly to the automobile industry regarding economies of scale and scope. Once a digital game title can be considered a COTS (commercial off-the-shelf) product, it can certainly be exploited by economies of scale as an automobile model with a specific configuration can. Nevertheless, in the same way economies of scope are used for an automobile model customization (color, engine, chassis, accessories, etc.), it can also be used for creating (configuring) games belonging to a same genre. In resume, game development falls in the case where economies of scope are complemented by economies of scale.

While these arguments make it reasonable to say that digital games development is subject of industrialization by software factories, they do not answer the question of which factory benefits games development can take advantage of, and which may not be applied at all. Since digital games are targeted at players, not at enterprises (which chiefly exploit economies of scope), the tendencies are that not all of the industrialization can be applied to the domain. For example, some concerns exploited by software factories, such as business requirements and deployment, are more critical to enterprise software development than to digital games development.

In order to have a clearer photograph of such discussion, this research analyzed the challenges focused by software factories and how they threaten (are relevant to) digital games development. The results are displayed in the following subsections, each one grouping a set of serious challenges faced by the development of new applications (according to Greenfield et al. 2004) and their relevance level to digital games (possible values are Low, Medium and High).

3.1 New business requirements

Challenge: Support reengineered business processes and an increasing focus on process-oriented applications.
Relevance: Low. Apart from “serious games”, games are generally not focused on business processes.

Challenge: Expose existing systems to massive user loads created by web-based front ends.
Relevance: High. Many successful titles can be played online.

Challenge: Design protocols (valid message sequences) and enforceable service level agreements to support processes than can span multiple enterprises.
Relevance: Medium. Communication protocols are needed, but services that span multiple enterprises are rare in games.

Challenge: Determine strategies for versioned data and snapshots, such as price lists, that are widely distributed yet have limited lifetimes.

Relevance: High. In games, such data are represented by items such as demos, maps, scenarios, high-score tables, saved games and so on.

Challenge: Cope with the complexity created by transformed business models from business leaders such as Wal-Mart, who insist on deep integration with their partners.
Relevance: High. Such a challenge impacts in revenue sharing models in digital games. Besides that, game genres, gameplay experiences and interaction paradigms keep evolving, as well as engines and tools.

Challenge: Integrate heterogeneous application stovepipes and avoiding lossy transformations between them.
Relevance: Medium. Integration is done for components like servers and game engines, but it is not so heterogeneous.

Challenge: Avoid reintroducing the batch era problem of multiple file layouts and lack of data format consistency in the rush to describe XML schemas for every software service.
Relevance: High. Standards are desired for saving games, defining scenarios, distributed communication and so on.

Challenge: Determine strategies for wrapping older applications executing on heterogeneous platforms.
Relevance: Low. Legacy is not a recurring problem in game development.

Challenge: Customize packaged software to satisfy proprietary requirements.
Relevance: High. Customization can arise in the form of map/level editors, skins and MODs[1].

Challenge: Address the special need of data warehouse and business intelligence.
Relevance: Medium. While games generally do not involve data warehousing, business intelligence is crucial to some domains such as mobile device games.

Challenge: Demonstrate return on investment in custom software in the face of rising software development costs.
Relevance: High. Game engines are a formidable example of a high effort which can provide return of investment by means of customization.

3.2 Increasing focus on security

Challenge: Protect corporate data from hackers and viruses, while giving customers and partners direct access to the same resources.
Relevance: High. Games should not attempt to access unauthorized data or execute unauthorized operations; cheaters and hackers are a constant annoyance in online versions.

Challenge: Mitigate the increasing risk of legal liability from the improper use of corporate date.
Relevance: Medium. Although game data are generally not corporate and/or confident data, part of a users profile must be kept private.

3.3 Increasing deployment complexity

Challenge: Satisfy operational requirements, such as security, reliability, performance and supportability, in rapidly changing applications.
Relevance: High. Some of such concerns are critical to digital games.

Challenge: Integrate new and existing systems using a wide range of architectures and implementation technologies.
Relevance: High. Integration with game engines, tools and even servers is heavily demanded.

Challenge: Understand the effects of partitioning and distribution on aggregate qualities of service
Relevance: Medium. Quality of service is impacted by distribution in games by only a few variables, such as performance.

Challenge: Design multitiered applications that deploy correctly to server farms on segmented networks partitioned by firewalls, with each server running a mixture of widely varying host software configurations.
Relevance: Medium. This issue may appear only in some scenarios of online games, such as MMOGs (Massively Multiplayer Online Games).

3.4 Decentralized software development

Challenge: Support applications developed by end-users (for example, spreadsheets to 4GLs) while enforcing corporate policy and maintaining the integrity of corporate data.
Relevance: High. This concern rises in game development by means of map/level editors, skins, MODs or event behavior definition by players through script languages.

Challenge: Integrate personal productivity applications such as word processors and spreadsheets, with back end systems.
Relevance: Low. Such a challenge does not apply, since these applications are generally not integrated with digital games.

Challenge: Work with applications or components that are increasingly outsourced to development centers in remote locations forcing a discipline on requirements designs and acceptance tests that was often neglected in the past.

[1] MOD, or "Modification", is a package that can be applied to a game to modify its appearance (graphics, sounds, texts, etc.) or even its behavior. One of the most famous MODs is the Counter-Strike MOD [Valve 2006], which can be applied to the Half-Life game.

Relevance: Medium. Outsourcing may appear in the form of graphics modeling, artificial intelligence programming, level design, custom tool creation and other tasks, although this is not very common today.

Challenge: Make departmental application integrate effectively and scale to satisfy enterprise requirements.
Relevance: Low. Enterprise scalability makes no sense to digital games.

3.5 Software Development Implications

A similar analysis was done to the expected major implications of software factories. Such implications (also according to Greenfield et al. 2004) were analyzed and their relevance to computers game development was depicted, as shown below.

Implication: Development by assembly: only a small part of each application will be developed from scratch.
Relevance: High. Many aspects of game development, such as map generation and entity rendering, for example, are positively impacted by this implication.

Implication: Software supply chains: each participant consumes logical and/or physical products from one or more upstream suppliers, adds value, usually by incorporating them into more complex products, and supplies the results to downstream consumers.
Relevance: Low. Despite supply chains can be applied to some scenarios, such as graphics modeling, they are more applicable to other domains than game development.

Implication: Standardization of specification formats, packaging formats, architectures and patterns.
Relevance: High. Games development deeply welcomes such standardization. However, this challenge may be hard to overcome due to industrial secrecy and piracy issues.

Implication: Relationship management: managing costumer and relationship will become more important.
Relevance: Medium. A "digital game requirement" is not so well defined as in other domains, while beta-testers and publishers still have to be dealt with (obs: "costumers" should not be mistaken for "end users").

Implication: Domain specific assets: product line developers will build assets used by product developers.
Relevance: High. Game engines, language-based tools and other frameworks/tools are genuine examples of product line assets used by game developers.

Implication: Organizational changes: much about development and development organizations will change.
Relevance: High. Such an implication is relevant to virtually all software development domains.

Implication: Mass customization of software: software may eventually be mass customized like PCs ordered on the web today.
Relevance: Medium. Letting the player to completely configure and customize a game genre in order to create its own instance, and then order it, is still a novel scenario.

4. Conclusions

The study previously presented makes it possible to conclude that, although digital games development is clearly a domain where software factories are less effective than domains which deeply exploit economies of scope and customization, the number of addressed challenges and positive implications makes it worth to try the approach.

In addition, the major requirement for establishing a software factory (ongoing demands for solving problems in a common domain) is clearly present in game development and makes the use of the approach even more encouraging. Therefore, this research believes that there is enough evidence that the same way software factories provide sets of abstractions customized to meet the needs of specific families of systems, like e-commerce, financial arbitrage, or home banking applications, digital games belonging to a specific genre (such as 2D board games, 2D arcade games, 2D ½ isometric strategy games or 3D first-person shooters) may also be considered as families of systems which could benefit from software factories.

Finally, the productivity and automation provided by single factory fundamentals alone, such as modeling through visual domain-specific languages (DSLs) [Deursen et al. 2000] and code generators, are worthwhile additions to games development even without a complete software factories background.

References

DEURSEN, A., KLINT, P. AND VISSER, J., 2000. *Domain-Specific Languages: An Annotated Bibliography* [online] Available from: homepages.cwi.nl/~arie/papers/dslbib [Accessed 31 August 2006].

GREENFIELD, J., SHORT, K., COOK, S., KENT, S. AND CRUPI, J., 2004. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley & Sons, 2004.

GREENFIELD, J., 2004. *The Case for Software Factories* [online] Microsoft Developers Network. Available from: http://msdn.microsoft.com/vstudio/teamsystem/workshop/sf/default.aspx?pull=/library/en-us/dnmaj/html/aj3softfac.asp [Accessed 28 August 2006]

VALVE, 2006. *Counter-Strike Source* [online] Available from: www.counter-strike.net [Accessed 31 August 2006].