Toward Adopting Self-Organizing Models for the *Gamification* of Context-Aware User Applications

Daniel J. Dubois
Politecnico di Milano
Dipartimento di Elettronica e Informazione
Milano, Italy

***Abstract*—Recent developments in the area of small and smart devices led to a massive spread of them, which, in some cases are replacing traditional computers for performing common activities such as web browsing. These devices are usually equipped with specialized hardware to sense and interact with the environment. In this context the use of self-organizing techniques has been widely used to provide adaptation capabilities at the low level, such as for optimizing energy consumption, or for providing some fault-tolerance properties to the communication middleware. What we want to show in this work is how the same self-organization principles can be used at the user-experience level of context-aware applications. The approach we propose shows that self-organization can be used to model the introduction of gaming elements to motivate and simplify the use of context-aware applications, thus leading to higher quality software. This work is finally validated using a case study and empirical evidence from existing popular context-aware applications.**

***Keywords*-self-organization; software engineering; human computer interaction; gamification; context-aware systems.**

I. INTRODUCTION

The current trend of personal computers is to have them embedded in portable devices. An example is given by modern touch screen tablets, which are perfect for web browsing, social networking, and email reading and writing. These devices are becoming increasingly interesting also for different kinds of uses since their touch-screen user interfaces are easier to use, they can be moved more easily and have a battery that usually lasts more than the one of a traditional laptop. Moreover these devices are often equipped with many kinds of sensors, such as a compass, a GPS receiver, 3G/WiFi/Bluetooth data connectivity. This makes them able to sense the context in which they are being used and to possibly use such information together with some applications to provide some kind of value-added services.

Besides browsing, messaging, and socializing, one of the most important uses for these devices is to play games. According to popular ranks it has emerged that users prefer to use games that are easy to learn, that keep the users motivated, and that have some component of sociality, collaboration, and competition with other people. Moreover, the most successful ones contain reward mechanisms that keep high the attention of the user and stimulate the search for new achievements.

What motivates our work is to find a way to classify the mechanisms that are widely used in video games in a way that they may be abstracted and used in a wider class of software. The use of these guidelines at the design time of regular software would be the basis for the *gamification* process of such software.

To find proper guidelines we used our background experience on self-organization [1], which is a recurrent technology in distributed context-ware system [2]. Self-organization is a mechanism that is able to give to a system that is composed of smaller interconnected elements a high-level property in a spontaneous way. In other words a self-organizing system is able to converge to a desired state by relying on the interaction of its elements only, rather than relying on a special monitoring/actuating element or any other single decision point. Current mature uses of self-organizing techniques are usually relegated to the lower levels of the applications [3]. For example, they can be used at middleware level to give fault-tolerance to the application or to optimize some other non-functional requirements. The reason for adopting self-organizing models to support our gamification abstractions comes from an observation of the recent developments in the world of videogames. Some relevant examples are the matchmaking and achievements systems, which are calibrated using feedback information from the user that plays and any other user that is playing the game. Another example is the level of variability of the game, which, thanks to internal perturbations and past experiences of other players, is able to create new, often unpredictable challenges, at the right time.

In this work we first classify recurrent ways in which self-organization is used in some existing class of videogames, and then use our classification to derive abstractions/guidelines that can be used by software engineers to improve the usability of their context-aware software, under the assumption that usability is a non-functional requirement that is as important as other non-functional requirements that are addressed by the use of self-organization at the lower levels of the application. In this work we will stress the concept of context-awareness because self-organizing techniques rely not only on the local information of each element, but also on context information such as the location, the time, and any other information regarding the particular situation.

This paper is organized as follows. Section II describes more in general what self-organization is, what motivates its use, and in which contexts it is usually exploited. Moreover it clarifies the concept of gamification and discusses some relevant related work. Section III presents a classification of ways in which self-organization is used in modern games. In Section IV we propose our guidelines that can be used to

instrument context-aware software to improve the user-experience quality. Section V shows how the proposed abstractions can be applied in a case study, how we can find related real world uses of them in some popular software, and what are the social implications of these techniques. Finally, Section VI concludes the paper and shows some future research directions.

II. Background and Related Work

A. Background on Self-organization

Self-organization is a phenomenon defined as "*An increase of order which is not imposed by an external agent (not excluding environmental interactions)*" [1]. This means that, if a system starts in a non-organized form (with respect to a generic property), and ends up in an organized form without any external intervention of the environment or any other entity external to the system, then such system is a self-organizing system. The concept of self-organization has been studied since ancient times because self-organizing systems may be observed everywhere. Examples of self-organization phenomena are the chemical reactions of compounds, the movement of stars and galaxies, the organization of the cells in life beings, the organization of insect colonies, the organization of markets and human society in general, and so on. The challenges of scientists that study self-organization are to understand the basic rules that govern the movement of the system from its initial state to the organized state.

With respect to what we have presented above, recent research [4,5,6] wants to transfer to the area of Software Engineering a systematic way to exploit the most common observed characteristics of natural self-organizing systems, such as the simplicity of the operations of their elements [7], the resistance to levels of randomness in the execution of such operations, and the capability to reach complex global goals. Self-organization is usually engineered in software systems as a set of rules or policies, which guide the evolution of the system through some different internal states [8]. These rules and the specification of these states have in some cases been enclosed in common design patterns that simplify their adoption in existing software architectures as well as new ones in order to run and exploit them [4,5,6].

Since pure architectural aspects are out of the scope of this work, which aims at giving a conceptual methodology to obtain some high-level property, from now on we will focus on the self-organizing rules, and, more in general, to the type of interactions among the system elements.

B. Related Work on Gamification

The term gamification is very recent. One of the first attempts to define it has been made in [9]: "*gamification is defined as the use of game design elements in non-game context*". Even if the research area is new, there is an established literature on this topic. In [10] and [11] for example the authors propose some guidelines for the design and development of games. In these guidelines they consider many key aspects to build successful games that include the necessity of challenge, of interactions, the inclusion of creativity in the gaming experience, and finally a view of games as contexts for social play.

This preliminary literature led to the adoption of the same features that make a game successful to the context of user applications in order to have similar engaging effects [12,13], thus introducing gamification patterns into the design phase of the development process.

In our work we give a contribution to the gamification area by observing and abstracting in existing games some features that are typical of the world of self-organizing systems, such as agonistic (collaborative) and antagonistic (competitive) behavior of self-organizing system elements.

There are some existing works that have already identified the idea of collaboration [14] and competition [15,16] as individual factors for developing gamified applications. In these works the authors observe for example that elements of collaboration/competitions in games can stimulate hobbyists to develop unpaid works. An example is the common creation of the so-called "mods", which are modified versions of existing games developed by volunteers [17]. Moreover, other works analyze how the power of competition may be a huge source of motivation for players to achieve the so-called *pro* status (professional players) [18,19]. The difference between existing works and the one we are proposing is that we model gamification features as self-organizing phenomena. This allows the use of gamification in the same way self-organization is already used to achieve other kinds of high-level goals, such as runtime system adaptation to improve performance.

III. A Classification of Existing Uses of Self-organization in Modern Games

To motivate our classification of self-organization approaches in modern games we have to go back to the time of the first video games. First-generation videogames have usually a fixed scenario; a fixed set of levels, and a final outcome that is "Game Over" (which may be the result of a victory or a defeat). The most primitive achievement mechanism was to define a metric to express the performance of the player through a score, and to save it under the player's name at the end of the game in order to be positioned in a rank with previous players. This preliminary kind of competition motivated players to play more, and it has been a good source of revenue for the owners of coin-operated arcade game machines. This very old model of gaming that looks completely static and centralized can still be modeled as a very simple self-organizing system. The system elements are the following: many players and a single arcade game machine. The players will interact with each other in an indirect way through the arcade game machine and its status changes. A change in the status of the arcade game machine happens when one player adds itself to the global rank of the machine. Such change of status may motivate the player that has been removed from the rank to play again just to have back its leading position in the rank. In this situation the actual game is no longer the game itself, but the fact to end the game in such a way to change the state of the machine. This situation clearly shows that the capability to allow many players to interact to each other

using the machine gives to the system an *emergent property* [20], which is the one to motivate playing more. Since we have an emergent property that is the result from the uncoordinated interaction of different elements, we are in front of a self-organizing phenomenon. Of course in recent times, things changed a lot since the wide use of arcade game machines, and the emergent properties that game designers want to achieve are more articulated than in the past. In the following subsections we will show three classes of recurrent use of self-organization in modern games along with examples of use and high-level properties that are given to the system (a summary of the classification scheme can be seen in Figure 1).

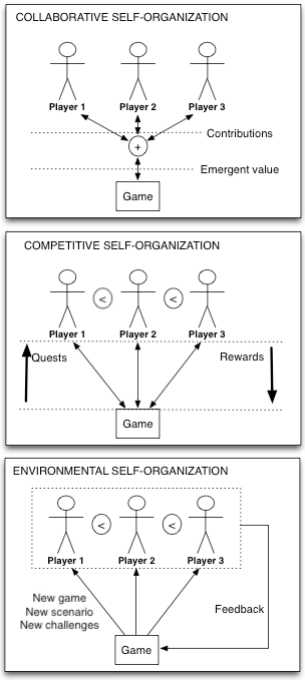


Figure 1. The three classes of self-organization in videogames.

A. Collaborative Self-organization

We define as collaborative self-organization all the social rules in which a player is motivated to give its contribution to a game to make it better for herself or for others. Successful collaborative self-organization can be seen in games in which new content can be created. In this case the creativity of a player is motivated by seeing other players using and extending her creation. Since usually the newly created content remains property of the company that owns the game and its infrastructure, this will create a virtuous circle that creates value for the company (thanks to the free new content available for other players) and for the players themselves that will experience a game with richer contents over time. Some examples of existing collaborative self-organization can be seen in games that are based upon user-generated content, such as Second Life, Little Big Planet, Minecraft, etc. These games come with a level editor that is often difficult to use, but the reward for making available some of their creations and increase their notoriety is enough for many player to climb up the steep learning curve of a difficult editor.

B. Competitive Self-organization

We define as competitive self-organization all the social rules that govern the competition of a player with another player. This form of self-organization is more intuitive since many events of the life of an individual are purely pushed up by competition. To explain why the concept of competition can be modeled as self-organization we have to consider that each player of a game has usually a goal that is decided by the game. If the game has the possibility to use information of its players to adapt its goals (similarly to what happens in the ranking system of the old arcade game machine), then the players are motivated to play more to achieve the new goals. A reiteration of this process will create players that become progressively more skilled and that will help to keep the game popular because they will be trained as good challenging opponents for potential new players. For videogames experts this type of self-organization may be seen trivial, but it is fundamental to explain how this can be used in other types of software.

Examples of modern use of competitive self-organization can be seen in the achievement systems that are becoming popular in any kind of game, such as Starcraft II, and the mobile versions of classical boardgames such as UNO, Monopoly, and many others. These achievement systems will reward users with new game features such as new personalization options, badges, scores, and many other things that are absolutely free for the game provider, but acquire a high value from the point of view of the players.

C. Environmental Self-organization

This third kind of self-organization is defined as the capability of the game environment to spontaneously change its state to create new, unpredictable challenges to the players. The difference between this type of self-organization and the others previously introduced is that this is not derived by social interactions among the players (neither collaborative nor competitive) but it is a spontaneous reaction of the game environment with respect to what the players have done, their context, and the evolution over time. This is usually needed to give variability to the game, so that it is perceived as a different game with new challenges every time it is played. A characteristic of this form of self-organization is that the rules governing it are often non-deterministic, meaning that the final result, although within some predetermined boundaries, is something with a good degree of originality.

Examples of games in which environmental self-organization is used are Diablo and World of Warcraft. In these games there is a map that may change overtime and enemies controlled by the game, which may be influenced by the behavior of other users and evolve autonomously.

IV. Self-organizing Models for User Applications from a Software Engineering Point of View

In this section we will extend the self-organization features highlighted in the classification of Section III and apply them in the wider class of context-aware software. The reason why we emphasize context-aware class of software is given by the fact that in our approach we do not limit our considerations to a simple generic software application, but also to the possibility for the application to interact with the environment, which may contain additional instances of the application or hardware/software artifacts that, by their nature, may be useful to contribute to the global property we want to give to the system.

In this context our goal is to use self-organizing models and the possibility to exploit context-awareness to improve the quality of the software in terms of the following non-functional requirements:

R1. *User Motivation*: motivate the user to learn basic and advanced features of an application to ease its learning curve.

R2. *Context-aware Learning*: adapt the learning curve of the application in such a way that the user learns first what is needed in the specific usage context.

R3. *Reward System*: make sure that, through a proper competition mechanism, a user feels like it can always improve and be better in its experience with an application.

R4. *Collaborative creativity*: if applicable, the user should be able to express its creativity when dealing with a given application, with the purpose of making some of its ideas, contents, or suggestions, available to other users.

All these requirements are non-functional because they aim at improving an application from a different angle with respect to its specific function.

Requirement R1 is fundamental for spreading the application and most of traditional HCI (Human Computer Interaction) efforts are spent on this point: if a software is too difficult and requires extensive and boring training to become usable it will lose market positions with respect to similar software (even if less powerful), but with a good usability.

Requirement R2 is something novel in software, since the context is something that was not easily detectable in the past, but today, thanks to automatic location detection and a constant integration with companies organizing tools, personal calendars, social networks, etc., it is possible to take advantage of this information. For example, a user of a spreadsheet application in a secretariat may be more interested in having standard features (2d charts, simple aggregation functions, etc.) emphasized in the software interface, rather than what it is done in a marketing/data mining department of a company, in which advanced statistics, and cubic charts may be more appropriate for that context.

Requirement R3 is a core part to reward some new achievements in the usage of the software and for feeling it like a game. Reward mechanisms, especially when there is the possibility to compare one's achievements with others, can be fundamental to increase the loyalty toward a software and to stimulate the learning of additional features with the only purpose to gain some achievements.

Last, but not least, some classes of software, especially in the context-aware area, may increase their value if there is the chance for a user to collaborate (**requirement R4**). An example would be to make available additional themes for documents in a word processor application, or to provide reviews in a mobile navigation application, and so on.

In the next subsections we will show how a Software Engineer can apply the self-organizing models that are widely used in games and introduced in Section III to the development of context-aware applications. A compact conceptual map of the guidelines needed to move from *Requirements* to *Design* is depicted in Figure 2.

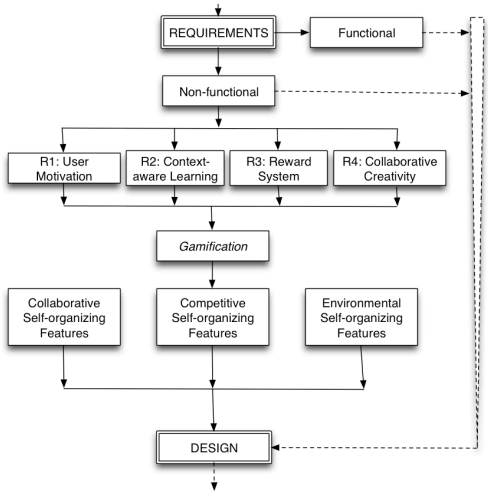


Figure 2. Gamification development process.

A. Design of Collaborative Self-organizing Features

As we have introduced in the previous section, we want to introduce features of collaborative games in generic software. This will be used to address requirements R1 and R4. To support the design phase of these features we propose the following steps.

1) Identify reusable contents of the application.
2) Associate classes of contents to the context in which the application is used.
3) Instrument the application with a way to submit and search the content.
4) Add a rating mechanism to support content selection and to have feedback on the produced content.

In the first step, the application should be decomposed at design time into classes of reusable artifacts, which may include any kind of data (for example templates, clip arts, themes, etc.) or also application logic (for example macros, automation scripts, etc.). This step is needed to decide which parts of the application or its data can be collaboratively extended.

In the second step, valid only for context-aware applications, possible application contexts must be

enumerated and associated with the classes of artifacts. This step is needed to give to the application the capability to adapt itself, and show only the artifacts that are relevant for that specific context. For example, if we consider a video-player application for a mobile device, it may decide to use a software decoder that finds the right trade-off between battery consumption, current battery level, current video quality; but the same application, when run in a context that is not mobile (i.e., the device is plugged), may decide to ignore completely the energy/quality optimization artifacts.

In the third step the user, depending on its skills, can be motivated to share possible artifacts that she has created for herself, or even to produce artifacts for the community with the purpose of gaining some sort of gratification in terms of reputation.

Finally, the fourth step is needed to give to the application the technological features needed to make collaboration activities of its users recognizable by other users. This may not be a simple feedback mechanism, but something that can be integrated in the social life of the users (for example allowing to show the contribution in the social networks of the user).

B. Design of Competitive Self-organizing Features

A possible way to contribute to the fulfillment of requirements R1 and R3 is to give to the application characteristics of competitive games. This phase of the design is more delicate than the one explained in the previous subsection because, if the level of competition is too strong, it may cause a strong loss of motivation to the users that are not able to compete with the others (further details regarding this point will be explained in Section V.B). In order to introduce a competitive game in a context-aware application we propose the following steps.

1) Identify one or more metrics that may be used to measure the level of experience of the user in an application.
2) Associate these metrics to the context in which the application is used.
3) Allow the application to share and compare the metrics of its users.

The purpose of the first step is to define a way to measure the skills of the user. This can be done in several ways that can be easily inspired from the world of video games. An example would be to mark the capability to use a particular feature of the application as an achievement (that may be converted into points).

The second step, valid for context-aware applications, would be to select or influence the metrics defined in the previous point, using context information. An example would be to give a different set of reward points when an application is used at work rather than at home; another example would be to consider the kind of device that is used to run the application, and so on.

The third step is fundamental to satisfy the competitive attitude of the application's users. Once a score target or an achievement is met, it should be important to make other users aware of that. This way other users are motivated to reach the same targets and thus contribute to a wide adoption of the application.

C. Design of Environmental Self-organizing Features

In the context of application design, environmental self-organizing features are the ones that contribute to the satisfaction or requirements R1 and R2. The idea behind this is that the application should have a way to analyze the activity of its users that is additional to the collaboration/competitions activities explained in the previous subsections. This knowledge can then be used to create a personalized behavior that is related to the particular characteristics of the user, his level of experience with the application, and the information of the context in which it is used. This can be deterministic or also non-deterministic to make it more difficult for the user to expect what it is going to happen from the "gaming" features of the application. The important fact that highlights the self-organizing nature of these features is that the "game" should be able to self-calibrate also from the past experiences of other users. An example would be to advertise possible achievements, create ways to gain more points than expected in certain situations by stimulating collaboration and competitions, and any other environmental change that can influence the way in which the application game is perceived. To achieve this we propose the following steps.

1) Identify possible stimuli that may attract the curiosity of the user to new cooperation or competition activities.
2) Identify situations in which the user may be less motivated to use the application. This may include user activity and the context (in context-aware applications).
3) Design a policy that associates stimuli to situations.

Differently from the previous two self-organizing features, all these three points may be all realized manually at design time or, with the proper training, at runtime.

In the first step the system already knows the possible ways to stimulate cooperation and competition because these aspects were already identified in the other specific self-organizing features. However, it should be given a way to select, based on previous experiences of other users, the right stimuli that may involve the user to the "hidden" application game. An example would be to offer a multiplier factor in the possibility to earn experience points as an incentive to increase user involvement.

In the second step the system should use its past and present information about the users and their context to identify problematic situations. Examples of problematic situations happen when the user has just started to use the application, when a user enters a new context, or when a user has been playing the hidden collaborative and competitive game with very poor performances.

In the third step a policy creates a runtime relationship between what has been identified in the first and in the second step. The final purpose would be to give some opportunities to the users, such as the possibility to earn additional experience points or special achievements in the first week of use of the application or when exploiting features offered in a particular usage context.

V. Case Study and Social Aspects

In this section we propose a simple case study in which we show how to apply our guidelines, some existing successful experiences that exhibit some of the properties of our approach, and finally we show some of the social implications that may arise when gamifying an application.

A. Case Study: Personal organizer

We assume we want to apply our guidelines to gamify a context-aware personal organizer. The Functional requirements of the personal organizer include the possibility to schedule tasks, keep track of previous/future tasks, modify how the tasks are advertised based on the device used, and the situation in which the personal organizer is being run (we assume to have the personal organizer accessible through a mobile device, a personal computer, and from the Internet using a web interface). We have chosen this case study because this application can be generally considered just another (boring) tool for supporting work and life activities rather than something that can be addictive and fun like a game.

Collaborative self-organization steps. 1) Reusable contents: invitations to some events that may be interesting for somebody; 2) associate the invitations above to the location in which somebody is at the moment, or is planning to go (according to the future events already scheduled in the organizer); 3) add to the user the possibility to advertise new events, specifying specific tags to identify the people that may be interested, and to identify valid contexts for these events (i.e., time and location); 4) make it possible to see how many people have put the newly added event into their schedule.

Competitive self-organization steps. 1) The level of experience can be a set of achievements that include the following: number of events per day, events scheduled for the first time, events scheduled with a particular person; 2) give a different type of achievements when an event is done in a particular time of a day, or in the weekend, or during holiday, or in a particular location, etc.; 3) let the user be notified when a friend or colleague obtains an achievement and rank the user against them to see who is the more "organized" person.

Environmental self-organization steps. 1) Suggest to schedule events that have never been scheduled (for example a dinner outside), or advertise the use of new features that have never been used in the application; 2) make it possible for the application to analyze the evolution of the context of the user (i.e., its location changes and its vicinity with other friends that are using the same application) and propose to add to the schedule social activities that have been scheduled by her friends (such as a walk in a park); 3) offer opportunities to earn extra points when the suggestions proposed from step 1 or step 2 are followed in order to increase user involvement and competitive attitude.

In the case study specified above we have shown informally how we can apply our steps for each kind of self-organizing feature. Of course the actual choices for each step are very coupled with the kind of application that needs to be gamified. More in general, the steps we propose should be done at the requirements level of a normal software engineering process, thus identifying use cases and scenarios in a more detailed and engineered way.

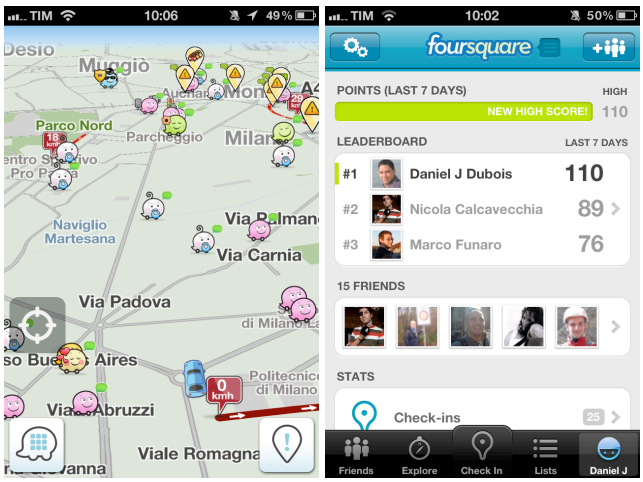
B. Existing Successful Experiences

The method we have discussed in this paper, although never proposed as a research methodology, is not completely new in the real world. We have a recent proliferation of some applications that already exhibit some of the features that we have proposed. The most recent examples are two mobile context-aware applications: *Waze* [21] and *Foursquare* [22]. Waze (see Figure 3a) is a free mobile application to support personal navigation; it contains maps collaboratively added by its users and provides updated information about traffic jams and nearby points of interests. Besides its usefulness this application managed to create its huge database using only the power of collaboration and competition among its users. Many users use Waze as a normal navigation support, but many others use it as a game, trying to earn achievements for discovering new streets, street changes, point of interests and so on. Foursquare (see Figure 3b) is another free mobile application to keep track of the visited locations of the users and their friends, and for finding new locations to visit. It uses a reward mechanism similar to Waze and its success is mostly due to the fact that most users use the application as a game. Both Waze and Foursquare are examples in which the gaming features of the application support those self-organizing phenomena that are able to motivate users to produce value for other users and for the application itself.

C. Social Aspects

We have seen that, with the use of gamification, in normal applications it is possible to create value for both the users and the application owners. However there are some critical social points that need to be evaluated before deciding to gamify an application. People who are not competitive may find the gaming features something that may leave a bad feedback to the user, especially if she fails the competition. For this reason the user should always have the freedom to disable them. As an alternative, instead of making a user in the condition of disabling them, it is easier for the application designer to find ways in which each user can feel to be the winner from some point of view, this way each one will have a positive reward in continuing

collaboration, competition, and thus in giving more value to its experience in using the application and learning new features.



(a) Waze. (b) Foursquare.

Figure 3. A screenshot of Waze and Foursquare.

VI. Conclusions and Future Work

In this paper we have proposed a set of guidelines to design context-aware user applications in such a way that non-functional requirements related to the user involvement and motivation in the use of the application have a good chance of being satisfied.

Our methodology is inspired by two worlds: self-organizing systems and existing experience in videogame design. We have seen that many features of modern (and less modern) videogames that make them addictive and long lasting can be abstracted using self-organizing models. These features include the possibility to give to their users a good sense of cooperation or competition to reach a given goal. Moreover the repeatability of a game is supported by the capability of changing overtime based on past performances of its users or other non-deterministic factors. What we have shown in this work is that the same features used to make videogames a pleasant experience can be also added to normal applications, with the double advantage of speeding-up the learning curve of the application thanks to the satisfaction of the motivational requirements, and of giving to the users that sense of fun that is typical of games even if the application itself is just (in an extreme case) a boring tool for a boring work.

What we plan to do in the future is to perform a more thorough analysis on which part of our methodology can be automatized or not, moreover some quantitative information on the productivity increase of a gamified application should be defined and reported using a real-world experiment. Last but not least, the social aspects related to the introduction of delicate psychological mechanisms such as competition in a work environment may need further attention and analyzed in an interdisciplinary way.

Acknowledgment

This research has been partially funded by the European Commission, under project SMSCom IDEAS-ERC 227977.

References

[1] S. Camazine, J. Deneubourg, N. Franks, J. Sneyd, G. Theraula, and E. Bonabeau, "Self-organization in biological systems," Princeton University Press, 2003.

[2] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," International Journal of Ad Hoc and Ubiquitous Computing, vol. 2, June 2007, pp. 263–277.

[3] E. Di Nitto, D. Dubois, and R. Mirandola, "On exploiting decentralized bio-inspired self-organization algorithms to develop real systems," SEAMS'09. ICSE Workshop, IEEE, 2009, pp. 68–75.

[4] B. Cheng, R. De Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, *et al.*, "Software engineering for self-adaptive systems: a research roadmap," Software Engineering for Self-Adaptive Systems, 2009, pp. 1–26.

[5] T. De Wolf and T. Holvoet, "Design patterns for decentralised coordination in self-organising emergent systems," Engineering Self-Organising Systems, 2007, pp. 28–49.

[6] G. D. M. Serugendo, A. Karageorgos, O. F. Rana, and F. Zambonelli, "Engineering self-organising systems, nature-inspired approaches to software engineering," LNCS, vol. 2977, Springer, 2004.

[7] S. Nicolis, C. Detrain, D. Demolin, and J. Deneubourg, "Optimality of collective choices: a stochastic approach," Bulletin of Mathematical Biology, 2003, pp. 65, 795–808.

[8] N. M. Calcavecchia, D. Ardagna, and E. Di Nitto, "The emergence of load balancing in distributed systems: the selflet approach," Runtime Models for Self-managing Syst. and Appl., ser. Autonomic Systems, Springer, 2010, pp. 97–124.

[9] S. Deterding, "Gamification: Toward a Definition," in CHI EA '11, Int. conf. on Human factors in computing systems, ACM, 2011.

[10] K. Salen and E. Zimmermann, "Rules of Play: Game Design Fundamentals," The MIT Press, 2003.

[11] C. Crawford, "Chris Crawford on Game Design," New Riders Games, 2003.

[12] G. Zichermann and C. Cunningham, "Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps," O'Reilly Media, 2011.

[13] M. Oja and J. Riekki, "Ubiquitous Framework for Creating and Evaluating Persuasive Applications and Games," Grid and Pervasive Comp. Workshops, LNCS, vol. 7096, Springer, 2012, pp. 133-140.

[14] J. McGonigal, "Reality is broken: Why games make us better and how they can change the world," The Penguin Group, 2011.

[15] O. Sotamma, "Have Fun Working with Our Product!: Critical Perspectives On Computer Game Mod Competitions," In Digital Games Research Association, 2005.

[16] D. B. Nieborg, "Am I mod or not? - An analysis of first person shooter modification culture," In Creative Gamers Seminar: Exploring Participatory Culture in Gaming, 2005.

[17] W. Scacchi, "Modding as an Open Source Approach to Extending Computer Game Systems," Open Source Systems: Grounding Research, IFIP Advances in Information and Communication Technology, vol. 365, Springer, 2011, pp. 62-74.

[18] T. L. Taylor, "Raising the Stakes: E-Sports and the Professionalization of Computer Gaming," The MIT Press, 2012.

[19] T. L. Taylor and E. Witkowski, "This is how we play it: what a mega-LAN can teach us about games," FDG '10 Proceedings of the Fifth Int. Conf. on the Found. of Digital Games, ACM, 2010, pp. 195-202.

[20] J. H. Holland, "Emergence: from chaos to order," Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1998.

[21] Waze, http://www.waze.com (last accessed 24/02/2012)

[22] Foursquare, http://www.foursquare.com (last accessed 24/02/2012)