A Software Process Engineering Approach to Improving Software Team Productivity using Socioeconomic Mechanism Design

Murat Yilmaz
Lero Graduate School in Software Engineering
Dublin City University, Ireland
e-mail: murat.yilmaz@computing.dcu.ie

Rory V. O'Connor
Lero, the Irish Software Engineering Research Center
Dublin City University, Ireland
e-mail: roconnor@computing.dcu.ie

May 26, 2011

Abstract

Software development involves teams of interconnected individuals who are encouraged to work collectively in a knowledge and communication network to produce software artifacts. At the social level, the interactions of these participants and their ability to cooperate are important for improving the productivity of teams and organizations. Software development is a knowledge and human intensive activity. It is therefore not surprising to discover that recent contributions in software development have repeatedly asserted the critical role of people in software development efforts. However, existing approaches to software development fail to fully exploit the importance of social and intellectual capital that has been highlighted in the fields of economics and sociology. Leveraging the existing approaches from economics and sociology and applying to software development can assist software organizations in maximizing their return on investment. For example, by applying one such approach, mechanism design, we can improve and model the organization's total productivity based on social aspects affecting productivity (i.e. social productivity).

Social productivity involves targeting the quality of social interactions in order to bring about productivity improvements. Furthermore, by optimizing the social structure and welfare of a software organization, we aim to improve software team collaboration, and maximize the team productivity.

1 Introduction

Software process and productivity improvement encompasses the activities which promise to increase the quality of a software product [5]. Predictably, these activities need to align process, tools and technologies with human factors and social considerations [11, 1]. Software development is a form of social activity [8]. Therefore, it is commonly conducted by teams consisting of individuals identified by characteristics of "individualism, rationality, and mutual interdependence" [12]. In this particular viewpoint, one can argue that several factors affecting the software development process should arise from the complexity of individuals' interactions and social communication costs. Therefore, the investigation of social factors and corresponding interest in social side of software development has become a part of software engineering body of research [10].

Emerged in the 1930s, *game theory* is a field of mathematics, which has been frequently applied to social sciences (especially in sociology and political sciences) for analyzing many different situations, e.g. variability of individual behaviors, and formation of coalition structures among the individuals and human networks [14]. Furthermore, the study of game theory has flourished over the last several decades and attracted many researchers. As a result, it has been applied to several diverse fields [15] including biology, linguistics, psychology, philosophy, and later in computer science [20]. There are two branches of game theory (cooperative and non-cooperative) both are useful for investigating social interactions among software teams and individuals. Non-cooperative game theory deals with situations where a limited number of players interact and their choices will affect the overall outcome, essentially participants are inclined to benefit more from these situations individually. Cooperative game theory deals with arrangement of participants in different stable combinations [4] for value maximization, which should be suitable for software business practices. Because, it explores the cooperative solution concepts, to achieve a collective outcome, cooperative game theory should be useful for assessment of several software development activities and products. For example, it can support the collaborative features of software development where coalitions are formed for profit and performance.

Software development organizations prefer to have their participants work collaboratively. Therefore, individuals need to be socially integrated so as to develop team cohesion. In a game theoretic sense, they need to correlate their activities to form teams of coalitions for better productivity. Recently, several studies have advanced our understanding of the possible use of game theory in software engineering research [2, 21]. The evidence suggests that software development organizations rely heavily on several *strategic nature* of software management activities [7, 9] (e.g. identifying stakeholders, understanding competitors and market conditions), and therefore we suggest that software management teams should benefit from what game theory offers.

2 Background

The ultimate goal of software process engineering is to provide a roadmap for the production of high quality software products that meets the needs of its stakeholders within a balanced schedule and budget [24]. It concentrates on the creation and maintenance of tasks and activity structures rather than the output or the end product. A typical software process aims to solve the potential and future problems of software development with respect to planning and budgeting constraints. In the context of our approach, *a process is considered as the coordination of structural social activities (e.g. management, production and maintenance) coupled and constrained with a set of people roles and skills (i.e. participants who perform the activities) for producing software artifacts in a predefined productivity level (see figure 1).*

In order to provide a comprehensive background to our approach, we first start by defining the concept of mechanism design and its potential impact on software practices. Secondly, we introduce the concept of software ecosystems to define a social viewpoint for software development organizations. Finally, we introduce a model to measure the social aspects that are affecting software productivity improvement.

2.1 The concept of mechanism design

A subfield of game theory, mechanism design, specifically deals with social decisions and their effects on outcomes. In this framework, a designer or a manager investigates how one designs the social structure of an organization so that the individual incentives of participants can be transformed into the organizational wide desired goals. In other words, mechanism design is an approach for thinking about the social structure of an organization. An organization can be modeled by depiction of social patterns (e.g. how interactions depends participants) and available actions for its participants. Further, we can make predictions about several organizational parameters with expected outcomes using game theoretical concepts.

Based on selectable parameters for desired goals or given objectives, we define a mechanism as *the model of an organization.* It is built on several inputs from individuals in order to produce several types of outputs. The goal is to dynamically portray an organization by designing the structure of its teams for its defined objectives and hence to motivate individuals to act in the service of an organization. We aim to establish a structural improvement inside an organization where it is based on the fact that the quality of organizational production relies on the structure of the organization [6].

In software development practices, collaboration and communication is observed in several stages of the development life cycle. Consequently, information gathering and its distribution have become somehow decentralized. Although this decentralizion may facilitate an ability for self organization and improved evolvability, it could also increase the communication costs and information based complexity.

The theory of mechanism design and its modeling implementation on software development organizations can provide a way to explore the effects of social structures on team composition, where we can use this information for better team and organizational structuring. An economic mechanism involves designing the rules for the economic activities that govern the social interactions of the participants. These rules, for example, can be designed to motivate individuals by stimulating them to behave in a certain manner, and to achieve certain economic or social outcomes. Finally, a mechanism establishes the fabric between the actions of individuals and social landscapes of software organizations. We suggest that, a mechanism enables us to maximize the economic and social outputs of the software development effort - through modeling the structure of software teams and further envisioning a software development organization.

2.2 The software ecosystem

In recent years, initial exploration of the importance of the interactions of an economic community, highlighted the fact that software development organizations should co-evolve their capabilities and roles together for maximizing the opportunities for project and business success. Therefore, the traditional viewpoint of software business; selling software to the mass market, has been replaced by the idea of interacting companies in a form similar to a biological ecosystem [17]. Based on the idea that interacting participants and organizations of the business world is considered similar to organisms of a biological ecosystem, Moore [19] from Harvard introduced the definition of a software ecosystem as an *economic coating* that forms around a software product.

Concurrent to Moore's definition, Mitleton-Kelly from the London School of Economics investigates organizational complexity by applying the theory of complex social systems to the theory of organizations [18]. She suggests that complexity arises because of the interactions through the elements of a complex co-evolving social ecosystem, including all individuals and organizations based on their business, technical and organizational relations among suppliers, customers or competitors.

Ultimately, we suggest that, a software ecosystem may be based on various information exchange networks. It could be considered as a set of several business entities working on collective outcomes in a shared market where several entities play distinctive roles (e.g. shapers, contributors). The relationship is based on the exchange of knowledge in terms of several forms, e.g. artifacts. Recognition of the software development organization as a social ecosystem brought the realization that the investigation of its social structure (e.g.

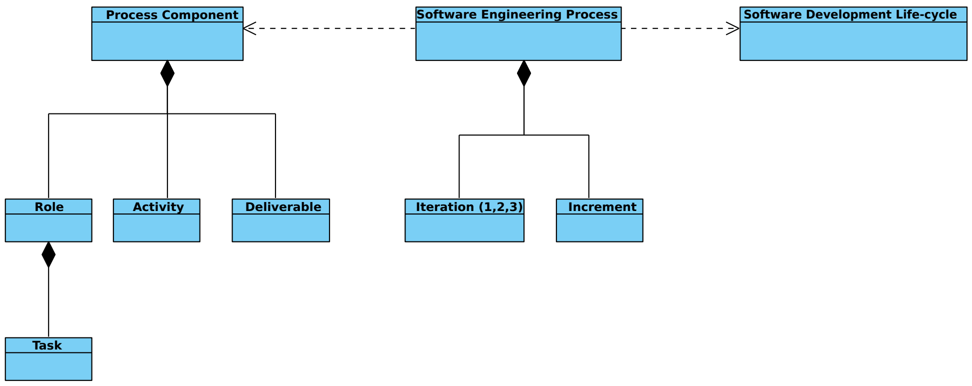

Figure 1: A meta model for software engineering process adapted from [23].

connectivity, cohesion or coupling of its members) may help to improve the human centric aspects of the business process.

3 A model of productivity improvement

Productivity improvement is one of the main concerns of a software organization. It starts very early in any development life-cycle. For example, previous research has indicated that size of a project, the development environment and the technologies (e.g. programming language) has an impact on software productivity [22].

Although a generally accepted *measurement* model of productivity is lacking [13], it can essentially be considered as the production rate or capacity of a process - something that agile software development often terms as the *project velocity*. Productivity should be considered as the value creation activities in a specific time period, and according to Boehm, the best opportunities for improving productivity in the software development effort are to be found in the attributes of people and their interactions inside the software organization [3].

Consequently, in order to improve the productivity of a software organization, crafted methods and development strategies for software development should leverage the knowledge contained in well-established people-centered approaches. Such efforts can only boost the potential for success in software development companies.

For example, a team of software practitioners is not only a good illustration of team oriented knowledge work but also a form of social (information exchange) network. A social structure can be defined in terms of social units, where they are considered as teams of interacting individuals gathered together for achieving a defined goal.

In software development activities the intellectual workers continuously collect and process the collected information (e.g. requirements, technologies) into knowledge that actualizes as software artifacts. Furthermore, the knowledge assets embedded inside the activities of a software organization are used for generating an economic value. This value should not only be determined by the outcome of the production process but (i) as the human part of the capital which encompasses the value added to the workers during the process and, (ii) as the social capital (i.e. *embedded resources in social networks* [16]), which is the capital captured by social interrelations. In order to bridge the gap between formal and the social world of software practices, any proposed valuation of a software development should not only be realized by financial form of the capital but also with its intellectual capital and especially in terms of the social capital.

In summary, software productivity is heavily dependent on social aspects of productivity which can be achieved by better *social alignment*, i.e. setting the roles of people better regarding to their personality types for maximizing productivity. In addition, it is the skills of individuals and teams which transform the acquired knowledge into software artifacts (e.g. source code, documentation, etc.) and constantly increases the competitive advantage.

3.1 Qualitative Simulation Paradigm

The notion of personality types can be considered to be socially constructed entities. Based on behavioral response patterns, personality is like an individuals' mask with which one present herself or himself to others. We suggest that the outcome of several situations are shaped by individuals personality types when they interact. These interactions, however, can be defined in terms of game theoretical forms in a way to use personality types to promote goal attainment of participants. These types are created to use on detailing behaviors of distinctive personality types and their reactions to several situations. Therefore, we term them as game theoretic personality types (GTPTs) which is a taxonomy of interactors that can be defined as a strategic situation (e.g. a situation where a decision has been made).

We propose an approach for providing a capability to induce the social environment: (i) to reveal participants personality types from a game theoretical perspective, (ii) to assess how individuals interact on software development landscape, (iii) to investigate the cause and effect relationship of social aspects over productivity, (iv) to sequentially create personality profiles of organizations based on our game-theoretic classification approach, (v) and to simulate and design team compositions based on several observed and hypothetical situations.

Depending upon the complexity of tasks and human interactions, in our setting, a precise quantitative model is hard to construct. However, based on a finite set of qualitative knowledge and relations, we should be able to analyze and simulate several team formations and situations by using several techniques such as sociometric graphs, situational context cards, qualitative simulations for examining several aspects such as communication and social structure of a software organization. Here, we formulate a new approach to overcome the complexity of understanding social and economic activities (see figure 2). We have termed this approach to be a *qualitative simulation*: a scenario based information gathering, analyzing, and evaluation method relies on blending several qualitative research techniques.

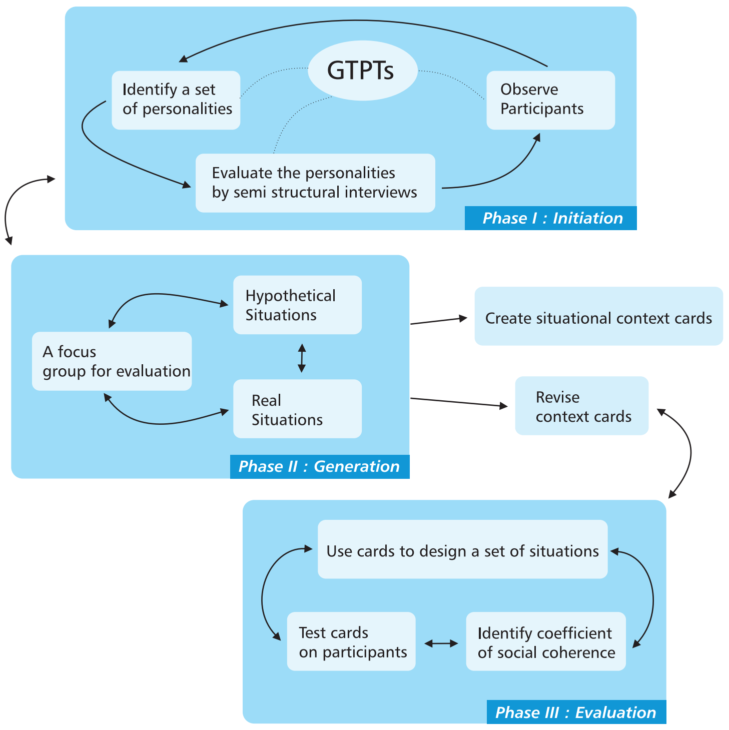
GTPTs
Identify a set of personalities
Observe Participants
Evaluate the personalities by semi structural interviews
Phase I : Initiation
Hypothetical Situations
A focus group for evaluation
Real Situations
Phase II : Generation
Create situational context cards
Revise context cards
Use cards to design a set of situations
Test cards on participants
Identify coefficient of social coherence
Phase III : Evaluation

Figure 2: The meta-model of qualitative simulation.

This method consists of three process cycles (i.e. a set of circular sequence of events designed to reinforced the outcomes): (i) initiation, (ii) generation, (iii) evaluation. The initiation cycle is used for creation of GTPTs. It starts with identification of a set of situations. Next, it continues with the observation of participants' types and we evaluate these observations by semi-structured interviews. The second process cycle is for enhancing the situational context cards creation. Several captured situation will be stored in this context, the cards however aim to help the identification of individuals and their reactions to situations. Finally, the revised context cards will be used to create several random or planned situations and they will be tested on individuals to determine their social coherence for creating optimal team compositions.

As a summary, we define qualitative simulation as a methodology based on situational context cards to define several simulation scenarios where we create hypothetical work flow for reapplying observed events to other participants. It should also include focus groups and expert reviews and case study for better construction. For example, it uses semi-structural interview techniques to validate the collected information and situational context cards (i.e. cards that store several situations), and to investigate participants reactions to several situations. Based on the data about the persons and the situations, our qualitative simulation model will be able to simulate scenarios and events that can happen during the software development life-cycle and ask several participants about their reactions to several events. Further, we will use this method to simulate and observe the changes regarding to social formation and conflicts among the structure of the software organization and share them with the participants to collect their responses.

4 Conclusions and Future Work

In this work, the primary outcome will be a modeling approach for team compositions on software development activities in terms of social interrelationships of participants, in particular by profiling the game theoretic personality types of participants. The research will extend the body of knowledge on software engineering based on the complexities several social issues such as team formations, interpersonal conflicts, social loafing that affect the group settings and structure software development teams. In addition to that, we have introduce several concepts as potential impacts and significant outcomes for software engineering research such as the notion of social productivity, and qualitative simulation which is a method formed by the combination of several qualitative techniques orchestrated to simulate several observed or hypothetical events.

Furthermore, we will develop game theoretic personality types which represents a novel kind of personality (socio) types or traits, specifically designed for team composition suitable for using our game theoretic interaction model. Furthermore, by using the personality traits, a team composer will be designed for software companies not only suitable for team composition but also for the choosing and integration process of new personnel.

In developing the concept of social productivity, a survey artifact is created, which will be used to determine the correlation among several factors of productivity and social aspects of productivity. Furthermore, by combining the notion of social network analysis and game theory concurrently, we will obtain a new viewpoint for software engineering research.

In addition to the contributions identified above, we consider our work will benefit future software engineering researcher by drawing a road map that establishes a body of knowledge, specifically on the structures and formations of software development teams and organizations. Furthermore, this project may also useful for researchers in examining the experience of applying game theoretic concepts to software development organizations, in particular the practitioners who are seeking to improve the social productivity of their organizations and to assist in the complications of social issues related to software development productivity. For example, uncontrolled fluctuations in team velocity that could cause considerable complications.

Acknowledgments

This work is supported, in part, by Science Foundation Ireland grant number 03/CE2/I303-1 to Lero, the Irish Software Engineering Research Centre (www.lero.ie).

References

[1] S. T. Acuna, N. Juristo, A. M. Moreno, and A. Mon. *A Software Process Model Handbook for Incorporating People's Capabilities*. Springer-Verlag New York, Inc., 2005.

[2] G. Bavota, R. Oliveto, A. De Lucia, G. Antoniol, and Y. Gueheneuc. Playing with refactoring: Identifying extract class opportunities through game theory. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1–5. IEEE.

[3] B. Boehm. *Software Engineering Economics*. Prentice Hall, Nov. 1981.

[4] R. Branzei, D. Dimitrov, and S. Tijs. *Models in cooperative game theory*. Springer Verlag, 2008.

[5] R. Conradi and A. Fuggetta. Improving software process improvement. *IEEE Software*, 19(4):92–99, 2002.

[6] M. Conway. How do committees invent. *Datamation*, 14(4):28–31, 1968.

[7] F. Deek, J. McHugh, and O. Eljabiri. *Strategic software engineering: an interdisciplinary approach*. CRC Press, 2005.

[8] T. DeMarco and T. Lister. *Peopleware: productive projects and teams*. Dorset House Publishing Company, Incorporated, 1999.

[9] T. Dingsoyr, T. Dyba, and N. Moe. *Agile Software Development: Current Research and Future Directions*. Springer, June 2010.

[10] Y. Dittrich, C. Floyd, and R. Klischewski. *Social thinking-software practice*. The MIT Press, 2002.

[11] R. L. Glass. *Facts and Fallacies of Software Engineering*. Addison-Wesley Professional, 2002.

[12] M. Grechanik and D. E. Perry. Analyzing software development as a noncooperative game. In *IEE Seminar Digests*, volume 29, 2004.

[13] C. Jones. *Software Engineering Best Practices: Lessons from Successful Projects in the Top Companies*. McGraw-Hill Osborne Media, 2009.

[14] A. Kuper and J. Kuper. *The social science encyclopedia*. Routledge/Thoemms Press, 1985.

[15] K. Leyton-Brown and Y. Shoham. *Essentials of game theory*. Morgan & Claypool Publishers.

[16] N. Lin. *Social capital: A theory of social structure and action*. Cambridge Univ Pr, 2002.

[17] N. Madhavji, M. Lehman, D. Perry, and J. Ramil. *Software evolution and feedback*. Wiley Online Library, 2006.

[18] E. Mitleton-Kelly. *Complex systems and evolutionary perspectives on organisations*. Emerald Group Publishing, Sept. 2003.

[19] J. Moore. *The death of competition: leadership and strategy in the age of business ecosystems*. HarperBusiness New York, 1996.

[20] N. Nisan. *Algorithmic game theory*. Cambridge Univ Pr, 2007.

[21] V. Sazawal and N. Sudan. Modeling Software Evolution with Game Theory. *Trustworthy Software Development Processes*, pages 354–365.

[22] R. Selby. *Software engineering: Barry W. Boehm's lifetime contributions to software development, management, and research*. Wiley-IEEE Computer Society Pr, 2007.

[23] B. Unhelkar. *Practical Object Oriented Analysis*. Thomson Publishing, Mar. 2005.

[24] S. Zahran. *Software Process Improvement: Practical Guidelines for Business Success*. Addison Wesley, 1998.