Position Paper accepted for the International Workshop on Distributed Software Development (DiSD'05) to be held in Paris on 29th and 30th August 2005.

COORDINATION AS THE CHALLENGE OF DISTRIBUTED SOFTWARE DEVELOPMENT

GAMEL O. WIREDU
Department of Computer Science and Information Systems, University of Limerick, Ireland

Abstract

This paper takes the position that coordination is the key challenge of the organisation of distributed software development. Based on distributed-, organisational- and software-based parameters, the paper presents an analysis of the mutual shaping between these parameters and coordination. The paper follows the analysis with tentative theoretical speculations of the potentialities of the complexities inherent in the mutual shaping; and argues for research efforts on distributed software development to be directed at the coordination challenge.

Introduction

Software development is a multifarious activity that presents extensive challenges to both researchers and practitioners concerned with it. In very broad terms, these challenges of software have been categorised into "essence and accidents" by Brooks (1987) to capture the inherent properties of software-in-development (essentials) and its attendant problems (accidents):.

> "Following Aristotle, I divide them into essence, the difficulties inherent in the nature of software, and accidents, those difficulties that today attend its production but are not inherent." (Brooks 1987).

While, both essentials and accidents of software are relevant in their own rights and in their combination, software development researchers, especially those in Organisation Science and Computer-Supported Cooperative Work (CSCW), have largely concerned themselves with developing management models and concepts aimed at tackling the accidents. To put software accidents into perspective and to capture their variegated nature, I categorise them broadly into three interdependent and interrelated facets – organisational, technological and socio-cultural. These categories seem arbitrary, but they are not: they fall under well-known dimensions of organizations, namely people, process and technology (see for example Hamilton and Kern 2001). The problem of organising therefore aims at optimising outcomes from these dimensions through several management functions (see for example Fayol 1949).

One of these attributes, coordination, emerges as a dominant concept that embodies action-related parameters such as communication, cooperation, collaboration and knowledge sharing; and inevitably relates with all other accident attributes in cause-effect fashions. This paper will argue that coordination is the pervading challenge of distributed software development. Based on the underlying premise that distribution is a significant accident, I argue, in addition, that research efforts must be directed at understanding the cause-effect complexities of coordination that are associated with distributed software development. Drawing upon the Coordination Theory of Malone and Crowston (Malone and Crowston 1990, Malone and Crowston 1994), an activity is coordinated through processes and mechanisms. Coordination processes are the actions such as managing "shared resources", "producer/consumer relationships", "simultaneity constraints" and "task/subtask dependencies." A coordination mechanism is a standardised or structured

representation of these processes: it is "a construct consisting of *a coordinative protocol* (an integrated set of procedures and conventions stipulating the articulation of interdependent distributed activities) on the one hand and on the other hand *an artefact* (a permanent symbolic construct) in which the protocol is objectified." (Schmidt and Simone 1996, p.165)(emphases in original).

Although "distribution" is specifically mentioned in Schmidt and Simone's definition, their deliberations of the concept virtually overlooked or ignored the specific problem of distribution – especially geographic or remote distribution – as a significant accident of software that is as pervading as coordination. By itself, distribution remotely separates software developers and automatically raises the threshold of software accidents and hence the coefficient of coordination efforts and the processes and mechanisms required (see Nidumolu 2001). An inevitable corollary of remote separation of developers is the emergence of location or place as a significant conditioner of developers' actions. "In a distributed activity, the modes of actors' actions necessarily correspond with the given conditions that directly derive from the peculiarities of those locations in the distribution that are hosting those actions" (Wiredu 2005).

Against this background, an intriguing question is *what are the conditions that derive from the peculiarities of those locations in a distributed software development activity*? In my opinion, they are the socio-cultural, organisational and technological accidents of software development, perceived as external factors that affect developers' actions. *How, therefore, do these accidents condition the construction and operationalisation of coordination processes and mechanisms in distributed software development?* The latter question is one that requires a comprehensive research endeavour that entails empirical studies and theoretical analysis to answer satisfactorily. However, this position paper will attempt to address it through a theoretical analysis and synthesis of software accidents, and their interrelation with their essentials in distributed software development settings. The discussions will outlay tentative, yet clear, conceptual foundations of distributed development conditions allowing for sound speculative propositions of the accidents' potentialities to condition the construction and operationalisation of coordination processes and mechanisms.

Although Brooks asserts that software engineering must direct more efforts towards the essentials, this paper directs attention towards the coordination challenge which is both manifested and operationalised through accidents. The motive is not to slam Brooks' assertion; rather, the motive is to bring the coordination problem of distributed software development to the fore. Nevertheless, it is necessary, first of all, to briefly present the key essentials with which the subsequent expositions of the distribution-related accidents will be cross-examined and analysed.

Distributed Software Development – Essentials and Accidents

General Essentials of Software Development

Software development has inherent issues that confront it and necessitates their effective handling. These issues – conceptualised as *essentials* by Brooks (1987) – directly reflect, mainly, the software product itself and the process of its development. Brooks labels the essentials as *conformity*, *changeability*, *invisibility* and *complexity* of the product and/or process. In more explicit terms, and in conformity with the coordination challenge which this paper espouses, the development process or approach holds as much significance as the outcome. Mathiassen and Stage (1992) categorise the software development into two broad, yet interwoven, areas. On the one hand, the approach, the "mode of operation" or "means of expression" adopted by developers is either "experimental" or

"analytical" depending on the situation. To them, the situation is engendered by either "uncertainty" or "complexity" of requirements information, and it influences the choice of approach to lead to the outcome, the "means of expression" – "specifications" or "prototypes."

The development approach is interesting in the analysis of the coordination challenge because the accidents directly affect the approach more than they do the outcome. Accidents impact directly on the mutual shaping between the mode of operation and means of expression because they introduce socio-cultural, organisational and technological variables that, more or less, determine the degrees of uncertainty and complexity in distributed software development.

Accidents of Distributed Software Development

The foremost distributed-related accidents to highlight are the organisational ones. In the context of this paper, the primary component is geographic distribution and the associated parameters such as distance between distributed locations and mobility. The distribution of a software development activity, the distance between locations, and the mobility of actors can individually or collectively be perceived within the dimensions of space, time and context. The spatial, temporal and contextual dimensions give more meaning to distribution, distance and mobility, and to the coordination processes and mechanisms that managers adopt to effectively organise distributed software development. Other organisational accidents are those normative tasks, processes, incentives, rules, and allocations of the resources aimed at achieving predefined goals of the development process.

In distributed software development, there are peculiar socio-cultural issues particularly related to the personnel/developers of the locations in the distribution. The degrees of socio-cultural accidents can be very minimal if the locations in the distribution are all characterised by common socio-cultural orientations or backgrounds. On the contrary, they can be very pronounced and determining if the locations do not share such similar socio-cultural characteristics. Typical examples of the significance of pronounced cultural differences are espoused by Sundeep *et. al*. (2003) in their research on globally distributed software engineering across borders. Culture is reflected in people's beliefs, perceptions, attitudes and orientations. Thus, in the organisation of distributed software development, managers have to be mindful of location-based socio-cultural accidents like the role of power and knowledge in the production and reproduction of cultural norms; belief systems that translate into context-bound meanings of information and nature of knowledge; reward systems and their process or outcome targets; modes of behaviour and outcome control; and the nature of organising in terms of markets, bureaucracies or clans (Ouchi 1979).

In the organisation of activities, managers often adopt information and communication technologies and leverage them into computational coordination mechanisms. In distributed software development, managers tackle the pervading problem of distance and distribution with ICTs to facilitate and optimise communications, cooperation and collaboration among distributed developers. ICTs are therefore technological accidents of a distributed software development that engage in a mutual interaction and shaping with organisational and socio-cultural accidents. To understand technological accidents and their impact on distributed organising (see Orlikowski 2002), it is important to examine two broad areas of computational coordination mechanisms (Schmidt and Simone 1996). On the one hand, models of structures and processes concern aspects such as data flows, conceptual schemes, knowledge management repositories, knowledge representations, and inscribed rules and methods (Hanseth and Monteiro 1997). On the other hand, models of presenting and access concern issues such as user interface, functionality, ease of use and affordance.

Inherent in these socio-cultural, organisational and technological accidents are coordination processes and mechanisms. In fact we can confidently view these accidents in totality as dependent variables, and the essentials as independent variables of distributed software development. Accidents are dependent on essential attributes such as task complexity and uncertainty (Mathiassen and Stage 1992), task interdependence and unit size (Van De Ven *et. al.* 1976, Malone and Crowston 1994), task variety and analysability, equivocality of information processing (Daft and Macintosh 1981), and the "means of expression" of the final product. As stated above, these essentials broadly relate to the software development process and product which are mutually determining.

Therefore, the mechanisms and processes that managers adopt and deploy to coordinate distributed software development reflect these essentials. For example, uncertainties in a highly interdependent distributed development may necessitate the need for computational mechanisms that facilitate communication and knowledge sharing. However, in this same instance, the knowledge sharing motive may be affected by socio-cultural belief systems that translate into context-bound meanings of information and nature of knowledge. The manager may therefore have to coordinate by instituting processes such as intermittent face-to-face meetings between developers in different locations through mobility – "travelling" and "visiting" (Kristoffersen and Ljungberg 2000). This can be interpreted as the deployment of a computational coordination mechanism to address a communication and knowledge sharing problem; and a subsequent substitution of the mechanism with a coordination process (mobility) to serve the same process. Note that while the software process and product condition coordination mechanisms and processes, the reverse effect also holds true. For example, wrong or poorly-timed coordination mechanisms and processes could further increase uncertainties instead of decreasing them. In this regard, we can also anticipate processes of construction and reconstruction of coordination mechanisms by developers in the development process. To wit, the cause-effect relationships between the parameters under consideration – process, product, uncertainty on the part of software, and mechanisms and processes on the part of coordination – are complex. The complexity translates into the need for a comprehensive study of these relationships through thorough empirical and theoretical analysis of the coordination challenge.

Position

Nevertheless, based on the brief discussions and expositions above, this paper takes the following position: *Distribution is a significant accident of the software development process that has direct implications on developers' actions. Distributed activities are profoundly different from localised activities; hence challenges of distributed software development are different from their localised equivalent. It is argued that the coordination challenge of distribution is pervading and therefore unique in its own right. In view of this, coordination models that will specifically address the peculiarity of coordination in distributed activities are critically necessary for the organisation of DSD activities. This translates into the imperative for research efforts that will identify the peculiar and unique problems of DSD activities, how these problems condition the development process and actions of developers, implications on the nature of developed software, and possible sustainable solutions to those problems.*

To conclude, this paper merely brings the coordination challenge of distributed software development to the fore by demonstrating the complexity of the problem, and by presenting snapshots of the potentialities of the relationships between the discussed parameters. It represents a mere tentative attempt to highlight the conceptual issues surrounding the coordination of distributed software development with an aim to stimulate further discussions.

References

Brooks, F. P. (1987) "No Silver Bullet: Essence and Accidents of Software Engineering," *IEEE Computer,* **20 (4)**, pp. 10-19.

Daft, R. L. and N. B. Macintosh (1981) "A Tentative Exloration of the Amount and Equivocality of Information Processing in Organizational Work Units," *Administrative Science Quarterly,* **26 (2)**, pp. 207-224.

Fayol, H. (1949) *General and Industrial Management,* Pitman, New York.

Hamilton, M. and H. Kern (2001) "Organizing for Successful Software Development," Last Accessed: 20 April 2005, Address: http://www.informit.com/articles/printerfriendly.asp?p=23953. Prentice Hall PTR.

Hanseth, O. and E. Monteiro (1997) "Inscribing Behaviour in Information Infrastructure Standards," *Accounting, Management and Information Technologies,* **7 (4)**, pp. 183-211.

Kristoffersen, S. and F. Ljungberg (2000) "Mobility: From Stationary to Mobile work," in *Planet Internet,* (Braa, K., Sørensen, C. and Dahlbom, B. ed.) Studentliteratur, Lund, 41-64.

Malone, T. W. and K. Crowston (1990) "What is Coordination Theory and How Can It Help Design Cooperative Work Systems?" in *Proceedings of the 3rd Conference on Computer-Supported Cooperative Work*ed.) ACM Press, New York.

_____ (1994) "The Interdisciplinary Study of Coordination," *ACM Computing Surveys,* **26 (1)**, pp. 87-119.

Mathiassen, L. and J. Stage (1992) "The Principle of Limited Reduction in Software Design," *Information Technology & People,* **6 (2-3)**, pp. 171-185.

Nidumolu, S. R. (2001) "The Effect of Coordination and Uncertainty on Software Project Performance: Residual Performance Risk as an Intervening Variable," *Information Systems Research,* **6 (3)**, pp. 191-219.

Orlikowski, W. J. (2002) "Knowing in Practice: Enacting a Collective Capability in Distributed Organizing," *Organization Science,* **13 (3)**, pp. 249-273.

Ouchi, W. G. (1979) "A Conceptual Framework for the Design of Organizational Control Mechanisms," *Management Science,* **25 (9)**, pp. 833-848.

Sahay, S., B. Nicholson and S. Krishna (2003) *Global IT Outsourcing: Software Development Across Borders,* Cambridge University Press, Cambridge, UK.

Schmidt, K. and C. Simone (1996) "Coordination Mechanisms: Towards a Conceptual Foundation of Computer Supported Cooperative Work Systems Design," *Computer Supported Cooperative Work: The Journal of Collaborative Computing,* **5** 155-200.

Van De Ven, A. H., A. L. Delbecq and R. Koenig (1976) "Determinants of Coordination Modes within Organizations," *American Sociological Review,* **41 (April)**, pp. 322-338.

Wiredu, G. O. (2005) "Mobile Computing in Work-Integrated Learning: Problems of Remotely-Distributed Activities and Technology Use," Doctoral Dissertation, University of London, London.