Software Modularization in Global Software Development

Dilani Wickramaarachchi
Department of Computer Science & Computer Engineering
La Trobe University
Melbourne, Australia
dwickramaarachchi@students.latrobe.edu.au

Richard Lai
Department of Computer Science & Computer Engineering
La Trobe University
Melbourne, Australia.
lai@cs.latrobe.edu.au

***Abstract*—In a Global Software Development (GSD) environment, multiple teams are located in geographically dispersed locations and project management is becoming a difficult task; and the work distribution process has become an important function in GSD. Work distribution process can influence both the benefits of global software development (e.g. cost reduction, availability of people, proximity to the customer) and its risks (e.g. communication and coordination overhead). Therefore, work should be distributed after a thorough study on the different tasks and the various locations. To do this, the categorization of work should be undertaken before it is distributed. This is known as modularization which can help overcome communication and coordination problems in the development process and makes the process more stable. In this paper, we present a survey on software modularization which includes principles of modularization and applicability of the concept to GSD.**

Keywords— Global Software Development, Modularization, Dependency

I. INTRODUCTION

Global Software Development (GSD) can be defined as "any aspect of software engineering that involves the combined efforts of software professionals in different locations separated by significant distances" [1]. In spite of the growing attention to GSD, there is a limited understanding of how to make GSD a success and why it fails [2]. Several studies illustrate that half of the organizations which shift processes offshore fail to generate expected financial benefits [3] and some of them fail completely [4]. GSD projects and case studies range from announcements of remarkable victory to total collapse [5]. The cost-benefits trade-off in GSD is still not fully understood [6]. No research so far has imparted a clear vision of the true amount of investment necessary to make GSD projects a success [5]. Despite these threats, GSD has become increasingly popular [7]. Therefore, the challenge is to minimize the threats in order to increase the likelihood of high profits [8].

Very often, large software projects are developed by multiple teams to meet deadlines and deliver the product on time [9]. In a GSD environment, multiple teams are located in geographically dispersed locations and project management is becoming a difficult task [10]. In particular, the work distribution process has become an important function in GSD [8]. The work distribution process can influence both the benefits of global software development (e.g. cost reduction, availability of people, proximity to the customer) and its risks (e.g. communication and coordination overhead) [11]. Therefore, work should be distributed after a thorough study on the different tasks and the various locations. To do this, the categorization of work should be undertaken before it is distributed [12] . This is known as modularization.

The software development process is difficult and modularization can make it more complicated [9]. However, modularization is important as it helps to overcome communication and coordination problems in the development process and makes the process more stable. The notion of modularity is central in the design and production of software artifacts, especially for large and complex projects [13]. Software development work can be distributed by development phases or by complete modules or subsystems allocated to different development sites The latter can be much more advantageous than the former [9] which highlights the importance of modularization in practice. Moreover, properly modularized software is easy to maintain and is of great assistance to the developer [9]. Therefore, one of the most challenging product-related issues is to find the right modularization to support communication between distributed teams [14].

In this paper, we present a survey on software modularization which includes principles of modularization and applicability of the concept to GSD.

II. LITERATURE REVIEW APPROACH

We have tracked down the evidence by which to achieve our aim cited above by surveying past research work extracted from a search process. In order to carry out a comprehensive analysis, search strings were established by combining the keywords through the logical connectors "AND" and "OR". The studies were mainly obtained from the following search sources: IEEE Xplore, ACM Digital Library, Science Direct, Springer Link, Wiley Interscience

978-1-4799-7996-7/14/$31.00 ©2014 IEEE

and Google Scholar. The quality of these sources guarantees the quality of the study. The literature review was done using the keywords "modularization", and "software dependency" with "software development/GSD".

III. MODULARIZATION IN SOFTWARE DEVELOPMENT

Modularization is a traditional approach used to cope with dependencies in product development [15]. This concept has caught the attention of engineers, management researchers and corporate strategists in a number of industries [16]. Prior research has shown that the identification and management of work dependencies is a major challenge in the coordination of software development organizations [17-19]. GSD makes it more challenging and complicated and the identification and management of work dependencies is even more important in GSD than collocated software development. Therefore, it is important to investigate the applicability of modularization to minimise the challenges of coordination in GSD.

A. Main Ideas

The concept of modularization in the software industry has a long history and it has been used to improve the process of software development at different stages to different levels. The main concepts behind software modularization were introduced by pioneer researchers forty years ago with a remarkable contribution being made by Melvin Conway and David Parnas.

Conway's Law: In 1968, Melvin Conway identified the relationship between software architecture and development organizations which is now known as "Conway's Law". Conway's Law states that the interface structure of a software system reflects the social structure of the organization [20]. Therefore, Conway is consistently cited as the first to identify this sociological aspect [21, 22] which postulates that the structure of a software system reflects the communication needs of the people performing the work [17]. Researchers have argued that, as a consequence of Conway's Law, software should be modularized [23-26].

Parnas' Concept of Information Hiding: In software engineering, Parnas [27] was the first to articulate the idea of modular software design, introducing the concept of information hiding in 1972 where, software modules should only share with other modules those details that are not likely to change in order to reduce the degree of dependency between modules. A particular module's details should be encapsulated within the module and should not affect other modules so that they can be changed more easily without having a significant impact. Accordingly, a distinction between a module's interface and implementation is proposed as follows: the interface is the stable part, whereas the implementation is the unstable part. Modules are connected to each other through their interfaces.

Moreover, the information hiding concept recommends that by reducing dependencies at the artifact level, it is possible to reduce developers' dependencies on one another, creating a managerial advantage [27, 28]. Currently, this is an accepted argument among researchers and practitioners. The textbook written by Ghezzi et al. [29] describes this as follows: if a design is composed of highly independent modules, it supports the requirements of large programs: independent modules form the basis of work assignments to individual team members. The more independent the modules, the more independently the team members can proceed in their work [29].

B. Techniques and Principles to deal with Software Dependencies

The need to deal with dependencies between software components has been recognised, hence there are several techniques, tools and principles to deal with these dependencies.

Programs [30] [31], components [32] and software architectures [33] have been developed to deal with the different abstractions used in the construction of software systems. These approaches are implemented in software development tools and are used to identify software dependencies which improve reusability and maintainability.

Interfaces, data encapsulation, and polymorphism [34] are mechanisms used in programming languages to reduce dependencies between software elements which are based on the principle of information hiding [27].

C. Advantages of Modularization

From an engineering perspective, a modularization generally has several advantages such as: to make complexity manageable; to enable parallel work; and to accommodate future uncertainty [35] [16]. The ultimate purpose of software modularization is to increase software productivity and quality [36].

D. Importance of Modularization for Communication and Coordination

The literature advocates the importance of modularization to reduce communication and coordination problems in software development. Some of these are shown in Table 1.

TABLE 1: IMPORTANCE OF MODULARIZATION IN SOFTWARE DEVELOPMENT

Statements	Reference
"A traditional approach to reduce the need to communicate and coordinate in development organizations is modularization or partitioning the system to be developed into nearly independent parts"	[37]
"The modularity of a product can be expected to be mirrored by the modularity of the organizational structure"	[38]
"Modularization is the approach typically used to minimize technical dependencies among the parts of a system "	[39]
"Dependency of system units is reduced through the efforts of system modularization"	[40]
"An ideal solution to the problem of organizing a large software effort is to decompose the system into highly cohesive and loosely coupled modules"	[41]
Modularity is "thought to reduce the complexity of software development" [22] and shorten development time [25]	[22, 25]
Modularity also insulates teams from changes made in other parts of the system	[41]
"Without modularity, there is little hope that contributors can understand enough of a design to contribute to it, or develop new features and fix defects without affecting other parts of the system.	[42]
Interrelated subsystems "should be developed by persons that have easy and frequent communication with each other" [43] because "developers dealing with dependent components are more likely to engage in communication than developers implementing independent components"[44]	[43, 44]

It can be seen that all these comments are based on practicality and are in line with either the principles of Conway's law or Parnas' concept of information hiding.

E. Measures of the Degree of dependency

Cohesion and coupling [45-47] are the theoretical measures used to indicate the degree of dependency where cohesion indicates within-module dependency and coupling indicates between-modules dependency.

Cohesion: Cohesion is a measure of the relationships between its elements. It measures the degree of dependencies within a module. Cohesion is a property of a software module or program that represents the functional relatedness of their internal elements [45] which is measured in terms of the strength of the binding elements within the module.

Coupling: Coupling measures the strength of the dependency relationships between different modules [29, 45]. If two modules depend on each other heavily, they have high coupling; otherwise, the modules have low coupling and are independent of each other.

Cohesion and coupling: The concepts of cohesion and coupling are interrelated: cohesion refers to the degree of interconnection among the parts of a single module, and coupling refers to the strength of the interconnections between different modules. Both concepts help to establish the quality of a particular design [48].

Researchers have defined coupling and cohesion as properties of decomposition units and generally accepted rules state that modules should have high cohesion and low coupling [49]. Therefore, measuring the degree of modularization of a software design can be achieved through the two quality measures of cohesion and coupling.

F. Modularization Approach

According to Kenneth Eskildsen [50], modularization can be expressed and initiated with four main focuses: (1) component bottom-up, (2) component top-down, (3) function bottom-up; and (4) function top-down which are represented in Figure 1. Each approach has different opportunities and benefits associated with the focus.

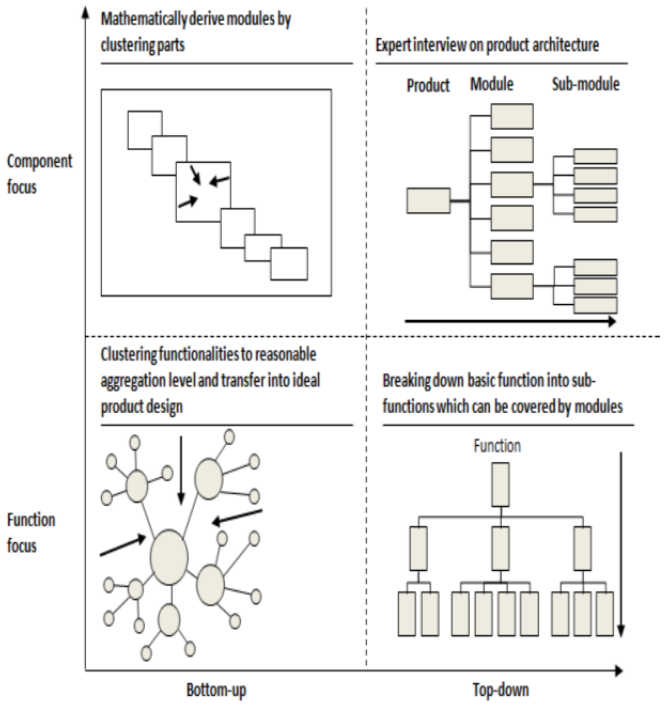

Figure 1: Modularization approach. [50]

IV. MODULARIZATION IN GSD

A. Collocation versus Globalization

There are certain aspects of any software project which require frequent and informal communication, making the collocation of team members important [51]. Evidence in the literature supports this. For example, Conch'uir et al. [52] find in their study that the collocation of team members is needed to develop certain units of functionality. Further, several researchers [23, 51] observed the difficulties, developers have to face to identify relevant parties to contact to ensure their work is done when they are distributed.

However in GSD, not all developers are collocated and the collocation of work is not feasible all the time. Therefore, work should be assigned according to the greatest possible architectural separation in a design [24]. Further, minimizing coordination needs is crucial in GSD since tasks take longer due to cultural differences, time zone differences and engineers need to spend more time coordinating their work across the globe [53]. Oberortner and Damian identify that challenges such as miscommunication, misunderstandings, and how to share information are more severe when teams are not collocated [54]. It has been found that coordination time can be 2.5 times greater in a distributed environment than when teams are collocated [19] due to these problems.

Therefore, it is important that tightly coupled work items that require frequent coordination and synchronization should be collocated [51, 55]. Modularization can be used to identify tightly coupled work. In the GSD literature, researchers argue that the more cleanly separated the modules, the more likely the organization can successfully develop them at different sites because this will remove the communication required among the different sites. To support this, the module should be loosely coupled.

B. Existing Work- Limits of Modularity

Researchers have identified that existing modularization principles intended to reduce dependencies among components of a system have some limitations in the context of GSD. The following are the main observations found in the literature in this direction.

Traditional assumptions on the basic principle of modularization

The concepts introduced by Parnas [27] and Conway [20] assume technical modularization represents task/work modularization. Accordingly, by reducing the technical dependencies among modules, task dependencies are reduced, therefore reducing the need for communication among work groups. According to several empirical findings in the context of GSD, there are certain limitations to these assumptions, outlined as follows.

Empirical evidence detailed by Cataldo et al. [56] indicates that the relationship between product/technical structure and task structure is not as simple as previously assumed. Existing software modularization approaches only use a subset of the technical dependencies, typically syntactic relationships, of a software system [57], therefore, other potentially relevant work dependencies might be ignored. In their later study [58], Cataldo et al. examine how different types of technical dependencies (syntactic, semantic, logical) relate to work dependency through change requests to source code developed in a distributed environment. The empirical evaluation shows that when developers' coordination patterns are congruent with their coordination needs, the resolution time of modification requests is significantly reduced. Furthermore, the analysis highlights the importance of identifying the "right" set of technical dependencies that drive the coordination requirements among software developers. The analyses indicates that traditional software dependencies, such as syntactic relationships, tend to capture a relatively narrow view of product dependencies that is not fully representative of the important product dependencies that drive the need to coordinate. Further, the study indicates the importance of logical dependency for a more accurate representation of the product dependencies.

Parnas' Information hiding principle

Modularization which suggests minimal communication might result in an adverse impact on GSD. The following are the main observations found in the literature in this direction.

Souza et al. [59] find that information hiding resulted in development teams being unaware of the others teams' work, resulting in coordination problems. Grinter et al. [18] report similar findings for GSD which highlights that the main consequence of reducing team communication increases costs because problems tend to be discovered later in the development process. In relation to the stability of interfaces, Souza et al. [17] find in a field study of a large software project that interfaces tend to change often and their design details tend to be incomplete, which can lead to serious integration problems.

Software modules differ in complexity and little is known about the impact of instability on coordination among distributed development teams. However, recent research has started to examine these issues. Cataldo et al. [56] present case studies where even simple interfaces between modules developed by remote teams create coordination breakdown and integration problems. The study reports that semantic dependencies are often problematic and they argue that the developers' ability to identify and manage dependencies is hindered by several inter-related factors, such as development processes, organizational attributes (e.g. structure, management style) and uncertainty of the interfaces.

V. CONCLUSIONS

As this discussion suggests, modularization is important in dependency identification and reduces communication difficulties, but the literature on the successful application of GSD modularization to avoid communication difficulties is limited.

In the software engineering literature, researchers jointly accept that the identification of dependency is important in software development and modularization is a technique to deal with dependencies. The ultimate purpose of software modularization is to increase software productivity and quality, and improve software maintenance. Therefore, the concept of modularization has already been embedded in programming languages and software tools.

It has been found that the main advantages of the concept of modularization are in line with the requirements of GSD but the empirical studies on GSD have found some limitations in traditional concepts of modularization. One main argument centres on the relationship between technical dependency and work dependency, as some researchers have found existing modularization uses only a subset of technical dependency which is not sufficient to represent work dependency. Moreover, some have suggested that modularization which suggests minimal communication can lead to an adverse impact on GSD.

Overall, the above review of the literature related to modularization leads to the identification of the following knowledge gaps.

Limitations of existing modularization concepts: Although GSD empirical studies on traditional modularization highlight several limitations, they do not suggest that it is not useful. Therefore, further research is required to identify the limits of modularity and supplement the existing modularization concept with a coordination mechanism to allow developers to deal with the assumptions that are not reflected in the specifications.

Early application of work modularization: It was found that the concept of modularization can be used to reduce communication and coordination requirements between distributed groups. Additional costs in GSD are mainly based on communication and coordination activities between locations. If proper categorization can be ensured, taking into consideration the interaction requirements and if the dependent work is grouped together before task distribution, additional overheads can be minimised. However, there is limited work in the literature related to the modularization of work before task distribution.

REFERENCES

[1] P. Taylor, D. Greer, P. Sage, G. Coleman, K. McDaid, and F. Keenan, "Do agile GSD experience reports help the practitioner?," 2006, p. 93.

[2] W. DeLone, J. Espinosa, G. Lee, and E. Carmel, "Bridging global boundaries for IS project success," *Proceedings of the 38th Hawaii International Conference on System Sciences,* 2005.

[3] R. Aron and J. Singh, "Getting offshoring right," *Harvard business review,* vol. 83, p. 135, 2005.

[4] N. Huda, N. Nahar, J. Tepandi, and P. S. Deo, "Key barriers for global software product development organizations," in *Management of Engineering & Technology, 2009. PICMET 2009. Portland International Conference on*, 2009, pp. 1081-1087.

[5] D. Šmite and J. Borzovs, "Managing Uncertainty in Globally Distributed Software Development Projects," *Datorzin tne un inform cijas tehnolo ijas,* p. 9, 2008.

[6] P. Ågerfalk, B. Fitzgerald, H. Holmström Olsson, and E. Ó Conchúir, "Benefits of global software development: The known and unknown," *Making Globally Distributed Software Development a Success Story,* pp. 1-9, 2008.

[7] D. Damian and D. Moitra, "Guest Editors' Introduction: Global Software Development: How Far Have We Come?," *Software, IEEE,* vol. 23, pp. 17-19, 2006.

[8] D. Wickramaarachchi and R. Lai, "A Method for Work Distribution in Global Software Development," in *Advance Computing Conference (IACC), 2013 IEEE 3rd International*, 2013, pp. 1443-1448.

[9] A. Khare and M. P. S. Bangare, "MEASURING THE QUALITY OF OBJECT ORIENTED SOFTWARE MODULARIZATION," *International Journal on Computer Science and Engineering (IJCSE),* vol. 3, 2011.

[10] R. Lai and N. Ali, "A Requirements Management Method for Global Software Development," *Advances in Information Sciences,* Human and Science Publication, USA, Volume 1, Number 1, pp. pp38-58., 2013.

[11] A. Lamersdorf, J. Munch, and D. Rombach, "A survey on the state of the practice in distributed software development: criteria for task allocation," in *Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on*, 2009, pp. 41-50.

[12] D. Wickramaarachchi and R. Lai, "A Method for Modularization in Unified Modeling Language Specification for Global Software Development," *Model Based Software Engineering: Some Perspectives, Infosys Labs Briefings,*. Vol 11 No 2, 2013.

[13] A. Narduzzo and A. Rossi, "Modular design and the development of complex artifacts: Lessons from free/open source software," *Quaderno, DISA,* vol. 78, 2003.

[14] F. Salger, "Software architecture evaluation in global software development projects," in *On the Move to Meaningful Internet Systems: OTM 2009 Workshops*, 2009, pp. 391-400.

[15] M. Cataldo, A. Mockus, J. A. Roberts, and J. D. Herbsleb, "Software dependencies, work dependencies, and their impact on failures," *Software Engineering, IEEE Transactions on,* vol. 35, pp. 864-878, 2009.

[16] C. Y. Baldwin and K. B. Clark, *Modularity in the design of complex engineering systems*: Springer, 2006.

[17] C. R. B. De Souza, "On the relationship between software dependencies and coordination: field studies and tool support," UNIVERSITY OF CALIFORNIA, 2005.

[18] R. E. Grinter, J. D. Herbsleb, and D. E. Perry, "The geography of coordination: dealing with distance in R&D work," in *Proceedings of the international ACM SIGGROUP conference on Supporting group work*, 1999, pp. 306-315.

[19] J. D. Herbsleb and A. Mockus, "An empirical study of speed and communication in globally distributed software development," *Software Engineering, IEEE Transactions on,* vol. 29, pp. 481-494, 2003.

[20] K. H. Britton and D. L. Parnas, *A-7E software module guide*: Naval Research Laboratory, 1981.

[21] I. Kwan, A. Schroter, and D. Damian, "Does socio-technical congruence have an effect on software build success? a study of coordination in a software project," *Software Engineering, IEEE Transactions on,* vol. 37, pp. 307-324, 2011.

[22] F. Bolici, J. Howison, and K. Crowston, "Coordination without discussion? Socio-technical congruence and Stigmergy in Free and Open Source Software projects," in *Socio-Technical*

Congruence Workshop in conj Intl Conf on Software Engineering, Vancouver, Canada, 2009.

[23] C. R. De Souza and D. F. Redmiles, "The awareness network, to whom should I display my actions? and, whose actions should I monitor?," *Software Engineering, IEEE Transactions on,* vol. 37, pp. 325-340, 2011.

[24] B. Lings, B. Lundell, P. J. Ågerfalk, and B. Fitzgerald, "Ten strategies for successful distributed development," in *The Transfer and Diffusion of Information Technology for Organizational Resilience*, ed: Springer, 2006, pp. 119-137.

[25] P. Carlson and N. Xiao, "Experience and recommendations for distributed software development," in *Collaborative Teaching of Globally Distributed Software Development Workshop (CTGDSD), 2012*, 2012, pp. 21-24.

[26] F. Beck and S. Diehl, "On the congruence of modularity and code coupling," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011, pp. 354-364.

[27] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Communications of the ACM,* vol. 15, pp. 1053-1058, 1972.

[28] J. D. Herbsleb and R. E. Grinter, "Architectures, coordination, and distance: Conway's law and beyond," *Software, IEEE,* vol. 16, pp. 63-70, 1999.

[29] C. Ghezzi, M. Jazayeri, and D. Mandrioli, *Fundamentals of software engineering*: Prentice Hall PTR, 2002.

[30] J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The program dependence graph and its use in optimization," *ACM Transactions on Programming Languages and Systems (TOPLAS),* vol. 9, pp. 319-349, 1987.

[31] A. Podgurski and L. Clarke, *The implications of program dependencies for software testing, debugging, and maintenance* vol. 14: ACM, 1989.

[32] M. Vieira and D. Richardson, "The role of dependencies in component-based systems evolution," in *Proceedings of the International Workshop on Principles of Software Evolution*, 2002, pp. 62-65.

[33] J. A. Stafford and A. L. Wolf, "Architecture-level dependence analysis for software systems," *International Journal of Software Engineering and Knowledge Engineering,* vol. 11, pp. 431-451, 2001.

[34] C. Larman, "Protected variation: The importance of being closed," *Software, IEEE,* vol. 18, pp. 89-91, 2001.

[35] J. Munnelly, S. Fritsch, and S. Clarke, "An aspect-oriented approach to the modularisation of context," in *Pervasive Computing and Communications, 2007. PerCom'07. Fifth Annual IEEE International Conference on*, 2007, pp. 114-124.

[36] Y. Cai, "Assessing the Effectiveness of Software Modularization Techniques through the Dynamics of Software Evolution," *Contemporary Modularization Techniques (ACoM. 08),* p. 40, 2009.

[37] M. Cataldo and J. D. Herbsleb, "Communication patterns in geographically distributed software development and engineers' contributions to the development effort," in *Proceedings of the 2008 international workshop on Cooperative and human aspects of software engineering*, 2008, pp. 25-28.

[38] T. Kude and J. Dibbern, "Tight versus loose organizational coupling within inter-firm networks in the enterprise software industry–the perspective of complementors," 2009.

[39] M. Cataldo and J. D. Herbsleb, "Communication networks in geographically distributed software development," 2008, pp. 579-588.

[40] D. L. Parnas, "Why software jewels are rare," *Computer,* vol. 29, pp. 57-60, 1996.

[41] C. Bird, T. Zimmermann, and A. Teterev, "A theory of branches as goals and virtual teams," in *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*, 2011, pp. 53-56.

[42] A. D. MacCormack, J. Rusnak, and C. Y. Baldwin, *Exploring the duality between product and organizational architectures: A test of the mirroring hypothesis*: Harvard Business School, 2008.

[43] H. Hadaytullah, O. Raiha, and K. Koskimies, "Genetic approach to software architecture synthesis with work allocation scheme," in *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*, 2010, pp. 70-79.

[44] B. De Souza, "Exploring the relationship between dependencies and coordination to support global software development projects," in *2006 IEEE International Conference on Global Software Engineering (ICGSE'06)*, 2006.

[45] W. P. Stevens, G. J. Myers, and L. L. Constantine, "Structured design," *IBM Systems Journal,* vol. 13, pp. 115-139, 1974.

[46] C. Larman, "Applying UML and Patters: An introduction to Object-oriented analysis and design and the Unified Process," ed: Prentice Hall, ISBN, 2002.

[47] T. N. Al-Otaiby, M. AlSherif, and W. P. Bond, "Toward software requirements modularization using hierarchical clustering techniques," in *Proceedings of the 43rd annual Southeast regional conference-Volume 2*, 2005, pp. 223-228.

[48] I. Sommerville, "Software Engineering. Harlow, England," ed: Addison-Wesley, 2001.

[49] L. Yu and S. Ramaswamy, "Verifying design modularity, hierarchy, and interaction locality using data clustering techniques," in *Proceedings of the 45th annual southeast regional conference*, 2007, pp. 419-424.

[50] K. Eskildsen, "Modularization," *International Technology Management, Aalborg Universtity – Copenhagen,* 2011.

[51] F. Détienne, "Collaborative design: Managing task interdependencies and multiple perspectives," *Interacting with computers,* vol. 18, pp. 1-20, 2006.

[52] E. O. Conchuir, H. Holmstrom, J. Agerfalk, and B. Fitzgerald, "Exploring the assumed benefits of global software development," in *International conference on Global Software Engineering (ICGSE)*, 2006, pp. 159-168.

[53] J. Bosch and P. Bosch-Sijtsema, "From integration to composition: On the impact of software product lines, global development and ecosystems," *Journal of systems and software,* vol. 83, pp. 67-76, 2010.

[54] E. Oberortner and D. Damian, "Towards patterns to enhance the communication in distributed software development environments," in *18th Conference on Pattern Languages of Programs (PLOP), Portland, OR, USA*, 2011.

[55] A. Mockus and D. M. Weiss, "Globalization by chunking: a quantitative approach," *Software, IEEE,* vol. 18, pp. 30-37, 2001.

[56] M. Cataldo, M. Bass, J. D. Herbsleb, and L. Bass, "On Coordination Mechanisms in Global Software Development," *ICGSE,* vol. 7, pp. 71-80, 2007.

[57] A. Garcia, P. Greenwood, G. Heineman, R. Walker, Y. Cai, H. Y. Yang*, et al.*, "Assessment of Contemporary Modularization Techniques-ACoM'07: workshop report," *ACM SIGSOFT Software Engineering Notes,* vol. 32, pp. 31-37, 2007.

[58] M. Cataldo, M. Bass, J. D. Herbsleb, and L. Bass, "On coordination mechanisms in global software development," in *Global Software Engineering, 2007. ICGSE 2007. Second IEEE International Conference on*, 2007, pp. 71-80.

[59] C. R. De Souza, D. Redmiles, L.-T. Cheng, D. Millen, and J. Patterson, "How a good software practice thwarts collaboration: the multiple roles of APIs in software development," in *ACM SIGSOFT Software Engineering Notes*, 2004, pp. 221-230.