Dynamic Networked Organizations for Software Engineering

Damian A. Tamburri
VU University Amsterdam
De Boelelaan 1081a
Amsterdam, The Netherlands
d.a.tamburri@vu.nl

Remco de Boer
ArchiXL
Nijverheidsweg Noord 60-27
3812 PM, Amersfoort
rdeboer@archixl.nl

Elisabetta di Nitto
Politecnico di Milano
Piazza Leonardo da Vinci 32
20133, Milano
dinitto@elet.polimi.it

Patricia Lago
VU University Amsterdam
De Boelelaan 1081a
Amsterdam, The Netherlands
p.lago@vu.nl

Hans van Vliet
VU University Amsterdam
De Boelelaan 1081a
Amsterdam, The Netherlands
hans@cs.vu.nl

ABSTRACT

Current practice in software engineering suggests a radical change in perspective: where once stood fixed teams of people following a development plan, now stand just-in-time Dynamic Networked Organizations (DyNOs), adopting a common flexible strategy for development, rather than a plan. This shift in perspective has gone relatively unnoticed by current software engineering research. This paper offers a glimpse at what processes and instruments lie beyond "current" software engineering research, where studying emergent DyNOs, their creation and steering becomes critical. To understand the underpinnings of this evolution, we explored a simple yet vivid scenario from real-life industrial practice. Using scenario analysis we elicited a number of social and organizational requirements in working with DyNOs. Also, comparing our evidence with literature, we made some key observations. First, managing DyNOs makes organizational requirements a first-class entity for development success. Second, research in software engineering should be invested in understanding and governing the DyNOs behind the software lifecycle.

Categories and Subject Descriptors

K.7.2 [**Organizations**]: Miscellaneous; D.2.9 [**Software Engineering**]: Management—*Programming teams, Software quality assurance, Software process models, Life cycle*

General Terms

Theory, Management, social Factors

SSE '13, August 18, 2013, Saint Petersburg, Russia

Keywords

Global Software Engineering, Networked Organizations, Social Software Engineering

1. INTRODUCTION AND MOTIVATION

The state of practice suggests that software engineering is now in the hands of many organisations, producing software in a networked fashion. Outsourcing [17], Open-Source Software Foundations [4] and Global Software Engineering (GSE) [9] are examples of how organizations constantly network with others to produce software. Moreover, the increase of scale and dynamicity of markets force these networked organizations to emerge and adapt extremely rapidly to organisational changes during the software lifecycle (e.g. employee or organizations turnover). The result is a dynamic networked organization (DyNO) reacting extremely rapidly to changes (in the internal structure or surrounding context). Organizations in the DyNO cooperate following a development strategy (i.e. a high-level plan in conditions of uncertainty), rather than sharing a plan (i.e. a fixed set of actions in pursuit of a certain goal). For example, we witnessed first-hand many small and medium enterprises (SMEs) team-up very dynamically and extremely rapidly, to share efforts during GSE, strategically agreeing development increments every step of the way. What results is a DyNO [25] of SMEs, able to adapt quickly (e.g. by including additional partners) as more requirements fluidly emerge from the development context.

The implications behind this dynamism are vast and need to be studied if future software engineering is to be successful. First, new coordination and steering needs are beyond the common principles and practices, e.g. in GSE [7, 24]. For example, coordination would require a shared vision and agreement on software architectures, while corporate policies would otherwise protect them as secrets from more open collaborators (e.g. open-source communities). Also, the social and socio-technical implications [2] in the scenario are far from clear. For example, on-the-fly collaboration could leave open "skill holes" when terminated. this could lead to maintenance problems in the future (since the skills are not in the community anymore).

In this paper we explore a real-life industrial scenario in

which a DyNO was crafted and continuously evolved during software engineering. From the scenario we elicited social and organizational requirements. Then, comparing with reference models (e.g. 3C model [22]) from current software engineering research and practice, we made three key observations. First, managing DyNOs requires explicit engineering of organizational requirements [20], i.e. the set of social and organizational needs that shape the organizational structure [25] behind the software lifecycle. These requirements can only be satisfied through continuous evolution [11] of the networked organization [26, 27].
Second, current software engineering research is focused on understanding software lifecycle problems [19, 6], rather disregarding the governance of the DyNOs lurking behind.
Third, two processes are critical for managing DyNOs: creation and adaptation for continuous evolution. Both processes introduce many unexplored challenges and factors for GSE and software engineering in general [18].
We conclude that software engineering is on the brink of an evolution. The evolution features just-in-time DyNOs for software development, e.g. using SMEs as part of a "virtual enterprise" [15]. Much research is needed to understand and support the social and organizational underpinnings of this evolution.

2. PREVIOUS AND RELATED WORK

We carried out work in the understanding of dynamic communities of developers working in GSE [24, 23, 26]. In previous work we explored the state of the art in social networks and organizations literature [25]. These works were a necessary step to understanding communities working on global software [26]. Also, many studies in GSE revolve around social community aspects. For example, distance limits informal communication within global communities [1], which in turn impedes the building of trust in virtual teams [5]. This limits the degree to which implicit knowledge is shared among teams, and interferes with the ability to solve process issues [8, 13]. Temporal distance also results in delays, e.g. in responding to inquiries or delivery. This can lead to incorrect assumptions and again, mistrust among teams due to perception of lack of commitment [10]. Language and cultural distance can cause technical misunderstandings of goals, tasks and requirements, and inhibit developers' awareness [6] and the formation of trust.
Moreover, there are several frameworks that we used as a reference to interpret what we found in our scenario. For example, we compared our results with organizational styles from open-source communities [4, 21]. Also, we considered concepts from Ultra Large Scale (ULS) systems theory [16], e.g. [12]. In [12] the authors offered a lens to understand many of the complexities emerging in our scenario. In addition, distributed and multi-site governance models [14] uncovered the implications for governance in a distributed setting. We used these models to interpret the novelties of our dynamic networked organization scenario. Finally, collaboration reference models, e.g. the 3C model from [22], offers a valuable map to understand the variables and requirements in distributed collaboration and communication.

3. CASE STUDY

This section outlines our case scenario and offers details on the DyNO involved. Then, this section introduces the organizational and social challenges involved in the scenario (e.g. the higher requirements for trust and visibility of tasks and tasks dependencies). Finally, the section introduces the organizational network evolution that was triggered by the outlined challenges (e.g. the adoption of a negotiated business process).

We investigated an organizational scenario from one of our industrial partners, an SME by the name of ArchiXL[1]. To gather needed data, we used a focus group involving two project managers responsible for the project at hand. Through an open, semi-structured discussion we gathered evidence describing ArchiXL's development scenario featuring DyNOs. Second, we analysed results through factor and scenario analysis. Finally, we operated systematic mappings between our data and current literature outlined in Section 2.

The scenario discusses the design and implementation of a semantic wiki, including the migration of existing content to a semantic structure and the design and implementation of innovative user interaction, i.e. the ways in which all possible end-users might want to exploit the deployed system. In its most "crowded" iteration the development DyNO involved 4 different organizations, in addition to 2 client organizations. All organizations were different in expertise and location. The minimum distance between organizations was in the order of tens of km, while the maximum distance spanned two continents.
The goals of the client were as follows: (a) revise the current version of the encyclopaedia using a semantic-wiki based technology; (b) improve the diversity and quality of user interactions with the encyclopaedia.

The project rolled out in three phases:

1. **Project Goal Analysis (approx. 1 month):** ArchiXL and clients discussed the project mission to identify objectives and agree on a project strategy. From this phase, an important organizational quality requirement emerged, namely missing expertise in Interaction Design [32]. This required ArchiXL to include an additional organization with the required expertise in the "development network" at the start of phase 1.

2. **Requirements analysis and solution design (2.5 months):** In a series of face-to-face workshops, the user interaction and the solution architecture were designed. About half of the workshops focused on interaction design and were led by the organization that was added as a result of phase1. The other half of the workshops focused on the architecture of the system-to-be and were led by ArchiXL. From this phase, additional organizational requirements emerged, most notably the need to scale up development workforce, to gain expertise in Javascript, and to get on board additional expertise in the area of animation design.

3. **Software Development (9.5 Months):** ArchiXL kicked-off this development phase after the involvement of two additional partners: (1) Animation Specialists Organization; (2) Development Support Organization.

Fig. 1 depicts the process through which the DyNO grew across the two phases of ArchiXL's project. Initially, clients

[1] `http://www.archixl.nl/`

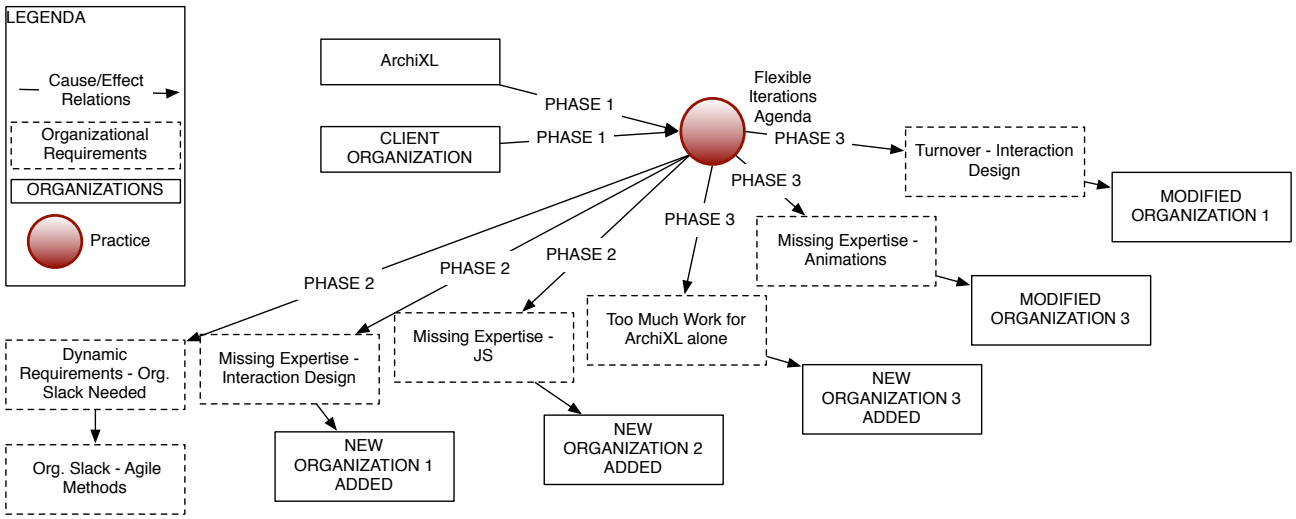

Figure 1: DyNOs forming in our scenario.

and ArchiXL sat to define the scope, objectives and budget for the project. A flexible development strategy and agenda was defined, to determine the increments needed for development. Also, a series of organizational requirements caused the addition of multiple partners at different points in time within the development agenda. Finally, emerging organizational requirements (employee turnover) caused modifications to the organizational structure in some organizations (e.g. Organizations 1 and 3 changed employees).

3.1 Social and Organizational Challenges

This section outlines the key social and organizational challenges we identified studying our case scenario. The following list elaborates on the social and organisational features emerging from the scenario (highlighted in bold):

- The development problem was extremely dynamic, i.e. featuring extremely innovative requirements with unclear and dynamic definitions. These requirements called for many skills in organizations other than ArchiXL. In addition, requirements were impossible to fully specify beforehand. Equally, requirements were explorative in nature (i.e. featuring unforeseen conditions and potential system users). This required the definition of a flexible development agenda or strategy, rather than a fixed plan. This caused **distress and skepticism** in the clients who expected a classic fixed "plan" with milestones and deliverables. Quoting from our focus group minutes: *"We divided the beast in iterations. In each we drew what we would be doing. For the first it was very clear, for the second it was somewhat clear, for the third one it was pretty much vague, and so on. So for each iteration we had a strategy on how to pursue the global mission, say - this has to be done, and this and this. The customer in the very beginning didn't really like that, they wanted a plan. They wanted to have a well-laid plan, with a timeline and day-to-day schedule. We were able to convince that that was not the right approach for this project."*.

- Again, the nature of the project required the adoption of agile methods (e.g. bi-weekly plenary sessions, reflection on bi-weekly increments, task-centric development). This choice was forced by two critical organizational requirements: (a) **common vision** on the project shared by all organizations in the development network; (b) **mutual trust** among organizations involved. Quoting from our focus group minutes: *"Even though we had time issues, budget issues, we never run into quality issues. Because that's the one thing that we really fixed and all agreed on. We would never diminish any of the quality no matter what, and everyone agreed on that. And everyone accepted that if more effort had to be put in, that was actually a calculated risk. That had to be done"*.

- The many organizations involved and the emerging organizational requirements rapidly resulted in a **complex DyNO**, long before any software code was written or design specifications worked out. This complexity forced ArchiXL to use the DyNO as a constraint to design the system. Moreover, requirements were extremely dynamic and definitely not clear right from the start. This forced **social and organizational ripple effects** to be handled on-the-fly. *"the project had many requirements with expertise that we don't have, like JavaScript for example - and it was a lot of work! This was the trigger to start looking for collaborators. The absence of all the needed organizations and people right from the start, i suspect was one of the main problems that we faced. We discovered this [the complete set of organizational needs] too late. If we had been able to have all organisations at the same table from the very beginning, when we were still defining what the project would look like, our life would have been easier"*.

- The project had an extremely tight budget and timeline. All contracted third-party organizations were hired on a fixed-price basis. Scope, budget, time and quality constraints were fixed as well and were agreed upon signing of contract. Consequently, risks in the analysed project were extremely high. Quoting from our focus group minutes: *"Mind you, the scoping and*

money were made clear upon agreements. And by doing that, organizations accepted a certain type of risk that they couldn't pull through the project". Rather than demoralising the networked organization, each SME involved was pulled by its **pride** to deliver expected quality under the agreed circumstances and constraints. **Engagement** across the whole network was kept high by this pride component. Quoting from our focus group minutes: *"I just recall the quote from one of the developers organizations, with a sort of spread of enthusiasm and pride saying - I don't know how you managed but you kept me engaged and working late to pursue my goals. These small organizations are very focused at what they do. And they also have a sense of pride, sort of personal loyalty to what's the project. And that kept the project going".*

- Every time a new organization was included in the DyNO, it had a different level of information and understanding on the project. As part of the inclusion within the DyNO, all organisations negotiated a new business process (i.e. a set of activities by all involved participants) for the DyNO. The process of negotiation made the project extremely **fragile and volatile**. In this negotiation, each organisation was forced to put away its domain-specific cookbook. This caused **disagreement and distress**, every time the DyNO "adapted". Quoting from our focus group minutes: *"[With every new partner] It took three to four weeks for every one to be at the same level of information. This period was tricky, at any point in this time we could have lost the project. A new partner would obviously initially lack the knowledge that had been shared and generated earlier in the project. Perhaps due to this, some would also bring with them a way of working that they were used to and which they assumed would fit this project too - which was not always the case. They expected to work with those and resisted the change forced by the innovation in our scenario. They were stuck on their cookbook. As soon as they let go of their way of working and what they thought was the right approach. And we leaned back a little bit and started thinking together what would be the right approach in this scenario, things started to get smoother".*

3.2 Ad-Hoc Organizational Practices

The description above elaborates on a rather vivid networked organization. This organization required for the adoption of ad-hoc practices to handle both complexity and fragility. Fig. 2 depicts the practices adopted and the effects they caused on the networked organization.

First, rather than designing a software architecture from functional/non-functional requirements ("classic" approach), ArchiXL used a "Conway in reverse" approach. Conway's Law [3] was used as an explicit software design constraint. Investigating organizational requirements first, ArchiXL used these explicitly to finalise the organizational structure and then use this to drive the definition of the software architecture. This allowed to smoothly organise the network and allocate tasks to skills effectively [18].
Second, all organizations involved were networked with each other and involved in each other's work through a shared, task-centric view. This transformed them from a set of loose organizational silos into a mesh of collaborative partner nodes. This increased collaboration and engagement across the network. This was key to increasing mutual trust.
Third, a task-centric view was adopted to divide work among collaborating participants in the networked organization. The task-centric view was shared among all participants and updated real-time. This allowed all partners to be aware and view all concurrent tasks allocated to fellow companies, thus increasing trust across the network.
Fourth, the networked organization inherited many agile practices from the Scrum way of working [29] (e.g. bi-weekly plenary meetings). These were mostly used to reflect and collaboratively solve tasks that could not be tackled in their allotted timeframe.
Fifth, finally, the flexible iterative development agenda was revisited bi-weekly, upgraded and agreed upon during reflection meetings. The networked organization agreed on the shared agenda increment. This allowed a plan to emerge spontaneously as the project unfolded.

4. DISCUSSION: WHAT LIES BEYOND?

To analyse our results we elicited a list of requirements from our scenario and operated a preliminary mapping with collaboration and governance frameworks from software engineering and GSE research [22, 14, 12]. As a result, we made initial observations and remarks (see Sect. 4.2). In addition, we compared with traditional software engineering approaches and evaluated the key differences.

4.1 Requirements in Developing with DyNOs

Social and organizational requirements evident in our scenario are summarised on Table 1. Column 1 and 2 provide requirements label and name, respectively. Column 3 elaborates a description, based on the evidence from our case scenario.

4.2 Observations and Remarks

To evaluate the dynamicity in our scenario on the three key dimensions for GSE projects, we mapped our requirements with the model from [22] (see Fig. 3 - square boxes contain our requirements labels from Table 1). More specifically, a requirement was mapped to a 3C model concept if the challenge represented by the requirement fell under that concept according to our industrial contacts. The 3C model states that three basic activities are dominant in (networked) organizations and their operations for software production: (a) communication with peers to realise organizational activities; (b) coordination of activities and tasks to achieve planned business goals; (c) cooperation on tasks that require concurrent and shared work/expertise. Underpinning these dominant activities is awareness [6], i.e. the perceived understanding of the current organizational state. The mapping in Figure 3 shows that DyNOs introduce many challenges for concepts in the 3C model. We made two observations. First, dynamicity in our scenario required ArchiXL to refine governance across all dimensions of the 3C model. New organizational requirements produced three effects: (a) compounded previous requirements, further complicating awareness maintenance across the development network; (b) required additional cooperation on new interdependent tasks; (c) required making explicit task-dependencies to aid coordination. More research is needed on all the above effects. For example, the best practices we elicited from ArchiXL's sce-

Table 1: Social and Organizational Requirements for DyNOs

Label	Name	Requirement Description
R1	Organizational Structure	DyNOs feature a complex and rapidly changing organisational structure. The structure must be made explicit through abstractions that allow measurement and adaptation. To instrument adaptation for DyNOs, the connected organizational requirements need to be managed explicitly, to ensure project success.
R2	Organizational Structure Traceability	Organizational structures are generated through organizational (i.e. socio-technical) decisions that match organizational requirements. Therefore, organizational requirements need to drive the software process. Also requirements and decisions taken to match them, need to be tracked much like common software projects keep track of software requirements and architecture decisions.
R3	Task-Centric Project Visualization	New and changing quality requirements necessitate cooperation on old and new tasks, to be shared across the development network. Visualising and (re-)allocating tasks and task dependencies is imperative.
R4	Reflection and Strategy for Incremental Planning	DyNOs are incompatible with "classic" project plans [28]. DyNOs need to agree on a schedule of fixed meetings and strategy in which to devise and divide work in a more agile and adaptable way. For example, ArchiXL used reflection meetings to decide work flexibly every iteration (see Fig 1).
R5	Awareness Maintenance	Awareness maintenance must become part of product maintenance procedures and costs. Maintaining awareness high across the development network makes sure that the networked organization stays agile and is able to react to new and changing organizational requirements. For example ArchiXL fostered trust in partners by granting open-access to task information and work distribution across the network (see Fig. 2).
R6	Ad-Hoc Business Process Sharing	DyNOs can reach success only by acting as an organized whole. In so doing, members of DyNOs agree on a shared business process to enact for development. For example, ArchiXL agreed on a shared business process with newcoming partners (see Sect. 3.1).
R7	Open-Source-Like Community Support	DyNOs exhibit many similarities with open-source communities. Pride, engagement and committment to the project become a much stronger pull than delivery or contract value. These community aspects need to be fostered. For example, ArchiXL used reflection meetings and other practices from agile methods to foster engagement (see Sect. 3.1).
R8	Core-Periphery Structure Explicitation	DyNOs exhibit a core-periphery structure [14]. This needs to be made explicit and supported throughout the project lifecycle. For example, ArchiXL as core contributor, used a task-centric view to visualize the distributed division of work with collaborating organizations (see Fig. 2).
R9	Explicit Open-Teams Support	DyNOs work as an organised whole, rather than a sum of loosely cooperating parts with tight decoupling. For example in ArchiXL, tasks were shared and incremental plans drawn and rethought at every iteration. Also, the organizational structure was adapted at every iteration (see Sect. 3.1).
R10	Organizational and Systems Requirements Slack	DyNOs need to prepare for the flexible implementation of both systems and organizational requirements. For example, ArchiXL was facing the presence of ever-evolving organizational and functional requirements. Both types of requirements needed explicit addressing (e.g. by adapting the DyNO and the system under development).
R11	Explorative Requirements Engineering	DyNOs are not assembled for every type of project. For example, ArchiXL was facing the presence of ever-evolving organizational and functional requirements. Both types of requirements needed explicit addressing. This and similar explorative scenarios likely require DyNOs.
R12	User-Centric Development	DyNOs likely work for the design and development of innovative, user-centric systems. These systems involve tackling a wider nature of requirements. For example, ArchiXL specifically mentioned that the client was trying to address emerging end-user needs with very specific and demanding characteristics. This forced ArchiXL and its partners to forget their classic cook-book and think up new ways to elicit and satisfy requirements from both the customer and the end-users.
R13	Explicit Conway Approach	DyNOs require the use of the Conway effect [3], in an explicit way. A clear picture of the organizational structure emerging in the networked organization must be used to drive the definition of workable software architectures. For example, in ArchiXL, some architecture decisions were explicitly dictated by division-of-work and work dependency requirements(see Fig. 2).
R14	DyNO Creation and Continuous Evolution	DyNOs require mechanisms to support the creation, visualization and continuous adaptation of networked organizations. Dynamic internal and external change can trigger network adaptation (e.g. by requiring the inclusion of additional partners). Change needs to be tracked and visualised to increase awareness. Effects produced on the DyNO and the assigned lifecycle need to be tracked.

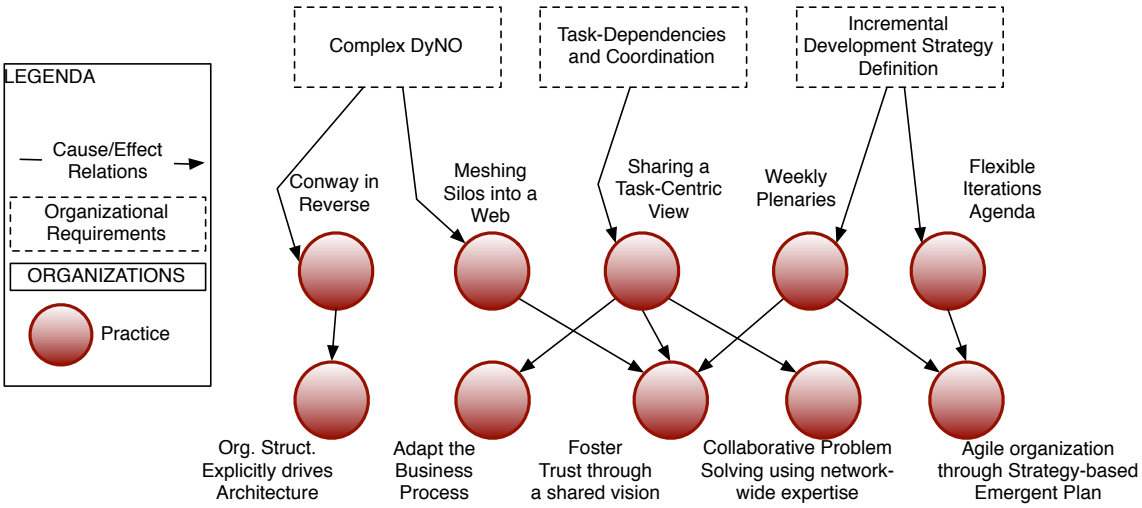

Figure 2: DyNOs management practices in our scenario.

nario can be further studied to determine their effectiveness in similar scenarios. Second, ArchiXL figured out quickly that agile methods produced two key benefits: (a) reflection sessions at the end of every iteration helped in reasoning on new organizational requirements for next iterations, i.e. to refine the development strategy; (b) frequent meetings helped increasing and maintaining high awareness across the DyNO, i.e. helped in awareness management. This suggests that the rise of agile methods in industrial practice could be due to an increased need for handling DyNOs.

Also, to understand the structural requirements resulting from our scenario, we compared with frameworks from [14] and [12]. This comparison revealed that DyNOs need visualisation and explicit support to emergent core-periphery organizational structures. However, visualisation and explicit support should model organizations, abstracting from the social network underneath. This entails developing task-centric and organizations-aware models to divide labour across a distributed development network. Task dependencies and automated task-progress tracking must be developed to support development with DyNOs.

Many requirements from our list however, cannot immediately map on the 3C model or the other literature we considered. This suggests that the "organization" concept is a pervasive and implicit inhabitant of the 3C model. Some uncharted research paths lie ahead. For example, from R10 and R11, *how can DyNOs be made to function even though organizational requirements are not clear ahead of start?* Also, from R11, *what is an ideal Explorative Requirements Engineering approach to be used in DyNOs?* or, from R14, *how does the practice of software architecting change in response to the continuous (co-)evolution of its DyNO?*

4.3 Key Differences with Traditional Software Engineering

Comparing the operations of DyNOs with traditional approaches we identified four key differences.

First, traditional engineering approaches use incremental-iterative development of requirements or tasks. For example, in agile methods incremental and iterative task-solving is used to carry out development. Conversely, DyNOs have **no**

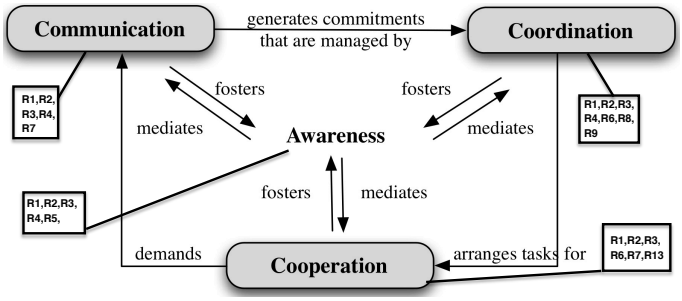

Figure 3: 3C Model from [22], as mapped with our requirements.

reference process model. Rather the process emerges dynamically, based on organizational requirements that alternate their way during development. For example, ArchiXL switched between classic plan-based iterative/incremental development with a flexible task-based agenda to accommodate new organizational requirements. Also, DyNOs combine incremental and iterative approaches with what can be called *paused-execution development*, i.e. stop developing tasks when people collectively understand that coordination is too difficult in the current state.

Second, DyNOs start their operation following a **Reverse-Conway regime**. Following this regime means studying the organizational structure strengths, weaknesses opportunities and threats to infer architecture and coordination requirements. This also means that DyNOs' initial configuration almost becomes the software architecture. For example, ArchiXL chose to draw and allocate tasks and their dependencies explicitly following the organizational boundaries and their characteristics.

Third, traditional software engineering organizations are not required to change the internal development business process. **DyNOs negotiate ad-hoc business processes** shared among participants. The process is built ad-hoc and adapted as needed. For example, ArchiXL and partners needed to renegotiate the business process currently in place, every time a new partner was added to the network.

Fourth, traditional software engineering is planned and organised around a clear vision of the system to be developed. DyNOs first require eliciting organizational-social structure [25] needs. For example, ArchiXL started development before having a clear vision of needed organizational needs. This forced a *paused-execution*, to backtrace organizational needs and use them to increase the development network.

5. CONCLUSION

This paper explores a real-life scenario in which a dynamic networked organization, which we call a DyNO, was created by SMEs to collaboratively work on a GSE project. We analysed available data eliciting a list of social and organizational requirements. Comparing these requirements with frameworks for collaboration and governance revealed some initial observations. For example, numerous unexplored venues for research beyond current software engineering and GSE. Also, the organization and organizational requirements for software projects seem to be an absent inhabitant of the 3C model from [22]. We conclude that additional research must be invested in how software engineering is carried out using DyNOs in continuous dynamic evolution. Finally, new requirements engineering and design techniques are need to cope with functional-/non-functional as well as organizational requirements for engineering with DyNOs. These new techniques, as evidenced by our case scenario, are essential for development success.

6. REFERENCES

[1] L. C. Abrams, R. Cross, E. Lesser, and D. Z. Levin. Nurturing interpersonal trust in knowledge-sharing networks. *The Academy of Management Executive*, 17(4):64–, 2003.

[2] C. Bird, N. Nagappan, H. Gall, B. Murphy, and P. Devanbu. Putting it all together: Using socio-technical networks to predict failures. In *Proceedings of the 2009 20th International Symposium on Software Reliability Engineering*, ISSRE '09, pages 109–119. IEEE Computer Society, 2009.

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

[4] K. Crowston, K. Wei, J. Howison, and A. Wiggins. Free/libre open-source software development: What we know and what we do not know. *ACM Comput. Surv.*, 44(2):7, 2012.

[5] D. Damian. Stakeholders in global requirements engineering: Lessons learned from practice. *IEEE Software*, pages 21–27, 2007.

[6] D. Damian, L. Izquierdo, J. Singer, and I. Kwan. Awareness in the wild: Why communication breakdowns occur. In *ICGSE*, pages 81–90. IEEE, 2007.

[7] C. Ebert and P. D. Neve. Surviving global software development. *IEEE Software*, 18(2):62–69, 2001.

[8] J. D. Herbsleb. Global software development at siemens: Experience from nine projects. In *Proceedings of the International Conference on Software Engineering (ICSE 05)*, pages 524–533, 2005.

[9] J. D. Herbsleb. Global software engineering: The future of socio-technical coordination. In L. C. Briand and A. L. Wolf, editors, *FOSE*, pages 188–198, 2007.

[10] J. D. Herbsleb and A. Mockus. An empirical study of speed and communication in globally distributed software development. *IEEE Transactions on Software Engineering*, 29(6):481–94, 2003.

[11] http://www.opengroup.org/projects/soa book/. Soa source book.

[12] R. Kazman and H.-M. Chen. The metropolis model and its implications for the engineering of software ecosystems. In G.-C. Roman and K. J. Sullivan, editors, *FoSER*, pages 187–190. ACM, 2010.

[13] F. Lanubile, C. Ebert, R. Prikladnicki, and A. Vizcaino. Collaboration tools for global software development. *IEEE Software*, 27:52–55, 2010.

[14] C. Manteli, H. van Vliet, and B. van den Hooff. Adopting a social network perspective in global software development. In *ICGSE*, pages 124–133. IEEE Computer Society, 2012.

[15] N. Narendra, L.S. Le, A. Ghose, and G. Sivakumar. Towards an architectural framework for service-oriented enterprises. In A. Ghose, H. Zhu, Q. Yu, A. Delis, Q. Sheng, O. Perrin, J. Wang, and Y. Wang, editors, *Service-Oriented Computing - ICSOC 2012 Workshops*, volume 7759 of *Lecture Notes in Computer Science*, pages 215–227. Springer Berlin Heidelberg, 2013.

[16] L. Northrop, P. Feiler, R. P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, D. Schmidt, K. Sullivan, and K. Wallnau. Ultra-Large-Scale Systems - The Software Challenge of the Future. Technical report, Software Engineering Institute, Carnegie Mellon, June 2006.

[17] R. Prikladnicki and J. L. N. Audy. Managing global software engineering: A comparative analysis of offshore outsourcing and the internal offshoring of software development. *IS Management*, 29(3):216–232, 2012.

[18] I. Richardson, V. Casey, J. Burton, and F. McCaffery. Global software engineering: A software process approach. In I. Mistrik, J. Grundy, A. van der Hoek, and J. Whitehead, editors, *Collaborative Software Engineering*. Springer, January 2010.

[19] R. Sangwan, M. Bass, N. Mullick, D. J. Paulish, and J. Kazmeier. *Global Software Development Handbook (Auerbach Series on Applied Software Engineering Series)*. Auerbach Publications, Boston, MA, USA, 2006.

[20] I. Sommerville. *Software Engineering*. Addison-Wesley, Harlow, England, 9. edition, 2010.

[21] R. Stallman. Why Open Source misses the point of Free Software. *Viewpoints*, 52(6):31–33, 2009.

[22] J. Swart and S. C. Henneberg. Dynamic knowledge nets - the 3c model: exploratory findings and conceptualisation of entrepreneurial knowledge constellations. *J. Knowledge Management*, 11(6):126–141, 2007.

[23] D. Tamburri and P. Lago. Supporting communication and cooperation in global software development with agile service networks. In *Software Architecture*, Lecture Notes in Computer Science, pages Vol. 6903, 236–243. Springer Berlin / Heidelberg, 2011.

[24] D. A. Tamburri, E. di Nitto, P. Lago, and H. van Vliet. On the nature of the GSE organizational social

structure: an empirical study. *proceedings of the 7th IEEE International Conference on Global Software Engineering*, pages 114–123, 2012.

[25] D. A. Tamburri, P. Lago, and H. van Vliet. Organizational social structures for software engineering. pages 1–35. To appear on ACM Computing Surveys, 2012.

[26] D. A. Tamburri, P. Lago, and H. van Vliet. Uncovering latent social communities in software development. *IEEE Software*, 30(1):29 –36, jan.-feb. 2013.

[27] D. A. Tamburri, P. Lago, and H. v. Vliet. Service networks for development communities. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 1253–1256, Piscataway, NJ, USA, 2013. IEEE Press.

[28] J. C. van Vliet. *Software engineering - principles and practice.* Wiley, 1993.

[29] L. Williams, G. Brown, A. Meltzer, and N. Nagappan. Scrum + engineering practices: Experiences of three microsoft teams. In *ESEM*, pages 463–471. IEEE, 2011.