A simple model of developer interaction in software projects

SUBHAJIT DATTA, School of Information Systems Singapore Management University
KISHORE KUNAL, International Institute of Information Technology, India
HAZIQSHAH WIRA, Temasek Polytechnic, Singapore
MOSES BAN, Temasek Polytechnic, Singapore
SUBHASHIS MAJUMDER, Heritage Institute of Technology, India

Developer interaction plays a key role in the development of large and complex software systems. Understanding the dynamics of such interaction is essential in the design of tools and processes to facilitate successful software development. Clustering is an important attribute of developer interaction, and is known to influence team outcomes. In this paper, we present a simple model of developer interaction to simulate the variation of clustering over the time. Our model draws on the dynamics of interaction across the different phases of the software development life cycle, as well as the need for impromptu connections between developers. Preliminary validation of the model's output using real-world data from two types of developer interaction networks, shows close congruence between empirical and simulated values. These results indicate that our model is able to capture an important aspect of developer interaction, and offers insights on how software development teams can be effectively assembled and governed.

CCS Concepts: • **Software and its engineering → Programming teams**.

Additional Key Words and Phrases: SDLC, interaction, clustering, model, simulation, empirical software engineering

ACM Reference Format:
Subhajit Datta, Kishore Kunal, Haziqshah Wira, Moses Ban, and Subhashis Majumder. 2018. A simple model of developer interaction in software projects. 1, 1 (November 2018), 18 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

1.1 Context

Over the past few decades, software systems of increasing criticality are influencing many aspects of our lives. These systems are generally built by large teams of developers who are distributed across geographies. The interaction between members of such teams influences the quality of work products in subtle and interesting ways [11]. It is important to understand key components of such interaction, so that effective strategies for team assembly and governance can be devised. With that perspective, this paper presents the development and validation of a simple model to examine the temporal variation of a important characteristic of developer interaction.

Large scale software development is a complex enterprise. Some aspects of the complexity arise from the properties inherent to software [3], while others are derived from the human-centric nature of software development [13].

Authors' addresses: Subhajit Datta, subhajit.datta@acm.org, School of Information Systems Singapore Management University, 80 Stamford Road, Singapore, Singapore, 178902; Kishore Kunal, International Institute of Information Technology, Bangalore, India; Haziqshah Wira, Temasek Polytechnic, Singapore; Moses Ban, Temasek Polytechnic, Singapore; Subhashis Majumder, Heritage Institute of Technology, India.

Addressing the effects of temporal and spatial separation between team members also pose challenges unique to large scale software development [34]. Thus organizations seek a deeper understanding of the essential elements that drive developer interaction, with a view to designing effective tools and processes. As with any other complex phenomena, modelling offers a way to extract the elements of interest using a simplified representation of reality [31].

1.2 Towards a developer interaction model

Abstracting the software development ecosystem as a network – with vertices representing individuals and edges denoting some facet of interaction between them – can offer a useful mechanism for studying and modelling developer interaction, as has been demonstrated across diverse collaborative systems [18]. A network representation allows us to identify elements of developer interaction that are of interest to us, using established metrics from network science. One such aspect is the nature and extent of *clustering* of interacting developers. As in many other large scale human enterprises, in the software development too, clustering is an indication of the extent of collaboration [28]. Clustering reflects how closely developers interact, as they work on individual tasks aligned to the team's collective objectives. Understanding the level of clustering at specific junctures of the development life cycle has strong implications at individual and organizational levels. For a developer, clustering is a key mechanism for integration into the team's fabric, and leveraging the development ecosystem's "tribal memory" towards completing his/her individual tasks [2]. Facilitation of clustering by reducing temporal and spatial asynchrony [34], and tuning tools and processes appropriately [11] can be effective measures towards successful project governance. Developer clustering has also been found to be significantly related to team outcomes in large scale software development [10].

The software development life cycle (SDLC) is segregated into *phases* according to the shifting focus of development activities, from the initiation of a system's development to its full operability by end users [21]. Consequently, developer interaction characteristics such as clustering also vary across the duration of the SDLC. The temporal variation of clustering has practical implications. A key concern in the governance of software projects with large and distributed teams is to facilitate easy communication between developers. Ease of communication is influenced by several factors, such as co-location of team members who are likely to interact more closely with one another, and devising tools and processes that can mitigate challenges arising from differences in time-zones. Thus, being able to identify the junctures of the SDLC where developers interact more closely with one another, and the factors that influence the variation of such interaction over time, can offer actionable insights for organizations on how members of teams can be distributed across geographies.

Developers communicate with one another to share information or expertise towards the fulfilment of the team's collective responsibilities. At the individual level, developer interaction is driven by local concerns, out of which global characteristics of the team's communication *emerge* over time. The objective of our model is to identify factors that influence the temporal variation of one such characteristic – the level of clustering between developers. Drawing on the changing communication structure across the phases of the SDLC, the variation of clustering as simulated by our model is able to match the empirical data from a large real-world software development ecosystem, with reasonable accuracy. This indicates that the model is able to identify key influencers on developer interaction in large scale software development ecosystems.

In the study of software development ecosystems, models have a critical role to play. While examining developer interaction, the traditional pillars of scientific enquiry – theory and empiricism – are challenged by the unique characteristics of large scale software development. Given the distinct nature of software as an industrial artefact [3], and the deeply people-centric processes through which software is designed, developed, and deployed [13], an all

embracing theory of software engineering has so far proved elusive, in spite of recent efforts[1]. Controlled experiments – the gold standard of causal inference – are very difficult, if at all possible, to design and execute in real world software development scenarios, given the infeasibility of segregating developers into control and treatment groups, while the system is being developed to fulfil the needs of real users. Thus, the most recent pillar of science – simulation – offers a promising direction. As we demonstrate in this paper, a simple model can illuminate many interesting scenarios, and when model output is matched with empirical data, useful insights can be derived.

1.3 Organization of the paper

In the next section, we present a background of how activities of the software development life cycle relate to our model (Section 2) followed by an outline of related work in Section 3. Subsequently, we develop our model in Section 4, and validate it with empirical data in Section 5. Threats to the validity of our results are identified and addressed in Section 6 along with our plans of future work. We next discuss the insights we derive from the results, and how our results can be useful in Section 7. The paper ends with a summary and conclusions in Section 8.

2 BACKGROUND

2.1 Developer interaction across the phases of software development

The influences of the structure of communication between developers on the structure of the software system being developed has long been recognized [8],[6], [25]. Several studies have also established how communication challenges in the team affect the quality of the software that is produced [7], [5]. As developers communicate across the SDLC, a *network* of interactions grows over time. In the context of this study, we define a general *developer interaction network* N to consist of a set of vertices $V = \{v_1, v_2, ..., v_x\}$ representing developers. In the period of study, two vertices v_m and v_n are connected by an undirected link (or edge) e_{mn}, if v_m and v_n have *both* engaged on at least one *common* development activity of interest, such as commenting on a particular bug, committing code for a particular work item etc.. Thus the set of edges E of N represents all instances of developer co-participation for given activity and time-period. Accordingly, for a specific development activity α, and time-period t_x, a specific instance of the developer interaction network will be denoted by $N_\alpha(t_x)$.

The whole point about segregating the SDLC into the phases of of Inception, Elaboration, Construction, and Transition is to "guide the division of work and the shifting of focus and priorities across the development life cycle" [9]. Each phase has distinct objectives, as is manifested in the different mores of interaction between developers. Consequently, as development proceeds through these phases, the structure of N also changes. With reference to Figure 1, we will now describe how developer interaction differs across phases, and how the the changing structure of N is captured in our model [2].

The *Unified Software Development Process* formalized segregation of the SDLC into the aforementioned phases [21]. However, the concerns identified in the phases are universal to iterative and incremental software development, and not merely artefacts of a particular methodology. Thus, if a project follows one among the many approaches collectively called "agile methods", the team has to progressively address issues germane to Inception, Elaboration, Construction, and Transition, even if the corresponding stages of development are called by other names. This enables the applicability of our model irrespective of the specific methodology being used.

[1] http://semat.org/

[2] Note: In the interest of clarity, the number of vertices and edges of the networks in Figure 1 have been arbitrarily chosen, and they do not correspond to the model validation described later in the paper.

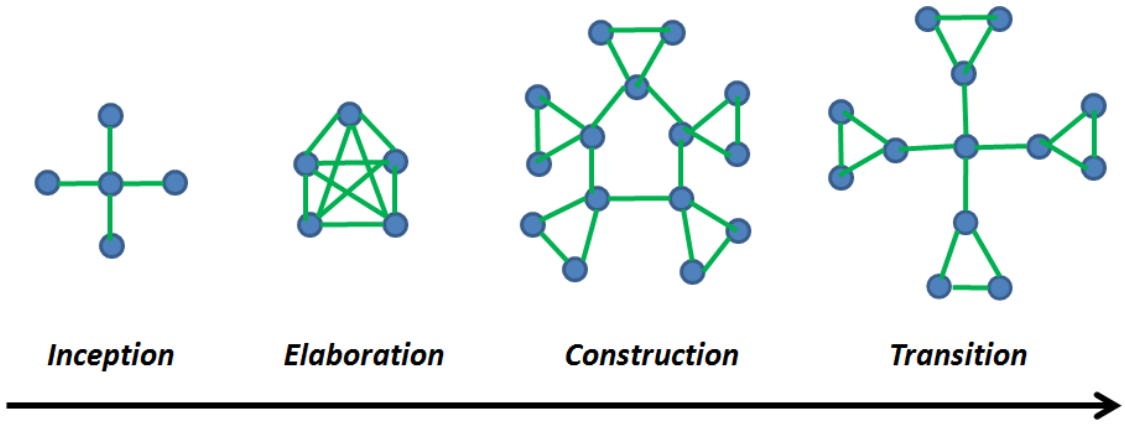

Fig. 1. Changing structure of developer interaction networks across the phases of the software development life cycle (SDLC)

2.1.1 Inception. Inception seeks to establish feasibility of the project, and developers are engaged in determining system scope and boundary, delineating system architecture, identifying critical risks, and building a proof of concept. The end of Inception is marked by the *life cycle objective* (LCO) milestone, which makes a case for either going forward with the project, or aborting development [21]. Inception starts with an individual being assigned to lead the project, who in turn engages a set of senior developers, who will eventually lead the development of modules. The project lead (PL) individually briefs each module lead (ML) on the context, deliverables, deadlines; and identifies the specific line of functionality each module has to implement. As each ML tries to understand the boundary and scope of his/her module, the MLs essentially work in silos during this phase, their sole point of contact being the PL. We thus model the structure of team interaction during Inception as a star network, with the central vertex – representing the project lead – connected to each peripheral vertex – representing a module lead, as shown in the first sub-figure from the left in Figure 1 .

2.1.2 Elaboration. Assuming the LCO has successfully established the feasibility of the project at the end of Inception, the project moves into Elaboration. Inception was concerned with the exploration of *whether*; Elaboration is focussed with the exposition of *how*. During this phase, the development team needs to create an architectural baseline, identify significant risks, specify quality attributes, extend the coverage of use cases, and prepare schedule and cost estimates. The *life cycle architecture* (LCA) milestone – which lays out a clear strategy for how the project will proceed – marks the end of Elaboration [21]. As evident from the issues Elaboration is primarily concerned with, in this phase, every member of the project team – the MLs, as well as the PL – start communicating closely among themselves, discussing how the modules will interface with one another, and building a common development plan for the whole system. So, essentially everyone is talking to everyone else in the team during this phase. Accordingly, the structure of team interaction during Elaboration is modelled as a fully connected graph, as shown in the second sub-figure from the left in Figure 1.

2.1.3 Construction. The project next moves on to the Construction phase. True to its name, Construction is about building the system. During this phase, the team works on extending use case identification, commencing or completing analysis, design, implementation, and testing related activities, maintaining integrity of architecture, and monitoring

and mitigating risks. The end of Construction is marked by the *initial operational capability* (IOC) milestone, which demonstrates basic workings of the system [21]. As Construction envelopes some of the central activities of the software development life cycle, project teams typically grow in size during this phase, with new members joining each module as developers (D). The team gets segregated into smaller clusters, each developing a particular module. Developers working on a particular module need to communicate very closely with one another, as low-level issues such as choices of data structures and algorithms are debated and decided upon. Each module lead, on the other hand, needs to remain in contact with leads of those other modules his/her module shares *interdependencies* with, so that module interfaces can be refined and kept in conformity. Accordingly, the structure of team interaction during Construction is modelled with the MLs connected together in a ring[3]; and each ML being part of a clique – a fully connected sub-graph – which represents the interactions between developers working on that module, as shown in the third sub-figure from the left in Figure 1. In a sense, this structure is similar to the "caveman graph" [36], with each "cave" being inhabited by the developers working on a particular module.

2.1.4 Transition. After Construction, the project moves on to the Transition phase where the users are prepared for operating the system, the system is tuned for actual operating conditions, and defects found in the user acceptance testing are rectified. The *product release* (PR) milestone marks the end of Transition [21]. During this phase, some of the developers are released from the project, bringing down the total head count of the team. Transition needs to ensure that each module combines seamlessly into the entire product. The PL plays a central role in this integration, communicating directly with the MLs, while the developers within each module continue interacting with their peers. The interaction structure during Transition is represented by a variant of the star network, with the PL at the center, and each peripheral vertex representing an ML, and each ML being a member of the clique of her module's developers, as shown in the fourth sub-figure from the left in Figure 1.

2.1.5 Impromptu connections. So, over the phases of Inception, Elaboration, Construction, and Transition we have changing structures of the team network as influenced by the developer interaction characteristic of that phase. However, there is also an *impromptu*, or need-based element in developer interaction that may go beyond the constraints of the network topology of a particular phase. In the software development context, developers of a particular module often reach out to developers of other modules, by way of general information sharing, and offering or receiving assistance for some particular task. Such impromptu connections allow critical knowledge and expertise to percolate within the team, and facilitate the resolution of unexpected issues. To address this situation, our model includes a parameter p, which is the probability of *any* two unconnected developers being joined by an edge, irrespective of whether the network structure of the current phase mandates a link between them.

2.2 Measuring clustering in the development interaction networks

As outlined in Section 1, our model seeks to identify factors that influence the level of developer clustering as they interact across the SDLC phases. To measure varying clustering levels, we use an established network science metric: the *clustering coefficient* [1]. Clustering coefficient (C_v) for a vertex v in a network is defined as follows: If v has a degree of k_v, that is there are k_v vertices directly linked to v, then the *maximum* possible number of edges between these k_v vertices is k_v *choose* 2, that is, $\binom{k_v}{2} = \frac{k_v(k_v-1)}{2}$. If the *actual* number of such edges existing is N_v, then

[3]Here, we make the simplifying assumption that each ML is only connected to her two adjacent MLs, one on either side. Implications of such assumptions are discussed in Section 6.

$C_v = \frac{2N_v}{k_v(k_v - 1)}$. Thus, the clustering coefficient of a vertex is the ratio of the actual number of edges existing between its neighbours and the maximum number of such edges that can exist. The level of clustering in the entire network is taken as the average clustering coefficient across all vertices [36]: $\bar{C} = \frac{1}{n}\sum_{v=1}^{n} C_v$. On the basis of the above definitions, a network's clustering can only lie between 0 and 1, both values inclusive.

With this background, let us consider some of the related work in this area.

3 RELATED WORK

Over the past two decades, global software development has increasingly led to the involvement of large and distributed teams in building and maintaining software systems. Whether and how interactions between developers in such scenarios relate to team outcomes have been studied from various perspectives. Herbsleb and Mockus have used data from source code management systems and surveys to explore the mechanisms of delay between the completion of work that is co-located versus work that is distributed [19]. Cataldo and Herbsleb have studied how communication networks in geographically distributed teams have evolved over time; they found that a group of developers emerged as connectors across geographic locations [5]. Studying communication networks in a large commercial project, Ehrlich and Cataldo examined how individual developers' performances are related to their respective positions; they found developers to perform better if they were more central in their team's communication networks, but worse when they were more central in the overall project [14]. The effects of geographic separation and temporal separation on the time to respond and the propensity to respond among developers in a distributed development team were examined by Wagstrom and Datta, and the authors found the former to have little impact, whereas the latter has significant impact [34]. In a study of large scale product development data involving a distributed team of developers, higher intra-team connection was found to relate to more defects in the team's work products, whereas more clustering among developers related to fewer defects [10]. This, among other studies underscores the importance of developer clustering in team outcomes in large scale software development. Although the aforementioned results represent a small selection of the wide and varied studies around developer interaction and individual or team level outcomes, some methodological aspects are common to such studies. Developer interaction is invariably abstracted as a network, with vertices representing developers, and links between developers signifying some sort of contextual association.

Various network models have been proposed for a wide range of contexts, where individuals interact in some collective enterprise [29]. Such models usually assume a particular network structure and then simulate network characteristic(s) by addition and/or deletion of vertices and/or edges according to some protocol. The utility of these models comes from identifying the relation between aspect(s) of the interaction being studied and some parameter(s) of interest. Effective models are able to simulate some interesting pattern of interaction by varying a set of parameters. Once such well known model is Schelling's model of segregation; it offers deep insights into the effects of "discriminatory individual choices" on collective human behaviour [32]. Another such model illuminated the so-called "small-world phenomenon" in large networks of interacting human individuals [37]. Guimera et al. have investigated team assembly mechanisms and team performance across a wide range of artistic and scientific pursuits, using a network based model [18].

Within the general framework of using network models to study complex behavioural dynamics, our study is positioned in the specific context of developer interaction in large scale software development. As discussed earlier, we have abstracted the software development ecosystem as a developer interaction network whose structure changes over the progression of the SDLC. In our proposed model, interactions are influenced by the extant network structure, as well

as the need for impromptu connections between developers. Each developer is embedded in his/her local development context and is driven by goals specific to a current phase of the SDLC. However, interactions between individual developers manifest in global characteristics of the network, such as the clustering coefficient. This leads us to consider *agent-based modelling* (ABM) as a paradigm for developing our model.

ABM is a relatively recent approach for investigating dynamics of systems where many individual entities – the "agents" – interact in some collective enterprise, with micro-scale behaviour leading to macro-scale outcomes [22]. ABM has been widely applied, in fields such as manufacturing [24], logistics and supply chain management [23], operations research [12], team assembly [15], and analysis of bibliometric data [30]. It has also been identified as a promising approach towards a deeper understanding of how the entire scientific ecosystem (SciSci) functions [16]. ABM is specially suited for modelling systems where straightforward rules of engagement between the agents can not be easily inferred, given the complexity of the environment the agents operate in. In the context of this study, the usefulness of ABM comes out of its focus on the interaction between agents as they work on individual activities towards the fulfilment of collective goals, leading to the emergence of system level characteristics.

Models of interaction in the software development context have so far been predominantly statistical in nature, drawn from observational studies (Section 3). As highlighted earlier, such approaches – though useful – have their limitations in understanding many of the dynamics of large scale software systems. In this paper we present an agent-based model and its validation as a complementary approach towards a deeper understanding how software development ecosystems function.

4 MODEL DEVELOPMENT

4.1 Model implementation and parameters

We use *NetLogo* – a programming environment that supports the development and running of agent-based models [38] – to implement our model. NetLogo is widely used for developing and running models in diverse areas [20]. Many well known models of collective human behaviour including the ones mentioned earlier – Schelling's model of segregation [32], Watts and Strogatz's model of the "small-world phenomenon" [37], and Guimera et al.'s team assembly mechanisms [18] have been illustrated using NetLogo[4] [5] [6].

Figure 2 shows the user interface of our NetLogo model[7]. The model has three sets of parameters S, D, and F, each allowing us to define one of the following aspects of the development ecosystem:

- S: Team *size*
- D: Project *difficulty*
- F: *Flexibility* of developer interaction

With reference to background of the model (Section 2), each set contains the following parameters. In S, we can specify m = the number of modules (each having one lead) and n = the number of developers working in each module; this helps us define the size of the teams in the development ecosystem. In D, we can specify i = the number of time-steps in each of the phases; this serves as a proxy for the difficulty of the project, as "typical" and "difficult" projects are differentiated by the amount of development time allocated to each phase [21]. In F, we can specify p – the probability of any two developer getting connected on an impromptu basis, as defined in Section 2.1. This parameter defines the

[4] https://ccl.northwestern.edu/netlogo/models/Segregation
[5] https://ccl.northwestern.edu/netlogo/models/SmallWorlds
[6] https://ccl.northwestern.edu/netlogo/models/TeamAssembly
[7] The model is available at http://bit.ly/hybrid-model

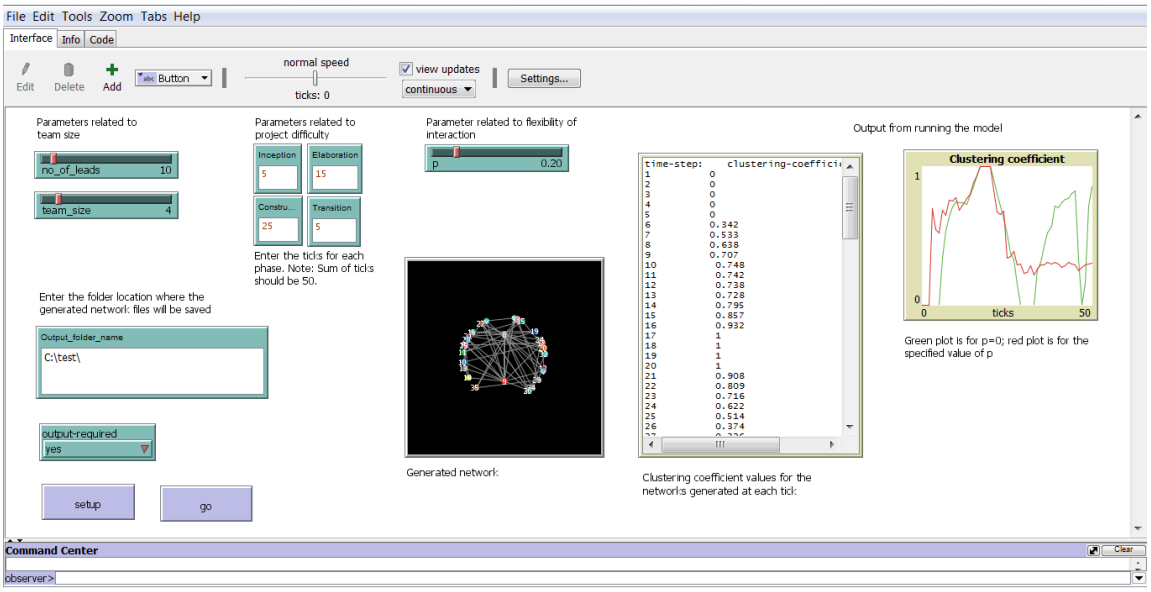

Fig. 2. User interface of our NetLogo model with details from a trial run.

level of allowance developers can exercise to bypass the modular communication structure of the team, thus p reflects on the *flexibility* of the developer interaction.

A "run" of the model stretches across 50 *ticks* – "tick" being the NetLogo parlance for a time-step. [8] At the end of each run, the model output consists of a set of 50 network files in the Pajek format[9], each file corresponding to the network for a particular time-step. The choice of 50 time-steps as the total project duration is based on considering factors such as the number of time-steps per phase, speed of execution of the model, and ease of visualizing the results. The pattern of the simulated results have been verified to remain unchanged if total duration is changed to 100 time-steps, 200 time-steps etc.. Over the life-span of a software development project, different SDLC phases occupy different durations. This is expected, given the distinct objective and outcome of each phase. For a "typical" project, 10% of the time is expended in Inception, 30% in Elaboration, 50% in Construction, and 10% in Transition [21]. Accordingly, out of the total 50 time-steps of a model run, Inception stretches across 5 time-steps, Elaboration across 15 time-steps, Construction across 25 time-steps, and Transition across 5 time-steps. The model interface allows the user to change the distribution of the ticks across the phases, to run the model for other atypical projects.

As our model has a probabilistic parameter p, we construct the networks over the phases – as shown in Figure 1 and further outlined in Section 2 – with a *randomized* selection of vertices and edges for addition/removal, at each juncture. This is a standard procedure in the development agent-based models and ensures no bias is introduced in the selection of entities and their interactions [38]. In our context, randomization ensures that our results are agnostic to any particular developer selection criteria within the team, and thus can offer general insights on developer interaction dynamics.

[8] In subsequent discussion, "tick" and "time-step" will be used interchangeably.
[9] https://gephi.org/users/supported-graph-formats/pajek-net-format/

During Inception, we start with a given number of vertices and select one vertex as the central vertex; the remaining vertices are then connected to the central vertex over the ticks to form a star network. Next, over the ticks of Elaboration, edges are added to form a fully connected graph. During Construction, this network structure is first transformed to a ring – by removing edges – and then cliques are appended to each vertex in the ring, by again adding selected vertices and edges. Over Transition, the cliques are removed to revert to a ring network, one of its vertices is then chosen to be the central vertex, and edges are removed to transform the network to a star structure; with vertices and edges being added to form cliques appended to the peripheral vertices of the star. When the probability of impromptu connections, p is non-zero, then in each tick of the model's run, *p times the number of non-present edges* are randomly added to the current network structure. The number of non-present edges are calculated by subtracting the current number of edges in the network from the maximal number of edges.

4.2 Scaling the output

From the above discussion it is evident that our model – as with any other model – represents an idealized view of reality[10]. For example, as modelled, at certain junctures of the development life cycle, fully connected graphs are generated which will lead to the highest level of clustering between developers. This implies everyone is connected to everyone else in the project team. However, in real world projects, *every* single member of a team may not be interacting with every other member at corresponding junctures of the SDLC. To adjust for such aspects while validating model output vis-a-vis real world data, we need to introduce a *scaling factor* (s); a positive rational number which is used to linearly scale the model output for every tick. The value of s for particular instances of model output is decided empirically, as illustrated during validating the model.

After presenting the development of the model, we next compare the model output with real-world data.

5 MODEL VALIDATION

5.1 Simulations from the model

As discussed in Section 4.1, the model's parameters allow us to specify team size, project difficulty, and flexibility of developer communication. In a development ecosystem, team size is defined by governance related factors such as project scope, budget, deadline etc., and the difficulty of a project is estimated from historical data within and across development organizations. However, the extent of impromptu connections is defined by needs specific to a team. To understand how impromptu connections affect the variation of clustering over the SDLC, we run our model with different values of p, while keeping other parameters constant; the results are presented in Figure 3.

As evident from Figure 3, the variation of clustering over time generally shows non-linear, and non-monotonic characteristics. When we do not consider any impromptu connections ($p = 0$), clustering reaches it highest value, falls precipitously to its lowest value, and rises again. As p increases, the range of variation narrows, even as the general characteristic remains the same, as seen in the plots for $p = 0.25, 0.5, 0.75$. Following a similar pattern, when there is highest probability of random connections ($p = 1$), clustering quickly attains its maximum value of 1 and remains unchanged. The variation of clustering with different values of p as seen in Figure 3 indicates notable influence of impromptu connections on developer interaction characteristics. To examine whether and how these characteristics relate to the real world of software development, we next validate our model using empirical data.

[10] Implications of such idealized assumptions are discussed in Section 6

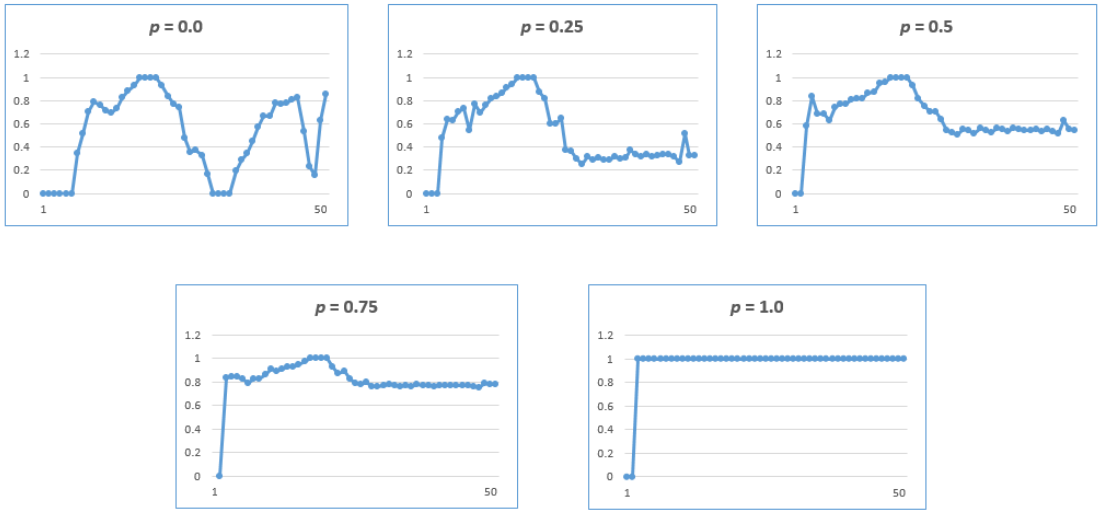

Fig. 3. Different characteristics of clustering coefficient variation (y-axis) across 50 time-steps (x-axis) with different values of p, as simulated by the model with number of modules m = 10, and number of team members, n = 4 in each case; for a "typical" project.

5.2 Empirical data for validation

For validation, we use the OpenStack[11] dataset as presented in a curated form by Gonzalez-Barohana et al. [17]. OpenStack is an open source platform for cloud computing, enabled for deployment as infrastructure-as-a-service (IaaS)[12]. The given corpus consists of four databases, covering *source code* (135 repositories; 183,413 commits; 3,836 authors), *problem tickets* (55,044 tickets; 635,895 updates; 7,582 identities), *mailing list* (15 lists; 88,842 messages; 4,399 posters), and *reviews* (119,989 code reviews; 3,533 submitters) [17].

This dataset is specially suited for validating our model as it allows us to examine developer involvement across multiple SDLC activities. Developers in large software ecosystems vary widely in their level of involvement. To ensure that we consider only those developers who are most actively involved, we filtered the dataset by identifying a set of developers (A) who have participated in *all* four activities as reflected in our dataset: committing code, involvement in problem ticket resolution, participating in mailing list discussions, and peer reviewing code. There were $|A| = 879$ developers who met this criterion.

As in the development and maintenance of large-scale systems, in OpenStack too, problem tickets are raised by testers or end users. Developers discuss the tickets by exchanging comments, code changes are made to address the problems, and the changed code is committed to the repository. Thus commenting on problem tickets and committing code changes are critical contexts for developer interaction. Accordingly, we instantiate two different types of developer interaction network (N) as defined in Section 2.1: *co-changes network* N_h denotes the network of interacting developers as they commit changed code to repositories and *co-comments network* N_c denotes the network of interacting developers as they comment on problem tickets. To reiterate the specific constructions of N_h and N_c within the general definition of N as introduced in Section 2.1, we note that vertices of both N_h and N_c are developers. Two vertices of N_h are

[11] https://www.openstack.org/
[12] en.wikipedia.org/wiki/OpenStack

connected by an undirected link (edge) if both the corresponding developers have committed code for at least one common unit of change; and two vertices of N_c are connected by an undirected link (edge) if both the corresponding developers have commented on at least one common problem ticket.

Comments on problem tickets and code change commit related information is available in our dataset from 2010 to 2017. We segregated this time period (T) into 50 time-steps $\{t_1, t_2, ..., t_{50}\}$ of equal duration. For each time-step t_x in T, the networks $N_h(t_x)$ and $N_c(t_x)$ were constructed and each network's clustering coefficient $\bar{C}_{emp}$ (as per the definition of a network's clustering coefficient given in Section 2.2) was calculated[13].

5.3 Comparing model output with empirical data

As discussed in Section 4.2 the scaling factor s allows us to linearly scale the model output to account for the effects of some of assumptions in the model's development[14]. To decide on the values of s for N_h and N_c we consider the maximum $\bar{C}_{emp}$ values for these empirical networks, as well as as the $\bar{C}_{sim}$ values from the networks simulated in the model output. We find that the maximum values for $\bar{C}_{emp}(N_h)$ and $\bar{C}_{emp}(N_c)$ are 0.5 and 0.35 respectively and the maximum value for $\bar{C}_{sim}$ is 1. So, taking the simplest approach, we set the scaling factor of the model output – for comparison with the empirical values from each type of network – to be the ratio of maximum observed corresponding empirical value and the maximum simulated value. Accordingly, for the co-changes network, the scaling factor is $s_h = \frac{0.5}{1} = 0.5$ and for the co-comments network, the scaling factor is $s_c = \frac{0.35}{1} = 0.35$.

With the scaling factors thus set, we next determine the value of p for each type of network that is most appropriate for each type of empirical network N_h and N_c. In this context, "most appropriate" is determined by the value of p that yields the least *mean absolute error* (MAE) when empirical values are compared with the simulated values for a particular network type. By varying p from 0 to 1 (with number of leads = 10, and team size = 4, and the tick distribution across phases for a "typical" project as mentioned in Section 4.1: Inception = 5, Elaboration = 15, Construction = 25, and Transition = 5, in each case), we generated a set of model outputs. Each set corresponded to a particular value of p, and consisted of 50 $\bar{C}_{sim}$ values, one each for each of the 50 ticks of the model's run. For each p, we computed the *error* as the difference between the empirical $\bar{C}_{emp}$ value and the corresponding simulated $\bar{C}_{sim}$ value for each of the 50 ticks. From the error values, the mean absolute error (MAE) is calculated. MAE is a well established measure of accuracy [33] and in this case is calculated by taking the arithmetic mean of the absolute error values of the clustering coefficients across 50 time-steps for each of the co-change and co-comment networks. With reference to Figure 4, we observe that $p = 0.5$ gives the lowest MAE for the co-changes networks, and $p = 0.8$ gives the lowest MAE for the co-comments networks.

Having thus determined the appropriate values of s and p for each of the co-changes and co-comments networks, we now run our model with these values to generate two sets of simulated outputs. Figure 5 presents the empirical and simulated values of the clustering coefficients for the co-changes networks. The mean absolute error (MAE) between the empirical and simulated values is 0.115, the standard deviation of the error (SDE) is 0.133, and the root mean square error (RMSE) is 0.162. Figure 6 presents the empirical and simulated values of the clustering coefficients for the co-comments networks. The MAE between the empirical and simulated values is 0.0725, the standard deviation SDE is 0.093, and the RMSE is 0.104. As these values indicate, the model is able to simulate the variation of the clustering coefficients for *both* the co-change and co-comment networks with reasonable accuracy.

[13]The subscript "emp" in $\bar{C}_{emp}$ indicates that these values of the clustering coefficient are calculated from the empirically generated networks. Correspondingly, in subsequent discussion, $\bar{C}_{sim}$ would denote clustering coefficient values from networks simulated by our model.
[14]The implications of those assumptions are discussed in detail in Section 6

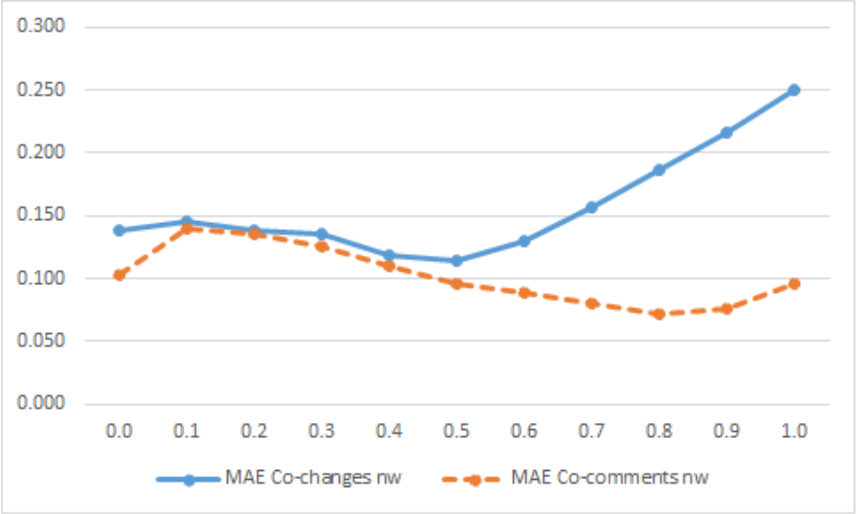

Fig. 4. Mean absolute error (MAE) values (y-axis) between clustering coefficients of co-changes and co-comment networks versus simulated networks, with different values of the parameter p (x-axis).

As our model has stochastic elements, we ran the model with the configurations for the co-changes and co-comments networks 50 times each. The averages of the 50-run model simulations were compared with the corresponding empirical values for the co-changes and co-comments networks. The MAE, SDE, and RMSE for the co-changes networks are 0.114, 0132, and 0.16 respectively. And the MAE, SDE, and RMSE for the co-comments networks are 0.073, 0.093, and 0.104 respectively. These results indicate that simulations from our model closely match the empirical values, even for replicated model runs.

To determine how suitable our model is for predictive purposes, we ran 10-fold cross validation of the of 50-run simulated values for the co-change and co-comment networks, vis-a-vis their corresponding empirical values. Results for the co-changes networks are MAE = 0.065 and RMSE = 0.09 and for the co-comments networks are MAE = 0.075 and RMSE = 0.102. These results demonstrate that the simulated values can be effective for predicting the empirical values.

The accuracy with which our model is able to simulate the empirical characteristic of developer clustering as they interact over two different activities indicate that we have been able identify important drivers of interaction in real world software development ecosystems. Before discussing the insights that can be derived from our results, let us identify some of the threats to their validity.

6 THREATS TO VALIDITY

Threats to **construct validity** relate to the correct measurement of the variables in a study. The general developer interaction network (N) defined in Section 2.1 and its particular instances of co-changes network (N_h) and co-comment network (N_c) are defined in Section 5.2 on the basis of established methods of abstracting developer interactions using a network construct [7], [35], [26]. The metric we use to capture the extent of clustering between developers – network clustering coefficient – is also calculated using a standard formulation [1]. Thus there are no notable threats arising from these factors. Our network construction protocol generated undirected links (edges) between developers, and all

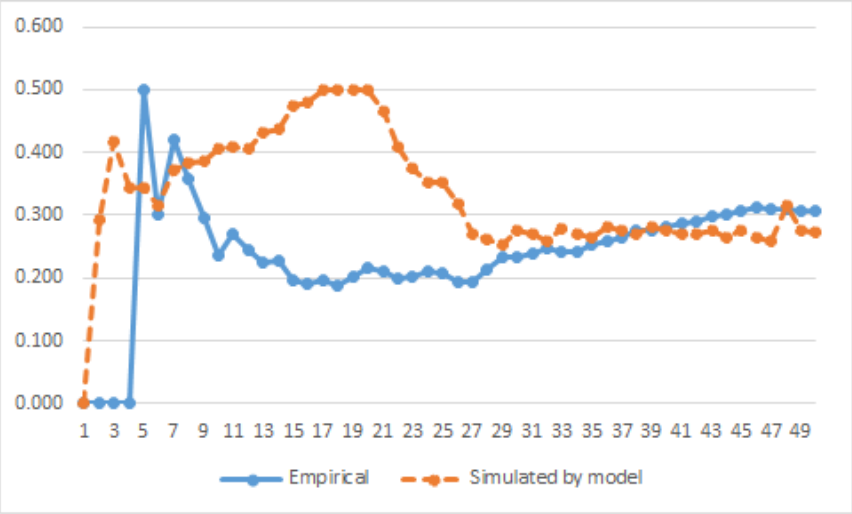

Fig. 5. Variation of clustering coefficient of the co-changes networks over 50 time-steps: Empirical versus simulated.

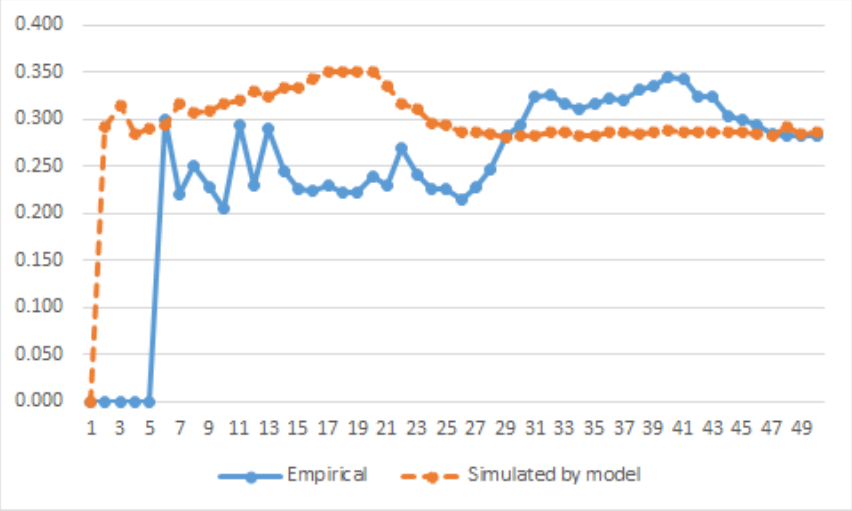

Fig. 6. Variation of clustering coefficient of the co-comments networks over 50 time-steps: Empirical versus simulated.

edges are deemed to have equal weights. If the directionality or weights of edges are considered, using some contextual interpretation, we may have a different set of results.

Threats to **internal validity** come from the presence of systematic errors and biases. As we have used publicly available data for empirically validating our model, the extent of this threat is confined to the veracity of the data source. The paper sharing the curated dataset [17] has been peer reviewed and presented at a premier venue related to mining software repositories; thus we trust this threat to be minimal. Developers in the OpenStack development ecosystem

belong to the open source community, as well as commercial organizations. The dataset used for validation draws from the open source development data available in the public domain. It does not include proprietary information confined within organizational firewalls. We acknowledge this to be a threat to internal validity.

Threats to **external validity** arise from the generalizability of results. Our model output has been validated using two different networks reflecting two distinct development activities of a single development ecosystem. As simulations from the model match the empirical data with notable accuracy for both networks, the relevance of our model beyond a single developer interaction setting can be observed. The dataset we have used for validation is of considerable expanse, and we have carefully selected developers of maximal engagement to construct our empirical networks, as explained in Section 5.2. Thus, we believe our results can offer useful insights on developer interaction in large software development ecosystems. However, since we only use data from one system for validation in this study, we do not claim our results to be generalizable as yet.

Threats to **reliability** are concerned with reproducibility of the results. Given access to the dataset used for validation, our results are fully reproducible. To facilitate replication of our results, we share the code for the NetLogo model [15] and the code used to generate the empirical networks from the dataset [16].

The purpose of developing a model to capture a dynamic system is to be able to isolate essential factor(s) that influence(s) the dynamics of the system, while eliding much of the underlying complexities. As widely recognized, the usefulness of a model comes from how much it is able to explain with limited number of parameters [31]. Accordingly, every model is based on some simplifying assumptions, and our model is no exception. Let us highlight the **implications of those assumptions**. As described in Section 2, for each phase in the SDLC we consider the corresponding developer interaction network to have a known regular structure; for example, a star network during Inception, a fully-connected graph during Elaboration etc. (Figure 1). In reality, network structures during the SDLC phases may differ from these idealized abstractions. Additionally, we assume each module under a lead to have equal number of developers, and each module lead to be interacting with two other module leads during Construction. While each module may not have exactly the same number of developers in real-world scenarios, sub-teams are usually evenly matched in terms of allocated activities, and consequently, the number of deployed personnel. And to facilitate modular design, each module interfaces with few other modules. Thus these assumptions are not wholly incongruent with real world software development. In our model, we have considered the parameter p – reflecting on the likelihood of impromptu interaction between developers – to be constant across all the entire SDLC (Section 4.1). As deadlines draw near in the real-world, developers contact one another with urgency. Thus the value of p can be expected to change over a project's time-line. Finally, we have introduced a constant scaling factor s to linearly scale the output of the model (Section 4.1), whose values have been empirically determined for each network type while validating the model (Section 5.3). The need to introduce a scaling factor arises from the difference in sizes of the networks simulated by the model and developer interaction networks constructed from real-world data. The model is designed to be run for quick exploration of various outcomes on the basis of changing parameters. This requires team sizes to be small compared to real-world teams, so that model runs can complete within reasonable time.

We observe that even in spite of these simplifying assumptions, model output matches empirical data. Let us examine the implications of this observation in more detail.

[15] http://bit.ly/hybrid-model
[16] https://bitbucket.org/subhajit_datta/gonzalez-barahona2015a-openstack/

7 DISCUSSION

In the preceding sections, we have presented the context, development, and validation of our model, followed by an examination of the threats to the validity of our results. With this background, we will now discuss the insights that can be derived from this study, the utility of our results, and plans of future work.

7.1 Insights

With reference to Figure 3 we reiterate that the characteristic of clustering coefficient variation over time is influenced by the value of p. In a development activity with $p = 0$, all developer interaction strictly follows established team communication structure (perhaps by managerial dictum), whereas, $p = 1$ indicates that impromptu interaction between developers is predominant. We expect p to lie somewhere in-between these extreme values for real-world scenarios.

As illustrated in Section 5.3, we found that the best fit between empirical and simulated values of the clustering coefficients for the co-changes networks (N_h) occurs at $p = 0.5$, and for co-comments networks (N_c) occurs at $p = 0.8$. Thus, the extent of impromptu interactions between developers in the co-changes networks is lower than that in the co-comments networks. This offers an interesting perspective on how developers connect over different development activities in the general context of “talk" versus “work" interaction in large scale software development [11]. Code change invariably involves ensuring that the changed code continues to function as it did before with other code units it is exchanging information with. This calls for interfacing with owners of other code units. Effective software design principles prescribe that a particular unit of code interacts with limited number of other such units, to minimize ripple effects of change, among other considerations [4]. Thus, while changing units of code, developers are usually working on well defined tasks, and interacting with a limited number of other developers each owning related units of code. On the other hand, when developers discuss matters related to problem ticket resolution with their peers, the objective is to gather and share information as widely as possible. A wide arc of interaction in this context facilitates quick understanding of the reported problem, and reaching a consensus on the most effective fix. Accordingly, we see a lesser propensity of impromptu interactions ($p = 0.5$) while developers co-change code, when compared with the same developers co-commenting on problem tickets ($p = 0.8$). The best fit model parameters thus effectively reflect on the difference between the dynamics underlying these two distinct activities developers interact over.

From Figures 5 and 6, it is evident that the level of congruence between model output and empirical data is not the same across all the phases of the SDLC. To examine this situation more closely, we calculated the MAE by the phases for each of the co-changes and co-comments networks. For the co-changes network, the MAE values for Inception, Elaboration, Construction, and Transition are 0.241, 0.189, 0.061, and 0.034 respectively. For the co-comments network, the MAE values for these respective phases are 0.236, 0.087, 0.044, and 0.005. As we observe, for both the networks, the MAE is reduced over the progression of the phases. Our model is able to closely simulate the level of developer clustering during Construction and Transition, and to lesser extents for Inception and Elaboration. As the parameter p is held constant across all the phases during the model run, the difference in the MAE values can be attributed to the underlying network structures we have assumed for each phase (Section 2 and, specifically Figure 1). During Inception we modelled the network as a star, and during Elaboration it is modelled as a fully connected graph. Both these network configurations are seen to yield clustering coefficients in the model simulations which are higher than the empirical ones during these two phases (Figures 5 and 6). Thus, for the real-world system we have studied, developers interact to a lesser extent during Inception and Elaboration vis-a-vis what the model estimates. However, as evident from the respective MAE values, model simulations match empirical data very closely during Construction and Transition. We

may highlight that these two phases are the most crucial in terms of execution and delivery in a software development project. Our model is thus able to closely capture the dynamics of developer interaction in these critical phases.

Observing the varying accuracy of the model's outputs over the phases offers insights into the dynamics of developer interaction across the SDLC. Evidently, developer interaction during the first two phases conforms to the assumed network structures to a lesser extents than the latter two phases. During earlier part of the development life cycle, the "cone of uncertainty" is wide, and it narrows as a project moves closer to completion [27]. With higher uncertainty, developer interactions are more unpredictable. As project deliverables, deadlines, and customer expectations become more streamlined over time, developer interactions align more closely with expected structures. This has implications for project governance. In development scenarios with low tolerance for unstructured developer interactions, care needs to be taken to reduce uncertainty early on in the development life cycle. There will be some projects where such uncertainty is inevitable; due to changing requirements, shifting business needs, or evolving operating environments. In such situations, appropriate tools and process need to be pro-actively devised and deployed to help developers find the answers they need, with minimal communication overheads.

7.2 Utility of the results

- A deep understanding of the complex chains of causes and effects that permeate software development ecosystems, is essential at the individual, team, and organizational levels. However, controlled experiments – the gold standard for connecting cause with effect – are not feasible to design and perform in real-world software development. An useful alternative is to explore what-if scenarios by modifying parameters that influence outcomes of interest, and then matching the various scenarios with real-world data. Our model offers a simple thought construct for such investigations, as well as an executable tool that can be run with various parameter combinations.
- As illustrated in Figure 3, different values of the parameter p illuminate *regimes* of developer interaction, as they are differentiated by variations in clustering. For relatively low values of p, the clustering coefficient is seen to oscillate more, while for higher values, it attains its peak and then settles down into a stable value. Evidently, each regime has its own needs for infrastructural support to facilitate developer interaction. By elucidating the effects of different levels of impromptu connections – at different values of p – on developer clustering, our model offers informed choices for project managers to support interaction needs of their team members at specific junctures of the SDLC.
- As observed in the empirical data, and simulated by our model, levels of developer clustering is not uniform across the SDLC. There are inflection points in the clustering coefficient curve where the slope changes, often drastically. Unless the team members are aware of such inflection points, and the team management is sensitive to their occurrence and timing, there are risks of communication breakdowns with their concomitant quality concerns [5]. By facilitating a priori identification of such inflection points in the variation of developer clustering, our model can facilitate suitable interventions.
- In addition to the results themselves, the general approach of this study makes a methodological contribution to the examination of developer interaction in large scale software development. To the best of our knowledge, this is one of the earliest applications of agent-based modelling in the study of software development ecosystems. Our results promise exciting new possibilities in future applications of this modelling paradigm in this area.

7.3 Plans of future work

In our future work, we intend to further validate the model using additional datasets to help establish whether the simulated variation of developer clustering indeed reflects general characteristics of large scale software ecosystems. Expanding the scope of validation will also allow us to examine the effects of relaxing some of the limitation and assumptions identified in Section 6. Additionally, it can aid in determining whether there is a universal scaling factor connecting the assumed team size in the mode and the one in the empirical systems used for validation. This will also help us examine how team size relates to clustering coefficient in the software development context. Finally, we intend to leverage the insights from this study towards developing an ensemble of models that can address developer interactions over the range of activities they engage in – designing, coding, testing, reviewing etc. – in a typical development ecosystem.

8 SUMMARY AND CONCLUSIONS

In this paper, we present a simple model for developer interaction across the phases of the software development life cycle. A key parameter of the model is the probability of impromptu connections between developers, which can go beyond established communication channels. With a view to understanding the temporal variation of clustering between developers, we examine the characteristics of the clustering coefficients simulated by running the model with different parameters, vis-a-vis those calculated from empirical interaction networks between developers of a large scale real world system, who are working together on code changes and discussion of problem tickets. We observe that – in spite of the simplicity of the model – simulated and empirical results for both networks show notable congruence.

Findings from this study underscore the importance of changing interaction structure and the propensity of impromptu connections as key influences on developer clustering. Clustering is of much consequence at individual and team levels. An enduring concern of software project management is to orient governance methods towards effective use of such clustering. By highlighting the varying nature of developer clustering during a project's lifetime, our model can inform interventions at specific junctures to help developers interact more easily. Such interventions may involve co-locating team members, distributing work along locations, fostering a culture where team members are encouraged to connect – on an impromptu basis – beyond their immediate work groups.

Almost all major software systems are now developed by large and distributed teams. By identifying the changing patterns of interactions between developers over different phases of development, and the drivers of such change, our model can help address some of complexities of large scale software development.

REFERENCES

[1] Albert, R., and Barabasi, A. Statistical mechanics of complex networks. *cond-mat/0106096* (June 2001). Reviews of Modern Physics 74, 47 (2002).

[2] Booch, G. Tribal memory. *IEEE Software 25*, 2 (2008), 16–17.

[3] Brooks, F. P. *The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition.* Addison-Wesley, 1995.

[4] Brooks, F. P. *The design of design: essays from a computer scientist.* Addison-Wesley, Upper Saddle River, NJ, 2010.

[5] Cataldo, M., and Herbsleb, J. D. Communication networks in geographically distributed software development. In *Proceedings of the 2008 ACM conference on Computer supported cooperative work* (New York, NY, USA, 2008), CSCW '08, ACM, p. 579â588.

[6] Cataldo, M., Herbsleb, J. D., and Carley, K. M. Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement* (Kaiserslautern, Germany, 2008), ACM, pp. 2–11.

[7] Cataldo, M., Wagstrom, P. A., Herbsleb, J. D., and Carley, K. M. Identification of coordination requirements: implications for the design of collaboration and awareness tools. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work* (New York, NY, USA, 2006), CSCW '06, ACM, p. 353â362.

[8] Conway, M. How do committees invent? *Datamation Journal* (April 1968), 28–31.

[9] Datta, S. *Software Engineering: Concepts and Applications.* Oxford University Press, 2010.

[10] Datta, S. How does developer interaction relate to software quality? an examination of product development data. *Empirical Software Engineering 23*, 3 (June 2018), 1153–1187.

[11] Datta, S., Sindhgatta, R., and Sengupta, B. Talk versus work: characteristics of developer collaboration on the jazz platform. In *Proceedings of the ACM international conference on Object oriented programming systems languages and applications* (New York, NY, USA, 2012), OOPSLA '12, ACM, pp. 655–668.

[12] Davis, J. P., Eisenhardt, K. M., and Bingham, C. B. Developing theory through simulation methods. *The Academy of Management Review 32*, 2 (2007), 480–499.

[13] DeMarco, T., and Lister, T. *Peopleware: Productive Projects and Teams.* Dorset House Pub. Co., 1987.

[14] Ehrlich, K., and Cataldo, M. All-for-one and one-for-all?: a multi-level analysis of communication patterns and individual performance in geographically distributed software development. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work* (New York, NY, USA, 2012), CSCW '12, ACM, pp. 945–954.

[15] Fan, X., and Yen, J. Modeling and simulating human teamwork behaviors using intelligent agents. *Physics of Life Reviews 1*, 3 (Dec. 2004), 173–201.

[16] Fortunato, S., Bergstrom, C. T., Borner, K., Evans, J. A., Helbing, D., Milojevic, S., Petersen, A. M., Radicchi, F., Sinatra, R., Uzzi, B., Vespignani, A., Waltman, L., Wang, D., and Barabasi, A.-L. Science of science. *Science 359*, 6379 (Mar. 2018), eaao0185.

[17] Gonzalez-Barahona, J. M., Robles, G., and Izquierdo-Cortazar, D. The metricsgrimoire database collection. In *Proceedings of the 12th Working Conference on Mining Software Repositories* (Piscataway, NJ, USA, 2015), MSR '15, IEEE Press, pp. 478–481.

[18] Guimera, R., Uzzi, B., Spiro, J., and Amaral, L. A. N. Team assembly mechanisms determine collaboration network structure and team performance. *Science (New York, N.Y.) 308*, 5722 (Apr. 2005), 697–702. PMID: 15860629.

[19] Herbsleb, J. D., and Mockus, A. An empirical study of speed and communication in globally distributed software development. *IEEE Trans. Softw. Eng. 29* (June 2003), 481–494.

[20] Hsu, S.-C., Weng, K.-W., Cui, Q., and Rand, W. Understanding the complexity of project team member selection through agent-based modeling. *International Journal of Project Management 34*, 1 (Jan. 2016), 82–93.

[21] Jacobson, I., Booch, G., and Rumbaugh, J. *The Unified Software Development Process.* Addison-Wesley, 1999.

[22] Jennings, N. R., and Bussmann, S. Agent-based control systems: Why are they suited to engineering complex systems? *IEEE Control Systems Magazine 23*, 3 (June 2003), 61–73.

[23] Kaihara, T. Multi-agent based supply chain modelling with dynamic environment. *International Journal of Production Economics 85*, 2 (Aug. 2003), 263–269.

[24] Kotak, D., Wu, S., Fleetwood, M., and Tamoto, H. Agent-based holonic design and operations environment for distributed manufacturing. *Computers in Industry 52*, 2 (Oct. 2003), 95–108.

[25] Kwan, I., Cataldo, M., and Damian, D. Conway's law revisited: The evidence for a task-based perspective. *IEEE Software 29*, 1 (Feb. 2012), 90 –93.

[26] Kwan, I., Schroter, A., and Damian, D. Does Socio-Technical congruence have an effect on software build success? a study of coordination in a software project. *IEEE Transactions on Software Engineering 37*, 3 (May 2011), 307–324.

[27] McConnell, S. *Software Estimation: Demystifying the Black Art.* Microsoft Press, 2006.

[28] Newman, M. E. J. Scientific collaboration networks. i. network construction and fundamental results. *Physical Review E 64*, 1 (2001), 016131.

[29] Newman, M. E. J. The structure and function of complex networks. *cond-mat/0303516* (Mar. 2003). SIAM Review 45, 167-256 (2003).

[30] Niazi, M., and Hussain, A. Agent-based computing from multi-agent systems to agent-based models: a visual survey. *Scientometrics 89*, 2 (Aug. 2011), 479.

[31] Page, S. E. *The Model Thinker.* Basic Books, New York, Nov. 2018.

[32] Schelling, T. Dynamic models of segregation. *Journal of Mathematical Sociology 1* (1971).

[33] Tabachnick, B., and Fidell, L. *Using Multivariate Statistics.* Boston: Pearson Education, 2007.

[34] Wagstrom, P., and Datta, S. Does latitude hurt while longitude kills? geographical and temporal separation in a large scale software development project. In *Proceedings of the 36th International Conference on Software Engineering* (New York, NY, USA, 2014), ICSE 2014, ACM, pp. 199–210.

[35] Wagstrom, P., Herbsleb, J., and Carley, K. Communication, team performance, and the individual: Bridging technical dependencies. Academy of Management Conference, 2010.

[36] Watts, D. Networks, dynamics, and the Small-World phenomenon. *The American Journal of Sociology 105*, 2 (1999), 527, 493.

[37] Watts, D. J., and Strogatz, S. H. Collective dynamics of /'small-world/' networks. *Nature 393*, 6684 (June 1998), 440–442.

[38] Wilensky, U., and Rand, W. *An Introduction to Agent-Based Modeling: Modeling Natural, Social, and Engineered Complex Systems with NetLogo.* Mit Press, 2015.