Supporting Consensus-based Software Development: a Vision Paper

Mathieu Lavallée
Independent consultant
math.lavallee@gmail.com

Guillaume Beaulieu
Université du Québec à Montréal
beaulieu.guillaume.2@courrier.uqam.ca

Michalis Famelis
Université de Montréal
famelis@iro.umontreal.ca

ABSTRACT

Traditional, vertical organizational models of software development have been challenged by more agile and collaborative structures. Recently, this has also been demonstrated in the emergence of explicitly horizontalist organizational structures, focused on consensus-based decision making. In this paper, we describe the principles and processes of these "Consensus-Based Communities" (CBCs) and outline the main challenges they face as they try to implement "Consensus-Based Software Development" (CBSD). We express these as early, high level requirements for a tool supported methodology. Based on these, we present and analysis of existing tools that shows that no single tool provides complete support for consensus-based group decision making. We thus outline directions for future research, identifying opportunities for the development and deployement of model-based techniques in this emergent field.

1 INTRODUCTION

Software development is a team activity [25, 53], in which internal team dynamics impact the quality of the produced software. A classic formulation of this is "Conway's Law":

> Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure. [16]

More recently, Nagappan et al. showed that organizational structure was a better predictor of fault-proneness than traditional technical metrics [44]. A defining aspect of a team's organization and communication structure is *Group Decision Making* (GDM), also known as *Multi-Person Decision-Making* (MPDM), [22, 40, 45]. Traditionally, organization structures are hierarchical and command based. However, the advent of Agile software development has changed GDM processes, as it tends to prefer horizontal structures [52]. Characteristically, the Agile Manifesto declares:

> The best architectures, requirements, and designs emerge from self-organizing teams. [3]

In practice, the degree of self-organization varies widely across different organizations. We illustrate this diversity in Figure 1, as a continuum of organizational horizontality in software development teams. On the one end there is traditional "command-based" development, where upper levels of the organizational hierarchy have the final say on what developers do. Moving from that extreme, some organizations maintain the use of project managers who have the final say in decisions while encouraging the building

COMMitMDE 2017, Austin, Texas, USA
2017. 123-4567-24-567/08/06…$15.00
DOI: DOIDOIDOIDOI

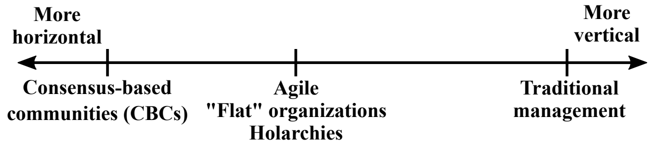
More horizontal
More vertical
Consensus-based communities (CBCs)
Agile "Flat" organizations Holarchies
Traditional management

Figure 1: The continuum from vertical (i.e. hierarchical) to horizontal (i.e. non-hierarchical) organizations.

of a team consensus. Some organizations actively pursue a "flat" model [48] or an "holarchy" [49], where decisions are delegated to self-managing teams. Often, organizations using self-managed teams still maintain a higher authority, simply reducing the amount of middle management. In Figure 1 we position Agile teams, that promote self-organized GDM, in the middle of the continuum, along with flat and holarchic organizations.

In this paper, we put the spotlight on the other end of the continuum. This is populated by software development organizations that intentionally adopt explicitly anti-hierarchical and consensus-based modes of GDM and collaboration. For example, Koumbit, a web-hosting company based in Montreal, describes its internal processes as "non-hierarchical self-management" [34]. Loomio is a developer-owned cooperative based in New Zealand that puts consent at the centre of its decision making process [38]. Other organizations like Zappos [59] and Medium.com [20] maintain horizontal principles. This is also often the case for open source communities, such as Drupal, which favours a self-managed approach [21]. We call such organizations *Consensus-Based Communities* (CBCs).

CBCs face a set of unique challenges when developing software, mostly based on their philosophical approach to GDM. The Agile Manifesto already pushed GDM away from the "Benevolent Dictator" management paradigm.

> Agile Software Development has completely dispensed with the formal job title of the project manager. Agile methodologies such as scrum seem to distribute the erstwhile responsibilities of the project manager into new roles. [52]

CBCs affirm that team decisions must go a step further, and ensure consensus building as much as possible, instead of majority decision-making. The rationale behind striving for consensus is based on a perceived flaw of democratic decision-making called "tyranny of the majority":

> The will of the majority may be seen as the will of the whole group, with the minority expected to accept and carry out the decision, even if it is against their deeply held convictions and most basic needs. It is possible for a voting group to

> look for solutions that would suit everybody, but it is more common for ideas with a majority backing to be pushed through. [50, p. 9]

This flaw can cause, for example a team with a majority of non-experts to overrule a minority of experts, leading to poor decisions. Instead, consensus-building requires team members to convince all their colleagues, instead of unilaterally imposing their propositions. This means that software developers in CBCs cannot exploit the process in order to shut down their colleagues, but must provide facts and data able to persuade them to change their mind.

CBCs such as the ones mentioned above also operate under anti-hierarchical principles. Their perception is that software development has become so complex that not one person can see the whole picture. Each software developer is therefore a piece of the puzzle, each with their own expertise, and decisions should then be made with everyone in mind [45].

Even though CBCs are currently not the norm in software engineering, studying them can provide interesting insights on team dynamics, and can help identify beneficial organizational practices across the continuum of organizational horizontality. Broadly speaking, the recent move away from traditional organizational practices can be understood as a concern about the degree to which they are appropriate for software development. Thus, interesting insights can be obtained from radically different organizational structures [48]. At the same time, since CBCs are software producing organizations in an era of software ecosystems [41], understanding the impact of their practices on software quality (cf. Conway's Law) may have ramifications beyond their niche.

In this paper, we focus on CBCs and propose a vision for a comprehensive, model-based, tool supported methodology for *Consensus-Based Software Development* (CBSD) that addresses their unique set of challenges. We identify the specific needs of CBCs and study existing collaboration tools to find gaps between the two and potential avenues of research and development. Specifically, we make the following contributions:

a) We highlight the existence of CBCs and describe their unique processes and challenges;
b) We describe how consensus-based GDM can be supported for software development;
c) We analyze the capabilities of existing tools, identify their limitations, and outline how model-based techniques can be used to improve the state of CBSD practise.

The rest of the paper is organized as follows: We present background information on CBCs in Section 2, detailing their typical internal processes In Section 3, we explain the challenges raised by consensus-based GDM and in Section 4 we outline the requirements for a tool framework for supporting consensus-based software development. We analyze the capabilities of existing GDM support tools in Section 5 and point out areas for improvement in Section 6. We discuss related work in Section 7 and conclude in Section 8.

2 BACKGROUND

To better understand the unique challenges in CBSD, we first outline the main characteristics of CBCs.

Consensus-based decision-making does not mean that everybody must agree on everything. In practice, consensus is typically "soft", which means that certain forms of disagreement are acceptable. According to the Consensus Handbook [50, p. 6]:

> Instead of simply voting for an item and having the majority of the group get their way, a consensus group is committed to finding solutions that everyone actively supports, or at least can live with.

The aim is to avoid a situation where there are winners and losers. CBCs claim that this helps everybody feel involved in the organization, and helps them learn new skills and be better, more complete software developers [34]. This further increases the sense of agency and ownership towards decisions, since each individual actively participates, ensuring that everyone's concerns are part of the process. A major benefit of consensus decision-making is therefore the increased solidarity among team members in cases where bad decisions are made. Faced with a negative result, since the whole team was engaged in the decision together, no single individual or group can be singled out for blame [60].

In the following, we describe a typical consensus-based GDM process, in terms of the activities performed and the roles and responsibilities associated. Consensus-based decision-making processes are adapted from the different kinds of democratic practices surrounding them [17]. They are adapted to the size and ability of the group.

While the exact process varies from one organization to another, we present here a basic model of typical *activities* in consensus-based GDM [50, p. 16]. These are:

Introduction (A1): Introduction of the issue on which a decision must be made;
Discussion (A2): Discuss the issue and collect ideas;
Concerns (A3): List concerns regarding the proposal;
Proposal (A4): Identify emerging proposals meeting concerns;
Refinement (A5): Discuss, clarify and amend the proposals;
Consensus test (A6): Test for agreement, usually through a straw poll. If not, return to the previous points;
Implementation (A7): Implement the decision.

We show these activities diagrammatically in Figure 2.

During a test for consensus (activity A6), a typical CBC uses four types of *votes* for the straw poll [50, p. 16] with the following meaning:

Block (V1): The participant thinks the proposal is fundamentally wrong and should not be implemented by the organization.
Stand aside (V2): The participant cannot support the proposal and is not willing to implement it. However they are willing to let the organization implement it without them.
Reservation (V3): The participant has some reservations, but is willing to let the proposal be implemented and to work for its implementation.
Agreement (V4): The participant supports the proposal and is willing to implement it.

The responsibilities associated with the good functioning of the team are traditionally assigned to a team manager. Instead, in self-managing teams, they must be delegated to team members. Consensus-based teams recommend to split responsibilities to different people to avoid a concentration of powers which would re-create a hierarchical structure [56]. Specific *roles* include [50, 56]:

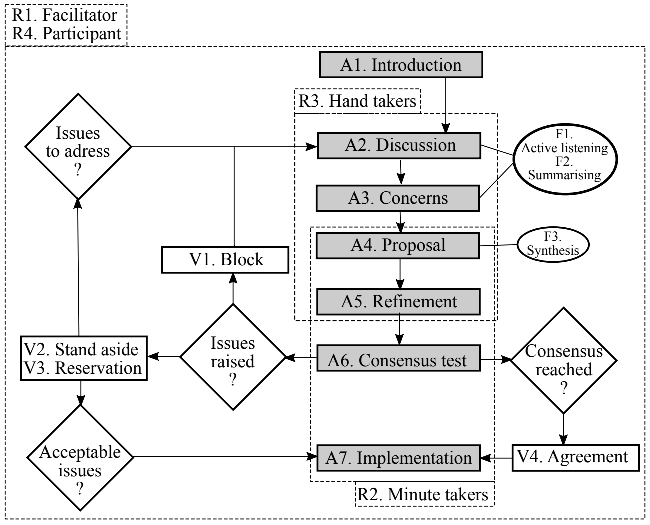

Figure 2: Consensus decision making model.

Facilitators (R1): A neutral party whose responsibility is to keep the meeting on topic and assist the decision-making process. In some cases, no single person is assigned the role, instead relying on the participants, assuming that they have internalized it enough to self-discipline.

Minute Takers (R2): Responsible for keeping track of the decisions and action items.

Hand Takers (R3): Responsible of keeping track of whose turn it is to speak next and control the time allotted to each speaker.

Participant (R4): Each team member is responsible for self-discipline and for the good functioning of the process.

Facilitation is a complex role, which can be broken down into the following *responsibilities* [50, p. 31]:

Active listening (F1): To assist speakers in expressing their ideas and its associated context.

Summarising (F2): To reformulate the intervention of the speaker in a clear and concise manner in order to confirm its interpretation.

Synthesis (F3): To merge the ideas in actual proposals potentially acceptable for all participants.

The list of work products of a consensus-based process includes the minutes of the meeting, along with the decisions and action items. It is recommended that action items be tracked independently from the rest in order to avoid losing track of them and to prevent power concentration into the same hands [56].

3 CHALLENGES OF CONSENSUS-BASED GDM

Software development teams that adopt a consensus-based GDM process face a unique set of challenges. These stem from the need to work and reach consensus. Degenerations of the process can introduce problems such as false consensus, where an explicit agreement is reached, but where team members implicitly disagree. We group such challenges along seven different dimensions.

One dimension (D1) is related to the heterogeneity of communication [45]. For example, software engineering teams include a variety of experts of different fields, depending on the needs of the project. These different experts can be proficient in different languages and techniques. For example, one expert can be well-versed with the UML language while the rest of the team is not. This creates a hurdle for communication as the expert struggles to explain his point to non-experts.

Another dimension (D2) is related to the need to persuade others in order to obtain consensus. Ideally, this persuasion is based on real facts, brought by real experts, but in practice, emotional factors can enter the equation. For example, "people are easily persuaded by other people that they like" ([13], quoted by [11]). Another example, "consistently slandering someone behind his back" [56, p. 33] is another tactic to persuade others using unsavory means.

This leads to the third dimension (D3) of how to adequatly generate alternatives for a complex issue. Ideally, GDM follows a rational decision-making approach. This approach is akin to a breadth-first approach, where all alternatives are studied and compared. The opposite is the naturalistic decision-making approach. This approach is akin of a depth-first approach, where one easy to find solution is studied in-depth [42]. For example, a typical software development team mixes the two approaches, going in-depth with a small number of alternatives [58]. Generally this is because the complexity of software development context makes a breadth-first approach impossible and requires limiting the breadth to what seems possible at the time [42, 58].

Generating and choosing the proper alternatives can be difficult, especially with dysfunctional teams (D4). Issues of "groupthink", where there is a reluctance of criticizing ideas, are well documented [55]. Instances of polarized factions constantly blocking the propositions of the others are also a symptom of a dysfunctional team [32, 55]. Even before such problems arise, social stratification due to factors such as racism, sexism and class, generally makes marginalized people take less share in the decision-making process. Worse, that phenomenon is often justified by those who are silencing themselves as they perceives themselves as less competent [6]. Creating a context for genuine participation is an ongoing process that requires patience and dedication.

The fifth dimension is related to the nature of anti-hierarchical organizations (D5). Without a manager, the developers themselves must assume responsibility of the task typically assigned to managers. These include managing conflicting priorities between stakeholders, assuring that action items are actually implemented, and giving more weight to the intervention of experts [22, 48].

Another challenge is related to documenting the rationale of the decisions made (D6). As Shahin et al. [51] writes:

> In the field of software architecture, there has been a paradigm shift from describing the outcome of architecting process mostly described by component and connector (know-what) to documenting architectural design decisions and their rationale (know-how) which leads to the production of an architecture. [51]

A GDM support tool should therefore document decision rationale to ensure that future developers can understand what arguments

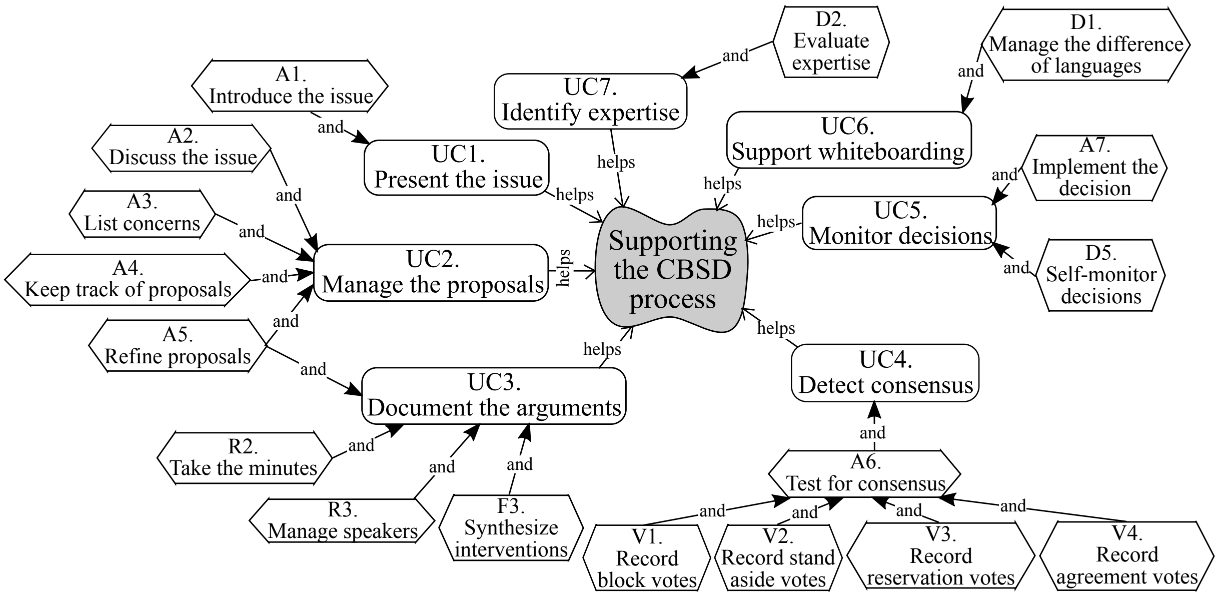
D2. Evaluate expertise
D1. Manage the difference of languages
A1. Introduce the issue
UC7. Identify expertise
UC6. Support whiteboarding
A2. Discuss the issue
UC1. Present the issue
A7. Implement the decision
A3. List concerns
UC5. Monitor decisions
A4. Keep track of proposals
UC2. Manage the proposals
Supporting the CBSD process
D5. Self-monitor decisions
A5. Refine proposals
UC3. Document the arguments
UC4. Detect consensus
R2. Take the minutes
A6. Test for consensus
R3. Manage speakers
F3. Synthesize interventions
V1. Record block votes
V2. Record stand aside votes
V3. Record reservation votes
V4. Record agreement votes
and
helps

Figure 3: Use-cases to support for the CBSD process, using the i* modeling language.

supported the decision. Ideally, such a tool would be able to see how decisions evolve [54]. For example, developers should know that a decision can be changed if the underlaying arguments are no longer valid.

But the technical context of the issue is not the only context to manage. Since CBCs are not the norm, the GDM process is generally a two-way educational process, first teaching newcomers about the GDM process, but also teaching ways to address concerns in a constructive manner. With consensus-based GDM, communities transition from a discursive space in which people can be either right or wrong to one where people are either more or less aware of the issues surrounding its practices. To make things more complicated, as is the case for any organization, the typical process is not followed to the letter, but changes are actually frequent: Since there are no leader to impose a process, any meeting can result in a process change. Hence, the bulk of the decision-making process is centred around educating participants on the current organizational context. Documentation of such processes is contextual and reflects a community's ongoing organizational dynamics. Therefore, the given organizational context of the CBC (D7) will have an impact on its GDM, more so than in a traditional contexts using common GDM practices. The rationale of the decision (D6) cannot be understood outside this constantly evolving organizational context of the CBC (D7).

4 REQUIREMENTS FOR CBSD SUPPORT TOOLS

We identify the high-level requirements for CBSD tools, based on the processes, activities, roles, and artifacts involved in CBSD described in Section 2, as well as the challenges outlined in Section 3. We present the requirements in the form of use-cases, listed below. We also give the requirements as goal models in Figures 3 and 4 using i*, a language for modelling early requirements [18, 57].

UC1: During the introduction of an issue, present its context in a concise yet understandable way (A1).

UC2: Collect the ideas brought during the meeting (A2), obtain concerns from the participants and keep track of them (A3). Turn ideas into actionable proposals that everyone can inspect and amend (A4). Record amendments to proposals (A5).

UC3: Document the arguments pertaining to each proposal (A5) and the identity of the person(s) who brought the argument forward (R3). While it would be difficult to support active listening (F1) and the reformulation of participants' intervention (F2), the tool should assist the Facilitator's synthesis responsibility (F3). Each decision should include the minutes of the relevant meetings in order to understand better the context of the decision (R2).

UC4: Detect consensus, through mechanisms like straw polls (A6). Straw polls should support the four types of votes (V1-V4). It should be possible to record the rationale of non-consenting votes (V1-V3).

UC5: Monitor the implementation of decisions after the meeting, to ensure it is carried out as decided (A7, D5).

UC6: Record relevant sketches (e.g., whiteboard photos). Discussion can often lead to a refined model that can be understood by all participants (D1).

UC7: Identify and evaluate the expertise relevant to the decision at hand (D2).

Introspection, reflection and self-improvement is a big part of the Agile philosophy [3], and the philosophy of CBCs. A CBSD support tool should also therefore support these use-cases:

UC8: After implementing a decision, determine if the alternatives studied were adequate or if the team should have spent more time finding alternatives (D3).

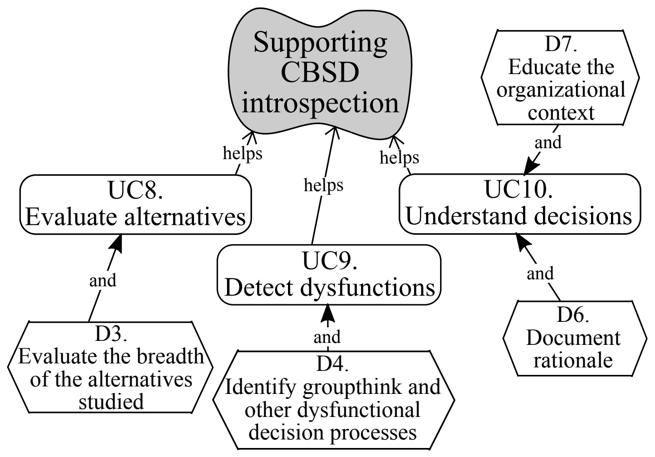
Supporting CBSD introspection
D7. Educate the organizational context
and
helps
helps
helps
UC8. Evaluate alternatives
UC10. Understand decisions
UC9. Detect dysfunctions
and
and
and
D6. Document rationale
D3. Evaluate the breadth of the alternatives studied
D4. Identify groupthink and other dysfunctional decision processes

Figure 4: Use-cases to support for CBSD introspection, using the i* modeling language.

UC9: Detect degenerated group decision making problems, such as groupthink, unresolved conflict, etc. (D4).

UC10: Some decisions can have a long-term impact. Developers years later might have to deal with the decisions made today and might want to understand why the specific decision was made (D6). Understanding the decision means understanding the organizational context wherein the decision was made at the time. Participants must therefore be educated on the organizational context in order to understand the related decisions (D7).

5 WHAT IS CURRENTLY MISSING IN CBSD SUPPORT TOOLS?

In this section we describe how existing tools and technologies fit with the CBSD activities, roles and work products described in previous sections. Ideally, CBSD support tools should be compatible with the processes presented earlier, and address the challenges of consensus-based decision-making. In practice, while some are covered by existing tools, other functionalities are still missing. The list of tools analyzed is a preliminary, non-exhaustive list of tools, based on a similar list published in the literature [26] as well as our own investigation.

ArguNet [4] and Carneades [28] are tools for structuring debates into arguments and logical propositions. They can record the ideas and arguments presented during a GDM meeting, using formal logic to encode their structure. They treat arguments as logically true propositions which can either support or oppose a proposal. They do not have voting mechanisms or support for evaluating expertise.

bCisive [46] is a tool for structuring debates. It provides a richer modelling language than ArguNet and Carneades. For example, it allows typing the sources of arguments, e.g., to differentiate between expert opinion and hearsay.

DebateGraph [2] is a mind-mapping tool offering the capability to structure data. It can link arguments and proposals, but its utility remains limited for GDM.

Loomio [37] is a GDM tool useful for voting when there are already clearly-defined alternatives. It does not help in finding and structuring the debate in order to find these alternatives.

Planning Poker [43] is useful for choosing an appropriate priority or estimate. Alternatives are clearly stated in the form of numerical scales. The facilitator can choose among various scales (such as the Fibonacci sequence, powers of 2, Small-Medium-Large, etc.) according to what is most appropriate for a given context.

Reddit [47] and Discourse [14] are discussion tools, where participants bring arguments based on an introduction statement. Reddit integrates a voting mechanism which ensures that entries supported by more participants are ranked first. Discourse is closer to a mailing list and presents arguments in a temporal sequence.

holaSpirit [29] is an implementation of the "holacracy" [49] framework, which focuses on managing responsibilities and meeting agendas. It includes a way to document tensions between participants, which can help diagnose some dysfunctionalities. It can help identify expertise, based on the assignment of responsibilities.

We summarize the results of our analysis in Table 1. Note that the activity "Refinement" (A5) appears in two columns and does not present the same results. The reason is that one analysis is taken from the point of view of management proposals (UC2) while the other is based on argument documentation (UC3). For example, a tool can permit edition of proposals (A5 from UC2), but does not cover argument documentationm (A5 from UC3), therefore resulting in different results.

Studying the results presented in Table 1, it is possible to pinpoint incompatibilities between current tool functionalities and the needs of CBSD. Some requirements are poorly covered by current tools, while others are not covered at all.

Results also show that many tools focus on one aspect of GDM exclusively. This is coherent with the current tendency away from a single tool able to do everything (e.g., Rational Rose) and toward multiple inter-communicating tools (e.g., integrating Slack with JIRA and Bitbucket). It would therefore theoretically be possible to cover the needs of CBSD using multiple inter-communicating tools. Unfortunately, most of these tools cannot do this right now, forcing a manual transfer of data from one tool to the next. For example, while none of the tools analyzed provides direct support for decision monitoring (UC5), it would be possible to support it through a connection to a project management tool like Microsoft Project or Atlassian JIRA.

However, one obstacle to this integration is the two widely different paradigm used by the tools analyzed. About half the tools studied support a rational decision process (e.g. ArguNet, bCisive, Carneades), rooted in logical propositions akin to Bayesian logic. These tools assume that arguments brought forward are equally true and build the GDM around argument structures. The other half support a naturalistic decision process (e.g. Loomio, Reddit, Planning Poker) and are rooted in fuzzy propositions. These fuzzy propositions are not assumed to be true, and GDM is therefore built around participants' votes.

The importance of both rational and naturalistic approaches to GDM has been detailed in published literature [42, 58]. On the one hand, purely rational tools are limited in practice. On the other hand, while some fuzzy argumentation models exist, they focus on artificial intelligence and are not supported by any GDM tools

Table 1: Results of the tool analysis based on the i* models in Figure 3 and 4. Tools are presented in alphabetical order.

Tool			Argunet	bCisive	Carneades	DebateGraph	Discourse	holaSpirit	Loomio	Planning Poker	Reddit
UC1: Present issue	A1		Yes	Yes	No	No	Yes	No	Yes	Yes	Yes
UC2: Manage proposals	A2		Yes	Yes	Yes	Yes	Yes	No	No	No	Yes
	A3		Partial	Yes	No	Partial	No	No	No	No	No
	A4		Yes	Yes	Yes	No	No	No	No	No	No
	A5		Partial	Partial	No	No	Partial	No	No	No	Partial
UC3: Document arguments	A5		Yes	Yes	Yes	Yes	Yes	No	No	No	Yes
	R2		No	No	No	No	Yes	No	No	No	Yes
	R3		No	No	No	No	Yes	No	No	No	Yes
	F3		Yes	Yes	Yes	Yes	No	Yes	No	Yes	No
UC4: Detect consensus	A6	V1	No	No	No	No	No	No	Yes	Yes	Yes
		V2	No	No	No	No	No	No	Yes	No	No
		V3	No	No	No	No	No	No	Yes	No	No
		V4	No	No	No	No	No	No	Yes	Yes	Yes
UC5: Monitor decisions	A7		No	No	No	No	No	No	No	Partial	No
	D5		No	No	No	No	No	No	No	Partial	No
UC6: Whiteboarding	D1		No	Yes	No	Yes	Yes	No	No	No	Partial
UC7: Identify expertise	D2		No	Yes	No	No	No	Yes	No	No	No
UC8: Evaluate alternatives	D3		No	No	No	No	No	No	No	No	No
UC9: Detect dysfunctions	D4		No	No	No	No	No	Partial	No	No	No
UC10: Understand decisions	D6		Yes	Yes	Yes	No	Yes	No	No	No	Yes
	D7		No	No	No	No	No	No	No	No	No

[31]. This implies that a fuzzy argumentation modelling support tool might be useful for practical GDM, and in extension, to CBSD.

Results show only partial coverage of the "Refinement" activity (A5) from the point of view of proposal management (UC2). A few tools support the editing of proposals and arguments, but none keep track of changes. This requirement is important in practice because it is often necessary to come back to previous versions of proposals, e.g., when a change is blocked by participants. It is also important for postmortem evaluation of the GDM process, especially postmortem alternative evaluation (UC8) and team dysfunction identification (UC9).

Among the uncovered requirements, none of the analyzed tools support postmortem alternative evaluation (UC8). This is cause for concern since poorly framed problems are the source of multiple software catastrophes [24]. It would be interesting to track changes made to the alternatives throughout the project in order to detect when poor alternatives were chosen, or where radical changes to alternatives occurred during development.

No tools support the postmortem evaluation of team dysfunctions (UC9), except holaSpirit, which does it partially through the manual identification of tensions. With a tool able to keep track of changes, it would be possible to see how the decision evolved throughout the project. It would therefore be possible to perform some reasoning not only on one static model, but on a serie of evolving models. It seems however that at this time, no framework exists to study the evolution of argument models.

Given the importance given to the management of expertise in the literature [22, 24, 45], it is surprising that more tools are not taking it into account. Only two tools support it, but they do not implement voting mechanisms. In other words, expertise is not factored in any of the vote-based tools. This limits the capability of stakeholders to assess the relative merits of alternatives, as all proposals, arguments and votes are considered equal, which might not be the case in practice.

6 GOING FORWARD

Given the mismatch between the capabilities of existing tools and the need of CBCs, we propose a roadmap for research on model-based techniques for supporting CBSD.

6.1 Main research questions

Research on CBSD should at a minimum address the following:

RQ1. Inter-tool communication: Developers should be able to seamlessly use chains of relevant tools. For example, they could pick one tool for problem statement, a second for proposals and a third for argument documentation (UC1, UC2, UC3), and be able to link the decision data to a project management tool (UC5) of their choice. It is a classic problem in software engineering and model-driven development. We propose to use metamodelling and language globalization [15] to create an infrastructure for interoperability between tools.

RQ2. Tool support for fuzzy argumentation: Existing tools are primarily oriented toward rational decision-making. In practice, when facing complex decisions, developers favour a more naturalistic approach. Tool support should reflect this practical need. This would be possible through the extension of existing work on discussion modelling [5, 39] with techniques from design space modelling and exploration. We are currently working on a project to model online discussions and provide online feedback and analytics about them to the participants.

RQ3. Evolving decision models: It is currently possible to make static argumentation models. It should be possible to take multiple snapshots of static models in order to reason on the evolution of the models over time. For reasoning on design decision rationale, the monitoring and management of compliance [27] and assurance [33] documentation over time is relevant. Crucially, decision-making should be informed by the principled evaluation of alternatives. This can be accomplished by integrating into CBSD techniques developed for design space exploration [10]. Further, a support tool should integrate these in a metamodelling platform that would allow comprehensive management [8] and versioning [7].

RQ4. Monitoring Team Dynamics: Current tools provide limited information on team characteristics, like expertise (D2) or organizational context (D7). Since these characteristics vary from time to time, and from decision to decision, it would be important to keep track of them in order to support better reasoning. For example, CBCs might accrue "consensus debt" (akin to technical debt [35]) when provisional short term decisions need to be made to address urgent contingencies. This debt should be recorded, monitored and resolved. Keeping track of team dynamics would enable better modelling of the social network, which would connect to decision models with specific heuristics to detect degenerations. In this, it is crucial to integrate into CBSD insights from industrial psychology [9], taking into account the specific philosophical concerns of CBCs, especially centred around issues of intersectionality [1].

6.2 Epistemological approach

CBCs have their roots in social movements (e.g., Loomio originates from the Occupy Wall Street movement [37]), and research in CBSD should take in account its origins and the particular concerns that lie at its core philosophy. Specifically, the philosophy behind CBSD evolved from critiques of abuses of traditional organizational structures. From the viewpoint of social movements, inequalities extant in society are reproduced as individual behavior inside organizations unless special care is given to avoid them.

This includes, from the perspective of CBCs, academic researchers. This poses a practical and epistemological challenge for research in CBSD, since traditional post-positivist and constructivist epistemological stances might be met with significant resistance and suspicion from CBCs. Instead, we propose that critical theory is the appropriate epistemological framework for research in this domain. Easterbrook et al. describe it as follows [23]:

> Critical Theory judges scientific knowledge by its ability to free people from restrictive systems of thought [12]. Critical theorists argue that research is a political act, because knowledge empowers different groups within society, or entrenches existing power structures

Within a critical-theoretic epistemological framework, researchers prioritize the use of participatory approaches, that provide them with intimate insights and high levels of familiarity with the teams being studied [36], as well as action research. Action research is a research methodology, where researchers acknowledge that they are not objective observers and instead work together with a community to affect change, explicitly acknowledging their biases. The focus is on building relationships of mutual trust and accountability between researchers and the communities they study, and on participatory theory building in order to address real problems withing the community, under the principle that research and action are indivisible [19].

In short, research on CBSD should be conducted in close collaboration with CBCs themselves, ensuring that their real needs are addressed in accordance to the specific concerns and priorities in their social and political milieu.

7 RELATED WORK

The research questions provide interesting new avenues for the development of tool support for GDM within generic CBCs. However, how does this can be mapped to CBSD? Will the theoretical tool presented herein provide any benefit for software engineering, at least within CBC contexts?

The work of Drury et al. [22] on Agile software development corroborates some of the CBC challenges. They are critical of traditional decision theory being too rooted within the rational decision-making paradigm (RQ2):

> Normative decision theory views decision makers as idealized, rational, extremely intelligent beings who overcome their inner turmoil, shifting values, anxieties, post-decision regrets, fear of ambiguity, inability to perform intricate calculations and limited attention span to make rational, optimum choices. [22]

They also see the self-managed nature of Agile software development as challenging. They mention that traditional authoritarian structures normally ensure that the voice of the experts carry more weight than the voice of others (RQ4). They also mention that in Agile, decision are not always implemented as discussed during meetings, if at all (UC5).

In practice however, Agile software development is not as self-managed as theoretically designed. Shastri et al. [52], in a survey of Agile software developers, found that 67% of them still had a manager above them. These Agile teams come from traditional authoritarian organizations, or are within mixed organizations (Agile and non-Agile), and are still beholden to high management.

The works of Pérez et al. [45], while not specific to software, does study engineering teams. They note the heterogeneity of experts (D1) within modern engineering teams. For example, a software development team might include a user interface expert, a database expert, a security expert, etc. They therefore build a theoretical model to help engineering team manage this heterogeneity of expertise, through a weighting approach (RQ4). This heterogeneity becomes especially difficult during software requirement negotiations, as it includes participants with varying backgrounds: domain experts, software developers, non-technical stakeholders, etc. [30]. Pérez et al. [45]'s model integrates a feedback mechanism which recognizes that consensus is a building, evolving process (RQ3). The objective of their model is to support this evolving process by providing appropriate feedback to the participants, based on their expertise. As the authors write, engineering GDM "is a dynamic and iterative process, composed of several rounds where the experts express, discuss, and modify their preferences" [45].

The importance of expertise must however be balanced with the concern of technocratic autocracy, "where expertise is the basis of authority" [42]. The works of Moe et al. [42] outlines the importance of managing concerns (RQ5) to avoid the case where the decision-making process becomes the domain of unchallenged experts. Their work also show the lack of research into naturalistic decision-making in Agile software development. They write that:

> Today's software systems are becoming more complex because of the need to balance the often disparate needs of diverse stakeholders, and the growing complexity of the technology used. This reality requires more unstructured decision-making. Second, the principles of agile software development align with the definition of naturalistic decision-making. [42]

There is therefore a need, at least within Agile software development, for naturalistic GDM support tools (RQ2).

In summary, the research questions identified are not generic needs of CBCs, but can also be applied to software engineering. A number of these research questions are not only limited to CBSD, but could also provide benefits for Agile software development. For Agile contexts, we note especially the need for naturalistic GDM tool support (RQ2) and the need for expertise management (RQ4).

8 CONCLUSION

We have outlined the group decision-making (GDM) process and its associated challenges in the context of Consensus-Based Software Development (CBSD), based on existing practices within Consensus-Based Communities (CBCs). We structured these in high-level requirements for supporting the CBSD with appropriate tooling and analyzed a preliminary list of CBSD-related tools to find gaps in the existing functionalities. Based on this analysis, we posed outlined a vision for research in CBSD, identigying four research directions and outlining the epistemological and methodological challenges for such research.

REFERENCES

[1] Doyin Atewologun, Ruth Sealy, and Susan Vinnicombe. 2016. Revealing intersectional dynamics in organizations: Introducing 'intersectional identity work'. *Gender, Work & Organization* 23, 3 (2016), 223–247.

[2] Peter Baldwin and David Price. 2017. DebateGraph. (2017). http://debategraph.org/

[3] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. 2001. Principles behind the Agile Manifesto. (2001). http://agilemanifesto.org/principles.html

[4] Gregor Betz, Sebastian Cacean, and Christian Voigt. 2016. Argunet - Open-Source Argument Mapping. (2016). http://www.argunet.org/editor

[5] Elizabeth Black, Peter McBurney, and Steffen Zschaler. 2014. *Towards Agent Dialogue as a Tool for Capturing Software Design Discussions.* 95–110.

[6] Pierre Bourdieu. 1984. *Distinction.* Harvard University Press.

[7] Petra Brosch, Gerti Kappel, Philip Langer, Martina Seidl, Konrad Wieland, and Manuel Wimmer. 2012. An introduction to model versioning. In *Proceedings of the 12th international conference on Formal Methods for the Design of Computer, Communication, and Software Systems: formal methods for model-driven engineering.* Springer-Verlag, 336–398.

[8] Greg Brunet, Marsha Chechik, Steve Easterbrook, Shiva Nejati, Nan Niu, and Mehrdad Sabetzadeh. 2006. A manifesto for model merging. In *Proceedings of the 2006 international workshop on Global integrated model management.* ACM, 5–12.

[9] Keith W Buffinton, Kathryn W Jablokow, and Kathleen A Martin. 2002. Project team dynamics and cognitive style. *Engineering Management Journal* 14, 3 (2002), 25–33.

[10] Saheed A. Busari and Emmanuel Letier. 2017. RADAR: A Lightweight Tool for Requirements and Architecture Decision Analysis. In *Proceedings of the 39th International Conference on Software Engineering (ICSE '17).* IEEE Press, Piscataway, NJ, USA, 552–562. DOI:http://dx.doi.org/10.1109/ICSE.2017.57

[11] F. J. Cabrerizo, F. Chiclana, M. R. Ureña, and E. Herrera-Viedma. 2013. Challenges and open questions in soft consensus models. In *2013 Joint IFSA World Congress and NAFIPS Annual Meeting (IFSA/NAFIPS).* 944–949. DOI:http://dx.doi.org/10.1109/IFSA-NAFIPS.2013.6608527

[12] Craig Calhoun. 1995. *Critical social theory: Culture, history, and the challenge of difference.* Wiley-Blackwell.

[13] Robert Cialdini. 2001. The Science of Persuasion. *Scientific American* 284 (2001), 76–81.

[14] Civilized Discourse Construction Kit. 2017. What is Discourse? - Discourse - Civilized Discussion. (2017). http://www.discourse.org/about

[15] Benoit Combemale, Julien Deantoni, Benoit Baudry, Robert B France, Jean-Marc Jézéquel, and Jeff Gray. 2014. Globalizing modeling languages. *Computer* (2014), 10–13.

[16] Melvin E. Conway. 1968. How do committees invent? *Datamation* 14, 4 (1968), 28–31.

[17] Andrew Cornell. 2011. *Oppose and Propose!* AK Press.

[18] Fabiano Dalpiaz, Xavier Franch, and Jennifer Horkoff. 2016. *iStar 2.0 Language Guide.* Technical Report.

[19] Paulo Sergio Medeiros Dos Santos, Guilherme Horta Travassos, and Marvin V Zelkowitz. 2011. Action research can swing the balance in experimental software engineering. *Advances in computers* 83 (2011), 205–276.

[20] Andy Doyle. 2016. Management and Organization at Medium. (2016). https://blog.medium.com/management-and-organization-at-medium-2228cc9d93e9

[21] Drupal. 2016. Drupal Code of Conduct. (2016). https://www.drupal.org/dcoc

[22] M. Drury, K. Conboy, and K. Power. 2011. Decision Making in Agile Development: A Focus Group Study of Decisions and Obstacles. In *2011 Agile Conference.* 39–47. DOI:http://dx.doi.org/10.1109/AGILE.2011.27

[23] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. 2008. Selecting empirical methods for software engineering research. *Guide to advanced empirical software engineering* (2008), 285–311.

[24] Chris Edwards. 2009. Agreeing to fail. *Engineering & Technology* 4, 7 (2009), 74–75. DOI:http://dx.doi.org/10.1049/et.2009.0716

[25] Tomás Flanagan, Claudia Eckert, and P. John Clarkson. 2007. Externalizing Tacit Overview Knowledge: A Model-based Approach to Supporting Design Teams. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 21, 3 (June 2007), 227–242. DOI:http://dx.doi.org/10.1017/S089006040700025X

[26] D. A. López García, T. d. J. Mateo Sanguino, E. Cortés Ancos, and I. Fernández de Viana González. 2016. A Debate and Decision-Making Tool for Enhanced Learning. *IEEE Transactions on Learning Technologies* 9, 3 (July 2016), 205–216. DOI:http://dx.doi.org/10.1109/TLT.2016.2556664

[27] Sepideh Ghanavati, Daniel Amyot, and Liam Peyton. 2007. Towards a framework for tracking legal compliance in healthcare. In *Advanced Information Systems Engineering.* Springer, 218–232.

[28] Thomas F. Gordon. 2016. Carneades - tools for argument (re)construction, evaluation, mapping and interchange. (2016). http://carneades.github.io/

[29] holaSpirit SAS. 2017. holaSpirit - Application for Holacracy and teal organizations. (2017). https://www.holaspirit.com/

[30] Hoh In and S. Roy. 2001. Issues of visualized conflict resolution. In *Proceedings Fifth IEEE International Symposium on Requirements Engineering.* 314–315. DOI:http://dx.doi.org/10.1109/ISRE.2001.948601

[31] Jeroen Janssen, Martine De Cock, and Dirk Vermeir. 2008. *Fuzzy Argumentation Networks.* Technical Report. Vrije Universiteit Brussel. 1–15 pages.

[32] Arun Kalyanasundaram, Wei Wei, Kathleen M. Carley, and James D. Herbsleb. 2015. An Agent-based Model of Edit Wars in Wikipedia: How and when is Consensus Reached. In *Proceedings of the 2015 Winter Simulation Conference (WSC '15).* IEEE Press, Piscataway, NJ, USA, 276–287. http://dl.acm.org/citation.cfm?id=2888619.2888648

[33] Sahar Kokaly, Rick Salay, Valentin Cassano, Tom Maibaum, and Marsha Chechik. 2016. A model management approach for assurance case reuse due to system evolution. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems.* ACM, 196–206.

[34] Koumbit. 2017. About us. (2017). https://www.koumbit.org/en/about

[35] Philippe Kruchten, Robert L Nord, and Ipek Ozkaya. 2012. Technical debt: From metaphor to theory and practice. *Ieee software* 29, 6 (2012), 18–21.

[36] Timothy C. Lethbridge, Susan Elliott Sim, and Janice Singer. 2005. Studying Software Engineers: Data Collection Techniques for Software Field Studies. *Empirical Software Engineering* 10, 3 (2005), 311–341. DOI:http://dx.doi.org/10.1007/s10664-005-1290-x

[37] Loomio Cooperative Limited. 2016. Loomio - Better decisions together. (2016). https://www.loomio.org/

[38] Loomio Cooperative Limited. 2016. The Loomio Co-op Handbook. (2016). https://www.loomio.coop/

[39] Ahmed Shah Mashiyat, Michalis Famelis, Rick Salay, and Marsha Chechik. 2014. Using Developer Conversations to Resolve Uncertainty in Software Development: A Position Paper. In *Proceedings of the 4th International Workshop on Recommendation Systems for Software Engineering.* 1–5.

[40] A. Maturo and A. G. S. Ventre. 2008. Models for consensus in multiperson decision making. In *NAFIPS 2008 - 2008 Annual Meeting of the North American*

Fuzzy Information Processing Society. 1–4. DOI: http://dx.doi.org/10.1109/NAFIPS.2008.4531213

[41] David G Messerschmitt, Clemens Szyperski, and others. 2005. Software ecosystem: understanding an indispensable technology and industry. *MIT Press Books* 1 (2005).

[42] Nils Brede Moe, Aybüke Aurum, and Tore Dybå. 2012. Challenges of shared decision-making: A multiple case study of agile software development. *Information and Software Technology* 54, 8 (2012), 853–865. DOI: http://dx.doi.org/10.1016/j.infsof.2011.11.006 Special Issue: Voice of the Editorial Board.

[43] Mountain Goat Software. 2017. PlanningPoker.com - Estimates Made Easy. Sprints Made Simple. (2017). https://www.planningpoker.com/

[44] Nachiappan Nagappan, Brendan Murphy, and Victor Basili. 2008. The Influence of Organizational Structure on Software Quality: An Empirical Case Study. In *Proceedings of the 30th International Conference on Software Engineering (ICSE '08)*. ACM, New York, NY, USA, 521–530. DOI: http://dx.doi.org/10.1145/1368088.1368160

[45] Ignacio Javier Pérez, Francisco Javier Cabrerizo, Sergio Alonso, and Enrique Herrera-Viedma. 2014. A New Consensus Model for Group Decision Making Problems With Non-Homogeneous Experts. *IEEE Transactions on Systems, Man and Cybernetics: Systems* 44, 4 (2014), 494–498.

[46] ReasoningLab.com. 2017. bCisive - online decision mapping. (2017). https://www.bcisiveonline.com/

[47] Reddit. 2017. Reddit. (2017). https://about.reddit.com/

[48] Rishipal. 2014. Analytical Comparison of Flat and Vertical Organizational Structure. *European Journal of Business and Management* 6, 36 (2014), 56–65.

[49] Brian J. Robertson. 2007. *Organization at the Leading Edge: Introducing Holacracy.* Technical Report.

[50] Seeds for Change. 2013. *A Consensus Handbook.* Footprint Workers' Co-op.

[51] M. Shahin, P. Liang, and M. R. Khayyambashi. 2009. Architectural design decision: Existing models and tools. In *2009 Joint Working IEEE/IFIP Conference on Software Architecture European Conference on Software Architecture.* 293–296. DOI: http://dx.doi.org/10.1109/WICSA.2009.5290823

[52] Y. Shastri, R. Hoda, and R. Amor. 2016. Does the 'Project Manager' Still Exist in Agile Software Development Projects?. In *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*. 57–64. DOI: http://dx.doi.org/10.1109/APSEC.2016.019

[53] Matthew J. Turk. 2013. Scaling a Code in the Human Dimension. In *Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery (XSEDE '13)*. ACM, New York, NY, USA, Article 69, 7 pages. DOI: http://dx.doi.org/10.1145/2484762.2484782

[54] J. Tyree and A. Akerman. 2005. Architecture decisions: demystifying architecture. *IEEE Software* 22, 2 (March 2005), 19–27. DOI: http://dx.doi.org/10.1109/MS.2005.27

[55] Maarten Van Mechelen, Mathieu Gielen, Vero vanden Abeele, Ann Laenen, and Bieke Zaman. 2014. Exploring Challenging Group Dynamics in Participatory Design with Children. In *Proceedings of the 2014 Conference on Interaction Design and Children (IDC '14)*. ACM, New York, NY, USA, 269–272. DOI: http://dx.doi.org/10.1145/2593968.2610469

[56] Delfina Vannucci and Richard Singer. 2010. *Come Hell or High Water.* AK Press.

[57] E. S. K. Yu. 1997. Towards modelling and reasoning support for early-phase requirements engineering. In *Requirements Engineering, 1997., Proceedings of the Third IEEE International Symposium on.* 226–235. DOI: http://dx.doi.org/10.1109/ISRE.1997.566873

[58] Carmen Zannier, Mike Chiasson, and Frank Maurer. 2007. A model of design decision making based on empirical results of interviews with software designers. *Information and Software Technology* 49, 6 (2007), 637–653. DOI: http://dx.doi.org/10.1016/j.infsof.2007.02.010

[59] Zappos Insights. 2017. Holacracy - Flattening the Organization Structure and Busting Bureaucracy. (2017). https://www.zapposinsights.com/about/holacracy

[60] Dave Zwieback. 2015. *Beyond Blame, Learning from successes and failure.* O'Reilly Media.