The Effects of Team Backlog Dependencies on Agile Multiteam Systems: A Graph Theoretical Approach

Alexander Scheerer  
SAP SE & University Mannheim  
scheerer@uni-mannheim.de

Saskia Bick  
SAP SE & University Mannheim  
bick@uni-mannheim.de

Tobias Hildenbrand  
SAP SE  
tobias.hildenbrand@sap.com

Armin Heinzl  
University Mannheim  
heinzl@uni-mannheim.de

Abstract

In agile software development, the prioritization of backlog items is the mission-critical responsibility of the product owners in order to maximize the customer value created by development teams. However, in the reality of large-scale development, the degree of freedom for such a prioritization is substantially restricted by various types of interdependencies between backlog items. In this work, we show, using a graph theoretical approach, the relation between the degree of freedom for prioritization and the occurrence of dependencies. To the best of our knowledge, the breadth and depth of such consequences has never been modeled or investigated up until now. Based on our results, we derive implications for real-world large-scale software development in agile environments.

1. Introduction

In agile software development, the prioritization of requirements in the form of backlog items is the mission-critical responsibility of product owners (“the sole person responsible for managing the backlog” [41:5]), both on a small scale, starting with a single team, as well as on a larger scale, i.e. in a multiteam system setting [25,38]. Single or multiteam, a backlog “is an ordered list of everything that might be needed […] and is the single source of requirements for any changes to be made” [41:12]. The degree of freedom (DoF), i.e. the relative amount of choice, compared to a backlog without dependencies, to freely prioritize team backlog items, is an essential instrument regarding not only team empowerment but also customer centricity in order to maximize the customer value created by development teams [38]. In reality, however, large-scale agile software development, and accordingly the DoF while prioritizing team backlog items, is substantially restricted by various types of dependencies between items.

Ideally, empowered development teams in a multiteam system setting should face a low degree of inter-team dependencies and a higher degree of intra-team dependencies to allow for effective multiteam system coordination and efficient intra-team work [45]. Yet, in real-world examples of large-scale agile development environments, product owners are often complaining about being responsible for prioritizing requirements and creating value for customers, while the stage is already set due to given dependencies. As described in the Scaled Agile Framework (SAFe) by Leffingwell [26], cross-team features in the shared program backlog are split into single-team backlog items, i.e. user stories, with tentative allocations to individual sprints, which limits the product owners and their teams in their “free choice” of backlog item prioritization.

According to the agile development paradigm, maximizing customer value and thus ensuring economic product viability for the company is a key principle [38]. Nevertheless, products are rarely built from scratch and – as a timely delivery of new functionality for standard solutions is often based on existing platforms, components and frameworks – legacy code may impose several restrictions for the development of new code [17].

These constraints on the product owner’s possibility to influence the order of the backlog are not only a highly relevant practical problem [4,24,26], but previous research has also called for more studies on the underlying fundamental concepts of agile software development [1,2,16]. Dependencies on an inter-team level are an under-researched fundamental concept which still lacks scientific depth [cf. 4,8,14]. The research question we seek to answer is: What are the impacts of dependencies in the team backlog and how do these effects emerge on a multiteam level?

As a first step towards solving this problem, our research aims at providing insights into consequences of dependencies in team backlogs in large-scale agile software development and suggesting two main approaches how to deal with existing dependencies. In this work, we show – using a simulative graph theory approach – the relation between level of dependencies and the degree of freedom for prioritization of backlog items. Based on our results, we derive implications for real-world large-scale software development in agile
environment in that we highlight the severity of dependencies in general and for multiteam systems in particular.

Our paper is structured as follows. In the foundations chapter, we depict current findings on agile software development, dependencies and coordination theory in order to provide the necessary knowledge for the subsequent modeling approach chapter. Here, assumptions and modeling items as well as the mathematical procedure are explained. The results are critically reflected thereafter in the discussion section. We conclude our work with future research implications.

2. Foundations

Starting with a brief overview on agile software development, we illustrate the environment surrounding our simulative study. After introducing large-scale agile and multiteam systems, a concept stemming from organizational psychology research, we provide a short overview on dependencies. Subsequently, we introduce the conceptual fundamentals of coordination.

2.1. Agile Software Development

In 2001, several prominent advocates of lightweight development methods gathered to create the Agile Manifesto [6]. They proposed four values which constitute the essence of agile development methods: individuals and interactions, working software, customer collaboration and responding to change. These values illustrate the considerable mind shift in agile software development. They emphasize cross-functional and empowered teams with time-boxed development iterations and continuous management of requirements.

Agile project management frameworks, such as the Scrum approach [40], and large-scale agile development approaches, e.g. the SAFe [26], which builds upon the former, both suggest work management through ordered backlogs. On a team level, such a backlog contains all items one team will directly work on. In agile software development, these backlog items are often represented as user stories, which describe a feature from the perspective of a certain user role including the value it provides in his/her context and gives the development team enough freedom to work out the details in implementation. The team can pull one or more of these items from the team backlog into their sprint backlog, i.e. for a timeframe of two to four weeks. After discussing the user stories and estimating the efforts, the team then commits these items for the sprint. A unique prioritization is recommended [41].

This prioritization is carried out by the product owner, who serves as the representative of the customers and thus should be able to order the backlog according to customer priority and value. The prioritization of requirements is one of the key functions in assuring customer focus, mitigating risk, fast value delivery and successful steering of a project. However, this job is strongly affected by dependencies between individual backlog items or user stories in particular.

Regarding this field of research, a considerable number of studies have been conducted concerning agility on a team level, some of which are still asking for more rigorous research and detailed insights as opposed to reports of mere anecdotal evidence [1,2,16]. Concerning large-scale agile development research, insights are even more limited [15].

2.2. Large-Scale Agile Development and Multiteam Systems

Large-scale agile development very closely builds upon the agile software development paradigm described before. In fact, it extends the latter by implying the collaboration of multiple teams which all make use of agile methodologies. On the practical side, several approaches, e.g. Scrum-of-Scrums [23]; Dynamic Systems Development Method [12] or Disciplined Agile Delivery [3], have been developed to provide a guiding structure for a multiteam setup. In our consideration of the research question, we will utilize the Scaled Agile Framework as an example [26]. Here, a program backlog contains potential product features which are further broken down into smaller user stories to nourish the team backlogs. User stories of one larger feature can be worked on by several different development teams. The individual product owner of a development team is henceforth provided with user stories coming from the higher level program backlog. Accordingly, he/she has to deal with pre-determined, and partly unknown, dependencies in the team backlog and is therefore limited in the DoF to prioritize the backlog [26].

Coming from organizational theory, another view on large-scale development systems emerged. This type of organizational setup, where several teams have to work together in order to complete a release of a development product, has been described within the organizational psychology domain as a multiteam system (MTS). The collective goal of this system can be broken down into a goal hierarchy and constitutes a key characteristic of any MTS. The goal hierarchy marks the boundary of an MTS in that all teams within the system share at least a distal goal while the individual teams pursue their more proximal goals individually. This structure of goals leads to teams displaying input, process and outcome interdependence with at least one other team [29,39].

5125
Whereas a certain level of research in the organizational psychology community already exists to provide us with insights on multiteam systems and resulting dependencies in general, research on dependencies in the large-scale agile development setting in particular is still rather scarce [4,5,15,24]. Learning from extant research, we aim to deepen the knowledge in the agile software development context as well as transfer our insights to real-world large-scale development settings.

2.3. Requirements Prioritization and Ordering

The term priority has historically been used in ambiguous ways, implying different concepts to different stakeholders. The definition of priority according to two English language dictionaries exemplifies this statement (see Table 1). While (2), (3) and (5) deal with a certain order in time or scheduling, (1) and (4) describe relative importance. These two concepts can also be seen as the basis for the prevalent views on prioritization. A widespread conception for requirements prioritization is to attach a priority to individual activities based on value, risk or cost (among many others). An example of this approach is the MoSCoW method [9], which describes four categories of deliverables: must, should, could and won’t. It becomes apparent that this approach describes the relative importance between individual requirements. However, Firesmith [18:37] argues, that “requirements prioritization is determining the implementation order of requirements and that prioritization by importance is merely one means to that end”. As such it has been suggested to use the term order to be more precise, which of the two meanings of prioritization is intended. Orders’ sole meaning, as can be seen in Table 1, is the arrangement of activities in time.

This focus can also be seen in the decision of the authors of the Scrum Guide [41] to change the wording from priority or prioritized backlog to ordered backlog. This emphasizes the goal of creating a list which has a unique order in which to process backlog items [13].

For this paper, we make use of the following conceptualization: Prioritization means to assign a relative value/importance to one particular activity. Ordering refers to a unique sequence of several activities. It is the job of the product owner to first assign priorities (and he is limited in doing so in a completely free manner by pre-determined dependencies) and then to order these prioritized backlog items in a unique way.

2.4. Dependencies and Coordination

We view the previously sketched research problem through the lens of coordination theory, which defines coordination as the management of dependencies [28].

In the literature, the terms dependency and interdependency are used interchangeably and refer both to a unilateral relationship, i.e. a is dependent on b but not the other way round. Accompanying and resulting from coordination theory, there are different taxonomic approaches to characterize dependencies, the most important of which we present subsequently. The widely used term task is to be understood as work or activity in this context. A traditional view was proposed by Thompson [45], who classified task dependencies into pooled, sequential and reciprocal types. In this typology, interdependence is conceptualized as a need for activity alignment. Based on this taxonomy, Van de Ven et al. [46] further specified the participating entities as workers within an organizational unit who are dependent upon each other in order to fulfill their individual tasks. Furthermore, the authors introduced a fourth type of interdependence: team interdependence. This state is present when work is done jointly and simultaneously by employees of one organizational unit at the same time. Yet another dependency taxonomy provided by Crowston [14] focuses on the relationship between tasks and resources which can

Table 1. Definitions of key terms.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
</table>
| Priority | (1) a preferential rating  
(2) the quality or state of being prior  
(3) something given or meriting attention before competing alternatives  
(4) A thing that is regarded as more important than another  
(5) The right to take precedence or to proceed before others |
| Order | The arrangement or sequence of objects or of events in time. [32]  
The arrangement or disposition of people or things in relation to each other according to a particular sequence, pattern, or method. [35] |
| Dependency | A situation wherein an activity cannot proceed until another activity is complete. [44:6] |
| Dependent | (1) Unable to do without  
(2) Contingent on or determined by [36] |
arise in organizational settings and proposes different combinations of these two entities, namely shared resources, producer-consumer and common output. Strode et al. [44] view dependencies as the antecedent of potential or actual constraints which might impact the success of a specific activity. The authors name three types of dependencies: knowledge, task and resource dependencies, which each comprise multiple sub-categories.

Figure 1. Categorization of Task Interdependencies [22,45]

In our research, we view a dependency as “a situation wherein an activity cannot proceed until another activity is complete” [44:6], i.e. backlog item A must be implemented first, as it depends on backlog item B (see also Table 1).

For this study, in the context of MTS in large-scale agile development, we make use of the relatively new concept of integration interdependence [22], as this addresses the integration challenges of several teams working together on one software product. Kumar et al. describe four characterizing aspects of integration interdependence. First, the overall work activity, in our case the program backlog feature, is subdivided into several sub-activities for multiple parties to work on separately. Second, not the individual activities themselves, but the whole, aggregated sum of the parts is of value. Third, to reintegrate the parallel working streams, an integration process is needed. Fourth, all members of the system collaborating in a setting of integration interdependence have to be aware of the status, potential issues and the progress of their co-teams as their own output is heavily dependent on the work of the others. Figure 1 shows the three classical interdependence types by Thompson [45] as well as the recent concept of integration interdependence by Kumar et al. [22].

As already described in chapter 2.2, these circumstances influence the degree of freedom to prioritize the team backlog items for the product owner. To be more precise, out of these integration interdependencies, other order dependencies, i.e. unilateral dependencies that express constraints and a specific directionality, arise in the team backlog and have to be considered when prioritizing a specific work sequence. In order to manage these dependencies, coordination mechanisms are needed to actively manage the backlog. Yet, before exploring resolution and prevention strategies, it is of importance to first closely investigate consequences of dependencies in agile backlogs. To the best of our knowledge, the breadth and depth of such consequences has never been modeled or investigated up until now.

In the next chapter, we introduce our modeling approach and present guiding assumptions and used parameters.

3. Modeling Approach

The approach taken within this research, to model a team backlog in a large-scale agile setting, is a simulative graph theoretical one. We initially chose this path as the problem presented itself to be of a multi-dimensional nature. Further understanding was needed in order to analytically tackle this complex issue. For our modeling approach, we consider the backlog of one team, as the multiteam system consists of several individual teams working together. In order to fully comprehend dependencies in the multiteam system, one has to first understand the dependencies emerging at the individual team which originate from the multiteam setting. We view this as the first step in understanding the complex dependencies in the full MTS.

The following paragraphs explain how we model the different entities relevant to this research and their representation in graph theoretical terms. Details on the calculations conclude this chapter.
3.1. Team Backlogs, Backlog Items and Dependencies

The team backlog is modeled as a directed acyclic graph (DAG), i.e. a graph with no cyclical arrangement of vertices. We model each backlog item as vertices within the DAG. Further, we view any kind of relationship between backlog items which necessitates that one be finished before a different one as a dependency between these two items. The edges between vertices in our DAG represent the order dependencies between individual backlog items.

In summary, we model each team backlog with $v$ items and $e$ dependencies as a directed acyclic graph with unlabeled nodes and $v$ vertices as well as $e$ edges. We represent the backlog as a graph with unlabeled nodes, as the backlog items are all of the same type, in our case a user story, and the structure of the graph is essential, not the naming of nodes. Hence, we only consider one graph of each isomorphism class in our approach. The number of DAGs with unlabeled nodes is shown in Table 2.

We limit our simulative examination to backlogs with a maximum amount of ten items due to reasons of super-exponential growth of this integer sequence.

Table 2. Number of acyclic digraphs with $n$ unlabeled nodes [42]

<table>
<thead>
<tr>
<th>$n$</th>
<th>$A(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>31</td>
</tr>
<tr>
<td>5</td>
<td>302</td>
</tr>
<tr>
<td>6</td>
<td>5,984</td>
</tr>
<tr>
<td>7</td>
<td>243,668</td>
</tr>
<tr>
<td>8</td>
<td>20,286,025</td>
</tr>
<tr>
<td>9</td>
<td>3,424,938,010</td>
</tr>
<tr>
<td>10</td>
<td>1,165,948,612,902</td>
</tr>
</tbody>
</table>

3.2. Calculating the Degree of Freedom

Degree of Freedom. We view the relative amount of choice, compared to a backlog without dependencies, to freely prioritize team backlog items as the degree of freedom. This is defined as the ratio of valid permutations $P_v$ in a backlog with dependencies to all possible permutations $P_a$, reflecting the portion of independent prioritization available such that $DoF = \frac{P_v}{P_a}$.

Implicit or Transitive Dependencies. By specifying dependencies within the backlog it is possible to have implicitly generated more dependencies as the initial specification would suggest. This situation can be seen in Figure 2. To account for transitive dependencies, we calculated the transitive closure of each DAG, which adds these implicitly specified dependencies to the graph representation of each backlog.

![Figure 2. Implicit (Transitive) Dependencies](image)

Valid Permutations. A permutation which respects the dependencies such that for each directed edge $v1 \rightarrow v2$ from vertex $v1$ to vertex $v2$, $v1$ comes before $v2$ is defined as a topological order. Thus, the amount of topological orders for each transitive closure is the amount of valid permutations for the specified DAG. At least one, but usually several orders can be calculated, which represent the ways in which the backlog can be processed while fulfilling the dependencies specified.

In summary, we establish the DoF by calculating the ratio of the number of topological orderings of each transitive closure of the specified DAG with edges and the number of topological orderings of the DAG without edges (see Figure 3).

Figure 3. Definition of the Degree of Freedom

We have generated our data set with the help of Sage Mathematics Software [33,43]. Hereby, we calculated the topological orderings for all unlabeled DAGs with nodes 1 to 8 [30] and created 18 million random DAGs with 9 vertices as well as 20 million random DAGs with 10 vertices. The previously mentioned super-exponential growth of increasingly larger DAGs makes the smaller subset of DAGs with 9 and 10 vertices necessary. In total, we calculated the DoF of 58 658 547 representations of a backlog with dependencies.
4. Results and Discussion

The degrees of freedom, including a trend line for increasing amounts of dependencies, can be seen for 2 to 10 item backlogs in Figure 5.

In answering the question of dependency impact on the team backlog, our main findings are (1) the DoF exhibits very strong decay, (2) this decay is independent of backlog size and (3) the structure of dependencies seems to be essential as only ranges of DoF could be identified for most combinations of parameters.

One dependency already decreases the DoF by 50% and this number drops below 20% as soon as we pass four dependencies (see Figure 5). The data shows strikingly, an increased amount of dependencies dramatically lowers the DoF. This confirms our initial expectation in chapter 2.2, that predetermined dependencies have a significant impact on the prioritization ability of the team product owner. In fact, the product owner is drastically limited in freely assigning priorities to the backlog items and subsequently ordering them accordingly. One dependency between two backlog items, for example, will already reduce his freedom of choice in how to prioritize the backlog by 50%.

The independence of DoF from backlog size initially seems counterintuitive, but can be explained by our definition of DoF, which normalizes the number of topological orderings of the backlog with dependencies by the topological orderings of the same backlog without dependencies.

What quickly becomes obvious is that for the tuple backlog size – dependency amount, only a range of possible DoF can be specified. This also holds true for the combination of other parameters which we calculated (Connected Components, Density, average Distance, Diameter, Radius, max In/Out-Degree, min In/Out-Degree, In/Out-Degree Sequence). In Figure 4 we provide an excerpt of the DoF for an 8 item backlog based on the number of bound items and the amount of dependencies which clearly shows the previously mentioned ranges. These ranges suggest that the structure of the order dependencies is essential in understanding the exact consequences of dependencies in team backlogs.

With regard to our initially stated research question, we see that the previously discussed effects emerge on the multiteam system level in a way that the greater the interdependence between work activities, the higher the coordination effort resulting in a higher chance of failure and delay [22]. Due to the system of integration interdependencies, delays on a team level have direct effects on the higher multiteam system level. If one team cannot deliver in time what it committed to for the sprint, other teams and thus the whole multiteam system might experience delays in a cascading manner.

Influenced by the level of existing dependencies in a team backlog, the resulting degree of freedom impacts the team’s planning complexity and thus its coordination effort. This impact is particularly severe as a certain level of remaining flexibility in the product owner’s freedom to prioritize items in the team backlog is needed to be able to react to unforeseen events as one of the key values of agile development [21]. This in turn affects the overall multiteam system predictability and delivery reliability [22].

5. Implications and Future Research

We have managed to gain first insights into the consequences of dependencies in team and program backlogs in large-scale agile software development systems. Even few dependencies in the team backlog dramatically reduce the DoF for prioritization of the team product owner. This impact is independent of backlog size and has a severe impact on the DoF.

The surrounding MTS setting has an impact on its embedded teams regarding the dependencies in team backlogs, as the team product owner has to consider pre-determined dependencies coming from assigned user stories in his/her team backlog. In the other direction, the effects of dependencies in the team backlog and the according prioritization in turn strongly influence the MTS level through the integrative nature of dependencies on the program backlog level. It is therefore essential to consider ways in which to deal with these types of dependencies.
The findings so far suggest two approaches how to deal with dependencies in large-scale settings. The first one refers to the management of dependencies of which a few heuristics and methods have emerged lately. Splitting requirements (“user stories”) in smaller chunks [25,38] as well as approaches which attempt to generate an overview of the connections between requirements (e.g. user story mapping [37] or walking skeleton [11]) are mechanisms which make requirement dependencies transparent early on.

Creativity techniques in upfront software development stages, such as Design Thinking [10], in combination with user story mapping are said to help creating a more solid product vision and more relevant requirements to start with [19]. This in turn helps to minimize functional dependencies and make them transparent across development teams and business units. The described techniques used in the context of larger features can also help to split and resolve user stories in the course of a project. There are also approaches to capture dependencies collaboratively in the course of the project while providing visual analysis and editing functions in real time [20].

Once requirements have been defined, many companies use backlog management and tracking tools to further improve transparency and coordinate work across multiple teams.
The other approach advocates the avoidance of dependencies altogether. Especially strategies concerning the composition of multiteam systems and teams in particular are worth mentioning. The current discussion on feature vs. component teams is a step in this direction (cf. [8]). The architecture of the software also plays a strong role in dependency avoidance. Here, modularization strategies from open source development, i.e. plug-in architectures, could help in this regard (cf. [27]).

Besides these highly relevant practical implications, our study also provides theoretical contributions. First, based on a literature review on agile development and dependencies, our study required carving out the subtleties of the existing interdependency taxonomies. In particular, applying our findings to the multiteam level contributes to the body of knowledge on the concept of integration interdependence. Doing so, we have been able to deepen the knowledge on interdependencies on different analytical levels. Furthermore, we sought to answer the persistent call for more rigorous studies on agility in software development by examining the under-researched, yet tremendously important phenomenon of dependencies in agile backlogs. While the amount and quality of studies on agility in large-scale development settings constantly increases, the effects of dependencies at the core of agile work teams, namely their ordered list of requirements, have mostly been disregarded.

Future studies could build upon the presented findings and pursue an analytical approach to mitigate the limitations of the previously discussed interval results. In our research, this approach has so far proved unsuccessful as the parameters to determine a distinct degree of freedom remained elusive. At the same time, specific dependency patterns, i.e. the structure of order dependencies in simulative calculations could be of immense interest.

Speaking of limitations, the simulative character of our study could be regarded as such. Furthermore, we only looked at one type of dependency at this point. Reciprocal dependencies are just one potential other type of dependencies which might be of interest in the future. In addition, our focus on the team backlog, although justified, implies certain limitations for the generalizability of our findings.

Future research should therefore concentrate on the following main areas: on the one hand, the need for deeper insights on inter-team dependencies, in large-scale development requires more empirical studies in a real-life setting. With respect to this organizational structure, the effect of different types of coordination and dependencies as well as mitigation strategies and how they affect each other and the development system seem to be a promising line of inquiry.

In this light, the actual impact of certain methods for resolving and avoiding dependencies, e.g. splitting requirements and user story mapping need to be researched more in depth. So far, only practitioner reports are available which state positive effects of these methods over numerous projects.

6. References


