CAPCO

AUTOMATED WORK FORECASTS WHEN USING THE AGILE POD APPROACH

ABSTRACT

Agile pods are small, custom agile teams that provide an effective way to manage engagements where backlog items vary in the type/scope of work and that potentially require different skillsets from one another. A key challenge of using this approach is creating realistic forecasts of when development of individual backlog items is bound to commence and finalize (deliver) so customer expectations can be set realistically. In this paper, we address this challenge by presenting a method that automatically assigns pods to sprints so resource utilization is maximized.

The method also automatically generates a Gantt chart as output that can be used to help adjust the overall team size and structure and/or reprioritize the backlog to ensure the work is being managed as efficiently as possible. Moreover, we developed an MVP implementation of this method and a link to the source code is provided. Finally, we discuss two potential extensions of the model. The first pertains to the auto-prioritization of the backlog given a fixed team so the execution time is minimized. The second extension pertains to defining an optimal team structure to work through a pre-prioritized backlog that has hard limits on the execution time.

INTRODUCTION

What is an Agile pod

Agile pods are small, custom agile teams, where each pod is responsible for a single task, requirement, or part of the backlog. Each agile pod can be of whatever size is needed for the task at hand, whether it's a small pod to define a solution, or a larger one to build upon the initial groundwork of the smaller pod.[1]
Pod members have skill sets and experience levels that complement each other and that are deliberately matched to best meet the need of the current backlog item they are assigned to. When the current pod assignment is completed, the pod disbands and members are assigned to (a) different pod(s), based on the next set of prioritized backlog items.

The type of engagements where pods are highly effective in is where the backlog consists of items that vary significantly in complexity, skills needed, time pressure, and sometimes in the nature of the work itself (for example, compare executing a cloud vendor recommendation study vs. conducting a proof-of-concept related to adopting serverless technology). As a practical example, consider a backlog that is comprised of 20+ diverse items, and a shared pool consisting of 15 resources, made up

1. Attacking the Two Pizza Problem: Agile Pods, Alexander Gladshtein, 2019, Attacking the Two Pizza Problem: Agile Pods | LinkedIn

of architects, business analysts, engineers, and testers. It will be the mandate of this shared resource pool to collectively work its way through the backlog of items in the most efficient manner possible for the duration of the assignment. Logically, during the time of the engagement, additional items may be added to the backlog, or item prioritization may change. Indeed, over time, even the composition of the shared resource pool may vary, i.e., restructuring the team to better meet demand within time boundaries.

Pods differs from scrum in that:[2]

- Pods are designed by external people based on the needs to fulfill the requirements. Scrum teams self-organize from the ground up.
- Pods change as needed per the requirements and skillset. Scrum teams are cross-functional and long-lived.

Benefits of the Agile pod approach

Following a pod approach for scenarios, such as the one described above, holds several advantages over traditional scrum teams for the following reasons:[3]

- Improved project delivery times since the pod had specifically been optimized for the task at hand,
- Better quality – each pod stays focused on achieving its goals,
- Cross functional awareness: Due to the continuous stand-up and tear-down nature of agile pods, team members develop a greater appreciation of how different roles interact with each other and an understanding of other mindsets and perspectives within the team.

Problem Statement

Since pods are comprised of roles to best meet the specific demands of each backlog item, the number of pods actively working backlog items will logically vary from iteration to iteration.

A question that arises is whether any techniques exist to manage prioritization among pods for teams with multiple product owners, especially if specific talent is in short supply [1]. Linked to this: A product owner may want to know approximately when any given item in the backlog will be worked on to set realistic expectations and manage interlocks with any other work that is dependent on the outcomes of this item.

To help answer these questions, we developed a model and technique that quantifies backlog forecasting through automation, which will be the focus for the remainder of this paper. We will first present a simplified version of the model to establish the basic premise, and then add additional complexity to it to better fit the realities of actual agile pod engagements. We also created an implementation of the simplified model. The source code for this implementation can be found at the following repository https://github.com/diegosarai/gantt-generator.

2. Agile Pods vs Scrum Teams, Curtis Slough, 2019, Agile Pods vs Scrum Teams | Scrum.org
3. Advantages of Working with Agile Pod Teams, Agile Pod Delivery Model | Advantages of Using Agile Pods | FDM Group | UK

SOLUTION

Basic Approach

We designed and built a Java application (using Java FX) that iterates over several sprints through a prioritized backlog that has pre-defined, for each backlog item:

- The pod structure. I.e., the desired role count for addressing the specific backlog item (example: four quality assurance (QA), three data architects, four solution architects and 50% of time from a software engineer).

- The estimated number of sprints required to complete the specific backlog item (considering the pod resource breakout).

When running the analysis, the total project resource pool composition is a starting point. This is illustrated in Fig. 1.

Fig. 2 shows an illustration of the prioritized backlog together with an estimate on the pod role breakout and the estimated duration of the backlog item.

As inputs into the form shown in Fig 2: Backlog items are registered in order of priority in “Backlog Items.” Columns “QA,” “Data Architect,” “Software Engineer,” and “Solution Architect” denote the anticipated resource demands for each backlog item (i.e., the pod structure). Fractions of resources are allowed. Column “Effort” holds the anticipated effort associated with each item, measured in sprints for the sake of simplicity.

As outputs from the form shown in Fig 2: Column “Start” displays the actual starting sprint for each backlog item, and this field will be generated automatically considering the best start date using the employee’s resource pool. The final sprint for each backlog item will be also calculated for each item (column “Stop”) and is calculated from the data in columns “Start” and “Effort.”

Also note this data is based on the best available estimates at that moment and can be adjusted at any time as more accurate information becomes available. After each update to the core data, the forecast can simply be executed again.

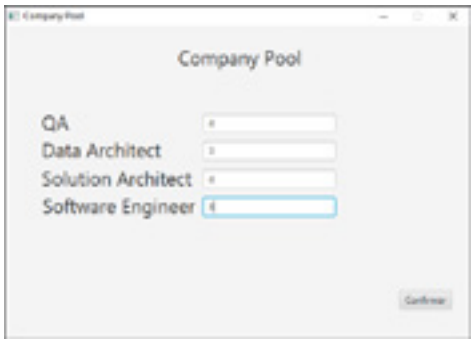

Fig. 1: Pool definition: The number of employees for each role in the project is registered. Pods will be constructed from this pool. Note that only four roles are considered in this example, but logically this can be extended to cater for more complex role compositions.

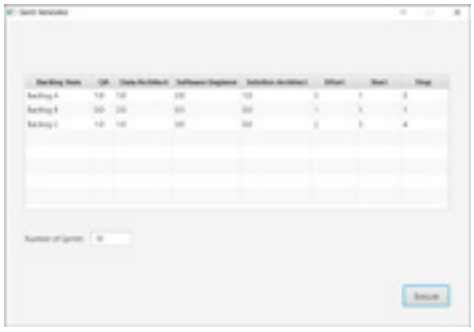

Fig 2. The application table that shows the Pod resource breakout and time (effort) required to complete each backlog item.

When the “Execute” button is hit, our solution attempts to assign pods in order of backlog priority, considering the match between the backlog item resource needs and resource role availability in the pool at that moment, and for a duration that is based on the number of sprints logged against the backlog item (“Effort”). As each sprint passes, the algorithm evaluates whether any items are scheduled to complete, and that will result in the pod handing its resources back to the pool. Such resources are logically then made available to other backlog items. Fig. 3 illustrates the solution logic in greater detail.

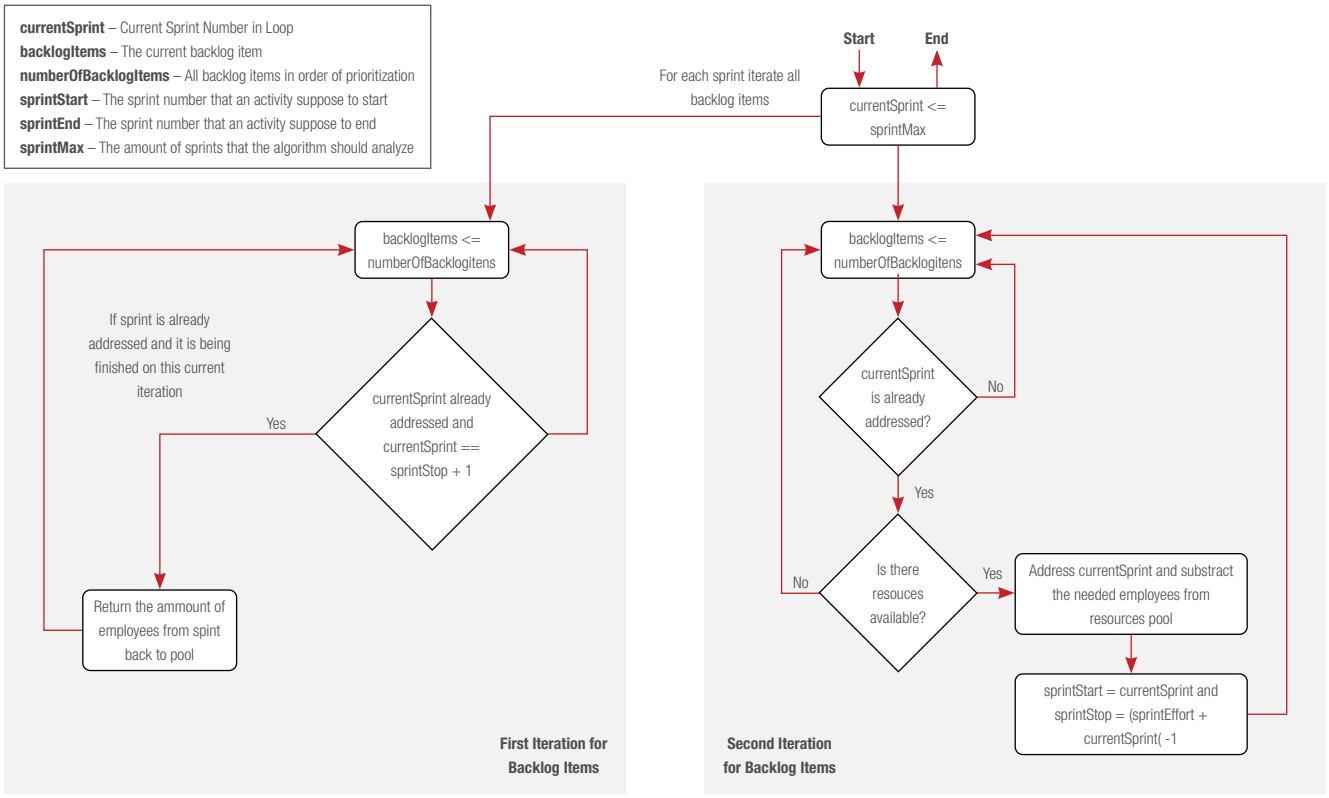

Fig 3: Flowchart that illustrates the assignment logic.

When the application completes execution, the recommended start and stop sprints are loaded to a Gantt chart, as is shown in Fig 4. This readout provides a view of the total duration needed to work through the backlog end-to-end and can help qualitatively identify bottlenecks in some of the backlog items.

Moreover, “what-if” analysis can be conducted to evaluate the effect of adjusting the pool size and/or role distribution, or the backlog prioritization has on the total number of sprints needed to execute the backlog.

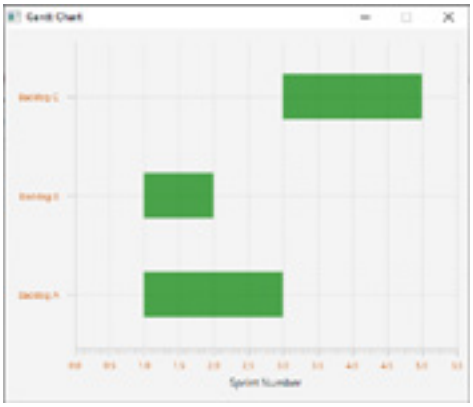

Fig 2. The application table that shows the Pod resource breakout and time (effort) required to complete each backlog item.

Solution extensions

The solution we describe above is basic and can be extended along several dimensions to make it more practically useful. We describe two such examples in the following sub-sections.

Backlog items are not always mutually exclusive
In the basic model, we assume there are no dependencies among backlog items, which is not always the case. For example, a backlog item that describes a pilot may need a proof-of-concept or case study to complete first, which is registered as a different backlog item. To address this, the solution can be extended with an additional column to denote where a backlog item has a dependency on another item. When the code logic executes, it will not assign a pod to a task that has a dependency on an item that did not complete or achieve a specified threshold of completeness. One example of such a threshold could be that 75% of the item`s sprints have already completed, or that it is currently in the last sprint.

One needs an exact match between resource needs (pod structural makeup) and what is available in the pool
A backlog item may need, for example, one architect, two engineers, and 0.5 testers. However, for example, the pool could have an architect and two engineers available, but not the tester. In the basic approach described above, an assignment will not be made as an exact match was not found. However, if we mathematically represent the pod needs vs. the closest available resources in the pool as two separate vectors, one can determine how closely the available resources match the requirements by means of a cosine similarity measurement. In the example above, this could be represented as:

Match = Cosine Similarity of [1, 2, 0.5] and [1, 2, 0] = 0.96

If the Cosine similarity value is greater than a set threshold (0.90, for example), the pod assignment will be made, and the missing resources will be prioritized to this backlog item for future sprints by creating a high priority pseudo backlog item. In the example we presented here, the pseudo item will a pod resource need of 0.5 QA.

Other use cases

The solution can also be extended to either suggest:

- an optimized backlog to minimize the execution time, given a fixed resource pool
- an optimal resource pool role distribution, given a fixed backlog and timeframe

Automatic prioritization of the backlog
We can extend the method to suggest how the backlog items should be prioritized to achieve the highest throughput (i.e., work through the entire backlog in the shortest time using fixed resources). This can be done by creating different prioritization scenarios and executing the forecast for each such scenario to suggest which prioritization sequence leads to greatest overall efficiency of the pool of resources.

However, since there are too many possible permutations, even for a backlog of 20 items, to analyze in a reasonable time, a smaller subset of scenarios should be pre-generated.

One such way is to break the backlog into a small number of clusters where each cluster aligns with business demand (highest priority, lowest priority, etc.), and for each cluster create a set of all possible scenarios. The number of permutations for each such cluster is n! where n denotes the number of backlog items in the cluster. For a cluster with 5 backlog items, the number of permutations is 5!, i.e., 120 possible scenarios that need to be analyzed to optimizes this cluster. This means that for a backlog with m clusters, the total number of permutations that will be analyzed are n1! + n2!+ … + nm!, where n1 denotes the number of backlog items in the first cluster, n2 denotes the number of items in the second cluster, and so on.

As a practical example: For a backlog of 10 items, without splitting the items into clusters, the number of possible combinations is 10! = 3628800. However, by splitting the backlog into two lots of 5 each, the total number of permutations that need to be worked through is only 5! + 5! = 240, thus reducing the work effort by more than four orders of magnitude. By then using, for each of the clusters, that sequence which

drives towards the smallest work effort we approximate the optimal item prioritization sequence.

Team structure Optimization

Finally, we can extend the method to optimize the pool structure (role distribution) and size to produce the desired throughput under a fixed time span. This use case can be useful for determining resource needs when performing project cost estimation exercises. By setting an initial default pool breakout and then tracking (while performing the analysis) which roles are bottlenecks when a backlog item assignment cannot be made, as well as which roles are always left with additional capacity when all other roles have been assigned, the algorithm can produce a readout of such under-represented and over-represented roles. These can then be used by the algorithm to tweak the team structure and/or size which will result in a more compressed timeline when the simulation is executed again. Through making such tweaks and executing the algorithm recursively, an optimal resource pool size can be determined to work through the backlog in the desired number of sprints.

CONCLUSIONS

Agile pods provide an elegant way to manage complex projects where backlog items vary in scope and the type and count of resources needed to address each item. A challenge with the approach is that it is hard to forecast when certain backlog items will be addressed, and indeed, whether the workforce will be sufficient to address all the backlog items over the duration of the engagement.

Our approach solves this problem efficiently and can easily be extended to deal with complex scenarios.

The method we developed is also useful in optimizing the utilization of the pods since it allows one to easily create “what-if” scenarios by changing backlog item priorities or determining what the effect would be if the original pool role breakout and/or backlog item role assignments are adjusted

AUTHORS

Gerhardt Scriven, Principal Consultant
gerhardt.scriven@capco.com

Diego Sarai, Principal Consultant
diego.sarai@capco.com

ABOUT CAPCO

Capco, a Wipro company, is a global technology and management consultancy specializing in driving digital transformation in the financial services industry. With a growing client portfolio comprising of over 100 global organizations, Capco operates at the intersection of business and technology by combining innovative thinking with unrivalled industry knowledge to deliver end-to-end data-driven solutions and fast-track digital initiatives for banking and payments, capital markets, wealth and asset management, insurance, and the energy sector. Capco's cutting-edge ingenuity is brought to life through its Innovation Labs and award-winning Be Yourself At Work culture and diverse talent.

To learn more, visit www.capco.com or follow us on Twitter, Facebook, YouTube, LinkedIn Instagram, and Xing.

WORLDWIDE OFFICES

APAC
Bangalore
Bangkok
Gurgaon
Hong Kong
Kuala Lumpur
Mumbai
Pune
Singapore

EUROPE
Berlin
Bratislava
Brussels
Dusseldorf
Edinburgh
Frankfurt
Geneva
London
Munich
Paris
Vienna
Warsaw
Zurich

NORTH AMERICA
Charlotte
Chicago
Dallas
Hartford
Houston
New York
Orlando
Toronto
Tysons Corner
Washington, DC

SOUTH AMERICA
São Paulo

WWW.CAPCO.COM

CAPCO
a wipro company