Table of Contents

Troy Magennis - Solving the Hairy Problem of Team Dependencies

Premise

When portfolio and program managers undertake quarterly (or annual) feature and portfolio planning, understanding team dependencies is made necessary to identify constraints and avoid overburdening one team. Complexity quickly grows beyond intuition after two or three team dependencies are identified, rendering current forecasting techniques unsatisfactory. This session will discuss the inadequacies of current tools and frameworks used to manage the dependency planning problem, and introduce the choices you have for reducing the complexity and new planning techniques that untangle dependencies into a doable plan.

At the end of the session, attendees will have a new understanding of the complexity team dependencies add to planning, and have a set of strategies and techniques to predictably manage products built in high team-dependency organizations. This session looks at example dependency graphs and graphical matrix techniques that are quick to build and give clear risk insight.

It often shocks organizations to learn mathematically that each team dependency HALVES the chances of an on-time completion of component or delivery. With two dependencies, there is a 1 in 4 chance of no delay; with three dependencies, there is a chance of 1 in 8 delivering on-time. One large legacy application the author worked with had seven dependencies from a core library to a user interface – that is a 1 in 128 chance no team will be delayed (127 times more likely to experience one or more delays). Planning clearly needs to consider how dependencies might impact each teams ability to integrate and build.

It is not as dire as it sounds, not every team suffers the same chance of delay. We look at how to analyze historical examples of delayed work to identify types of features that will encounter dependency delays in the future. Building a map (linked graph) and matrix visualizations of team dependencies gives a basis for examining this historical likelihood of delay and planning team organization structures or staffing plans that compensate. It is possible to predictibly plan in high dependency environments, its just too hard to do in ad-hoc ways.

Learning Outcomes:

Understand the impact of multiple team dependencies on planning and scheduling predictability Introduce ways to identify and visualize dependencies for planning Understanding current strengths and weaknesses of dependency management approaches New strategies for minimizing the impact of dependencies and planning cross team capacity

Summary

- Content rating (0-no new ideas, 5 - a new ideas/approach, 9-new ideas): 9
- Style rating (0-average presentstion, 5 - my level, 9-I learned something about presenting): 7

Action / Learning

- I think this is important as it shows how choices for execution is increasingly limited as you have more dependencies.
- Need to write related article to “just don't do it” that I have written about before - see Blog post on dependencies

Presentation

agile_2015_-_entangled_-_solving_the_hairy_problem_of_team_dependencies_troy_magennis_.pdf From Sim Resources

Paper references:

- A Taxonomy of Dependencies in Agile Software Development
- Reference to paper "The Effects of Team Backlog Dependencies on Multi-team Systems. Also see Slideshare.net

Notes

One thing that hasn't changed since 1990 is dependency are problem and it hurts

Dependency - one block, waiting on another (start work, do work, waiting in equipment)

See taxonomy of dependencies paper

Knowledge dependency Dependency on expertise - not resources

Task dependency

Resource dependency Entity and technical (not people)

Block progress

Breaks out flow in projects Every dependency we have reduces options of when to start work Increase lead time Creates friction between teams

Reduces freedom

Dependencies in backlog Order that you start work is limited 1 dependencies halves the prioritization

See chart - one dependency cuts options in half Two deep cues cuts options to 1/6th No matter how big the backlog

4 dependencies you are below 1% in choice of order to do

Any way you can have no dependency is a good thing Not possible to get to zero, but easy to get to 1

Increased lead times of work items

Chains of events Team dependency diagram 7 layers of dependency - 1 chance 128 Remove dependency - half chances of being delayed

Leads to these long tail cycle times

How forecast

Estimate by time = optimistic + 4 x most likely + pessimistic Most likely shouldn't be waited

Delays in flights had nothing with the flights Delays are et caused by the work we are trying to estimate

Create a chart of team dependency - who do you depend on

Another approach Estimate = sum sprints x 1.5

Question - when would it get to the customer Look at longest path (for sprints)

Optimal All work 1 sprint 7 x 2 X 1.5 21 weeks

50% teams will miss expected times - asynchronous sprints 27 weeks - ie 1/2 a year to get something that at the out edge

People miss the changes that I do impact others - ie testing Worry about your test dependencies

Team members

7 +- 2 George Miller - Miller Law Justified based on number of communication lengths N(n-1)

What about more than 7 teams

Larry Macherone - Rally etc Performance vs team size Higher is better Best performing team 5-9 in

Increase team size Performance drops Predictability increases (not waiting on skills) Responsiveness / quality same

Consider up to 15 people to decrease dependency (last resort) Keep small if team performance > system performance, performance > predictability, team coordination cost low (eg great at scrum of scrum)

Team organization Merge teams Create multi discipline feature team Co locate teams (not remote person where dependencies)

Do something temporarily to allow you to move forward (deferring dependency)

Visualize dependency and group

Component vs feature teams Pros and cons Component teams create dependency Feature teams integration / consistency harder

Skill levels Can teach others Can do Have a desire No idea and never will

Role of manager Skills needed in the future Make sure we have them when we need them

Growing teams Start with a team Need new people in one skill Find trainer on team for this skill Take on new skill in work Now backfill with new hire Then split team

This is how you generate T shaped people

Backfill for new hires No more than 30% for a team

Can now plan this process

Two step skills matrix Ask about expertise Then ask about how desirable they want to do this

Shows whether you have single point of failure If you are ok Or need to do significant training / work

Incentives

Most teams incentive is make a feature get a cookie Need to make sure incentive schemes are the same so can make decisions locally

Friendly Competition aligned desire outcome

Wrong order list Wrong order'o'meter concept

Desired order vs not order We want feature 1 and 5 Green if order, light green if got something, grey if not in right order

Forecast, Conference, Planning, Dependencies