Usage: prince [OPTIONS] doc.html Convert doc.html to doc.pdf prince [OPTIONS] doc.html -o out.pdf Convert doc.html to out.pdf prince [OPTIONS] FILES... -o out.pdf Combine multiple files to out.pdf Try 'prince --help' for more information. Usage: prince [OPTIONS] doc.html Convert doc.html to doc.pdf prince [OPTIONS] doc.html -o out.pdf Convert doc.html to out.pdf prince [OPTIONS] FILES... -o out.pdf Combine multiple files to out.pdf Try 'prince --help' for more information.

Skip links

- Skip to primary navigation
- Skip to content
- Skip to footer

- Education
- Coaching
- Articles
- Testimonials
- Calendar
- Contact us

Toggle search Toggle menu

Michael James (MJ)

Guy who simplifies your organization.

Follow

- Seattle, WA
- Twitter
- LinkedIn
- YouTube

Large Organization Software Development Misconception #1: Are “Dependencies” In Knowledge Work Caused By Immutable Laws Of Physics?

2 minute read

Background

Henry L Gantt

Just before World War I, a colleague of Frederick Taylor named HL Gantt promoted the use of a project schedule chart highlighting activities that depend on each other. This can be useful in manufacturing, construction, logistics, and other fields where the dependencies are imposed by physical reality.

(Interesting trivia: the Soviet Union tried to use a Gantt chart for it's first five-year plan.)

SAFe yarn Unfortunately, people who haven't discovered that knowledge work is different from traditional work misapply these techniques to software development. Software development companies steeped in the project management mindset sometimes try to plan around perceived dependencies with traditional Gantt charts, or sometimes by modeling them with red yarn. The yarn method is only superficially different than a 100 year old Gantt chart, yet it's still being sold as an "Agile" technique today!

Physical analogies for software development (and other knowledge work) are mental traps

The following real quote reveals the mindset of a project manager with tools to manage dependencies in physical work – where they actually exist – but has not written software in a modern way:

> People who think that dependencies are just figments of old-style thinking should convince me by first putting on a shoe and tying the laces, then putting on a sock that goes between their bare foot and the tied shoe without removing the shoe.

In software development, asynchronous "dependencies" are not caused by immutable laws of physics!

The illusion that we need to plan around "dependencies" through multiple Sprints is a symptom of obsolete organizational structure, policy, and skills. We gain agility by directly addressing those problems rather than institutionalizing them with Gantt charts or red yarn. The socks and shoes argument inadvertently demonstrates that "dependencies" in software development *are* figments of old-style thinking.

Ten years ago I'd usually hear the same misconception expressed as a *house analogy*: "If you want to build a house, you have to build the foundation first. You can't start with the roof." And this may be true … about houses.

The fundamental constraint of knowledge work – such as software development – is our ability to discover and create knowledge.

Imagine that you had spent Monday through Thursday working on an essay, a song, or even a computer program. You tore up seven ways that you realized wouldn't work. You took a walk outside. You slept on it. You did research and found an eighth way that looked like it would work, until you showed it to a friend who pointed out a serious flaw with one part of your idea. By Thursday afternoon you had nearly finished a ninth way that combined the best of your previous approaches and looked to be nearly done.

Then Thursday night your cat walked across the keyboard and erased everything you typed that week.

Would it take you another four days to get back to where you were? Of course not! By 10AM Friday

you'd be ahead of where you were Thursday night. Therefore what was the actual constraint on that knowledge work? It wasn't the typing. It was the learning and creation of new knowledge.

Perceived "dependencies" in knowledge work are caused by knowledge gaps.

Once we internalize this realization, it leads to actions which were counterintuitive before – often the opposite actions that project management would advise. Unfortunately your organization's structure and policies have actively encouraged these knowledge gaps.

A skillful software development organization allows the Product Owner to *prioritize* the Product Backlog from a business perspective, not just "order" it. Put away the red yarn.

Updated: August 07, 2019

Share on

Twitter Facebook LinkedIn
Previous Next

You may also enjoy

Nine Disadvantages of LeSS, From Someone Who's Doing It

2 minute read

Why Write This? You may know that I helped Manoj Vadakkan establish Fans of LeSS, our attempt to un-hijack Scrum and Agile. But just as Ken Schwaber used to...

Please Hold On To Your Wallet When You Hear Someone Claim That Scrum Scales Linearly

1 minute read

Be skeptical of anyone who tells you that nine Scrum teams will finish your product in one ninth the time. All else being equal, it's not going to happen.

Why We Do Not Recommend Trying To Break Products Into

Small Independent Pieces For Small Teams

1 minute read

Just today I heard two people give the same naive scaling advice: divide products into independent pieces for different teams to work on. This advice is so ...

An Example Dysfunctional Contract Relationship

1 minute read

Q: I am having a struggle with our contractor over scrum meetings. I am proposing that only the scrum master (me) and the developers (contractors) attend a s...

Enter your search term...

- **Follow:**
- Feed

Michael James (MJ)

Guy who simplifies your organization.

- Seattle, WA
- Twitter
- LinkedIn
- YouTube

Large Organization Software Development Misconception #2: Are All Teams Working On Equal Value Stuff?

2 minute read

Product Owner Misconceptions Page 16 If someone in an seven-team company had only a superficial understanding of Scrum, they might think each team should have its own segregated list of stuff to do. That keeps them "productive," right?

This is a great idea if your goal is to keep everyone busy. If your goal is to do the highest priority, highest value work, it could only be justified if several implausible things were true:

1. Each team's backlog would need to contain the *same priority work*, work of equal value. Team A's #1 item (A-1) would need to have the same customer impact as Team G's #1 (G-1) item. If you've spent much time with your eyes open in real organizations, you already know that one

team's work will be more important in the big picture than another team's work. In real life it's not hard to imagine the value ratio exceeding 1000 to 1. The problem will be masked if we have single-function teams (e.g. design only, test only) or *component teams* (e.g. back end only) rather than feature teams.
2. Each team's sub-list would have to *stay* the same priority over time. Even as we learned more about customer needs, we would not be able to reprioritize beyond the limited context of each team's list. We'd have to decide up-front, once and for all, the value of each team's type of work. This type of "Scrum" would work if we didn't need agility. Of course we could cheat and move items between Team C and Team D's lists, but that would be admitting that it's really just one list we've artificially segregated. (The more common form of crypto-reprioritization is to yank key people from one thing to another: *resource management*. This habit is so widespread, it should be a clue that superficial-Scrum actually *reduces* agility, and managers need a way around it!)
3. Each team would have to complete every item at the same rate. If Team A started #A-3 before Team B started #B-2, they would be working on the stuff we'd declared lower value at the expense of higher valued work. Have you ever gotten in line at a grocery store and noticed that latecomers in a different line finished before you?

Product Owner Misconceptions Page 17

These are the points that people who advocate the so-called "Spotify Model" (aka. Shiny Happy People Holding Hands) are usually missing: there's a difference between *feeling* productive, and actually doing the highest value work. Employee satisfaction is a potentially useful indicator. But by itself, employee satisfaction isn't evidence that the product development organization can learn and adapt to reality as it's discovered. A happy company that cannot pivot is not Agile.

There may be other justifications for keeping separate lists. Maybe they really really really are unrelated products, though we should be skeptical of this. Maybe we're bound by overspecialization because the skills and knowledge gaps seem insurmountable. Maybe we've made a management decision to keep our less capable developers on Team F. Maybe Teams X and Y are on the other side of the world and we're not willing to give them anything important. Let's use Ken Schwaber's term for these things: *organizational impediments*.

Gentle reader, I have no idea what you should do in your own situation. I hope I've at least helped you gain clarity about why Scrum has only one Product Backlog per product, regardless of the number of teams.

Japanese version: 大規模組織におけるソフトウェア開発の誤解その２:すべてのチームが等しい価値の業務に取り組んでいるか？

Updated: August 08, 2019

Share on

Twitter Facebook LinkedIn
Previous Next

You may also enjoy

Nine Disadvantages of LeSS, From Someone Who's Doing It

2 minute read

Why Write This? You may know that I helped Manoj Vadakkan establish Fans of LeSS, our attempt to un-hijack Scrum and Agile. But just as Ken Schwaber used to...

Please Hold On To Your Wallet When You Hear Someone Claim That Scrum Scales Linearly

1 minute read

Be skeptical of anyone who tells you that nine Scrum teams will finish your product in one ninth the time. All else being equal, it's not going to happen.

Why We Do Not Recommend Trying To Break Products Into Small Independent Pieces For Small Teams

1 minute read

Just today I heard two people give the same naive scaling advice: divide products into independent pieces for different teams to work on. This advice is so ...

An Example Dysfunctional Contract Relationship

1 minute read

Q: I am having a struggle with our contractor over scrum meetings. I am proposing that only the scrum master (me) and the developers (contractors) attend a s...