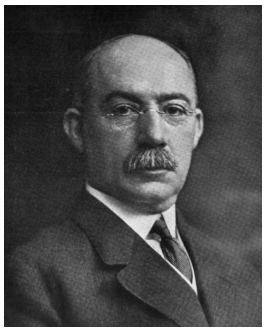
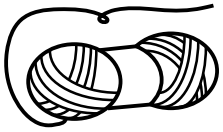
Large Organization Software Development Misconception #1: Are "Dependencies" In Knowledge Work Caused By Immutable Laws Of Physics?

Background



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.)



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.