# Kanban for your team

Tomek Kaczanowski

# Table of Contents

# Chapter 6. Work in Progress

> Don’t spread yourself too thin.
>
> — Common knowledge

WIP, or *Work in Progress*[1] is the number of tasks that the team is currently working on.

All the tasks which have been started, but have not been completed yet, increase the WIP. Thus, if the first column on our board is *ToDo*, and the last one is *Production*, then WIP includes all the tasks that have left *ToDo*, but have not been deployed to production yet. It does not matter whether we are actively working on them, or whether they are waiting for the QA team to have sufficient bandwidth to take care of them. This is not important. What matters is that the work has been started, that they are "hanging over our heads", that they cannot be crossed out and forgotten.

> Unclosed loops
>
> Your question could be what the cost of tasks which have not been completed and are still "hanging over us" is. None, in theory, quite big in practice.
> These are like the unclosed loops from the Getting Things Done. They do not seem like much, but in an imperceptible way they are taking away some of our bandwidth. We are not doing anything with them, but we are still thinking about them and getting frustrated when, time after time, we realize that this task is still not completed.
> In the case of development, the cost of such unfinished tasks may even be higher because they usually have to do with unfinished code, which lies somewhere in the repository, perhaps making work on the next tasks more difficult.

## 6.1. Penalty for high WIP

Thus, each open task leads to some additional work and to a number of problems. Let’s have a look at them.

[1] or *Work in Process*.

- **Context switching**. With many tasks pursued in parallel, we are forced to switch between them. Every time we do this, we are losing time. That's just the way we are – "entering" a new topic fully can take as much as 15 minutes. In addition, we have to record (in memory or in writing) where we interrupted the previous task.
  Every now and then we will be making mistakes: we might think that we have already done some things (when in fact we were just thinking about them), or that some things have only been planned (while we have actually done them). In both cases, our work will lack a steady rhythm.

- **Re-discovering knowledge**. Each time we are returning to an open task, we must understand its nuances again. Depending on how complex the task is, and the time that has passed since we last worked on it, this could take a moment, or much longer. The real problems begin when we are continuing a task that was started by another person…

- … because that's where we are hitting the **knowledge transfer** problem. Experience shows that some things are not transferable, and no type of documentation can really help with that. If we are lucky we might get a chance to talk to our predecessor, but even that will not allow us to get all the knowledge. And we will have to reinvent something that has already been invented.

- The more tasks being done in parallel, **the more delayed their completion will be**. And we will deliver them to the client later. And this results in:

- **Delayed feedback**. The later we present a demo or deploy to production, the later we will learn what the client thinks about a given fragment of the system under construction. And, in the case the client is not fully satisfied, we will pay a higher penalty than we would have to pay if we had allowed the client to review our work earlier. There will be more to change, redo, and rethink.

- As the development time increases, the risk increases that changing business requirements will **reduce the importance of the task for the client** (vs. earlier delivery). In extreme cases - when a deadline is exceeded - the task may lose all its value altogether.

Imagine a dual carriageway with five cars. All of them are driving quite fast and will reach the target soon. Now imagine

there is fifty of them. The resulting traffic jam will make the journey time much longer for all of them. This is exactly the same with the team of programmers – the more tasks are being worked on, the longer it will take to complete them.

## 6.2. Morale losses

Until now, we were thinking about the impact of the high WIP on project completion. It seems that delivering it on time may be at risk when the team's efforts are spread onto too many parallel tasks. Let's now have a look at the impact of WIP on the team members' morale.

Let's start with a short quote from the personal development guru Brian Tracy:

> Each time we complete a task we get an influx of energy and enthusiasm, and our self-esteem increases considerably.
>
> — Brian Tracy *Eat that frog*

When WIP is high, every team member is occupied with more than one task during the day. And this means they will complete only a portion of each. So you can forget about the enthusiasm influx related to task completion.

Of course, such a situation is not without impact on the team member's mental states. Working on one task, you are thinking of another (which is also urgent and important), which leads to stress. When several things are being done at once, usually it is not possible to complete any of them. Hence the feeling of lack of achievement at the end of a working day: so much work but nothing really done. Add to this the constant context switching and having to remember things related to both the first and the second task (and perhaps also the third one and the fourth one). It is hard not to feel overwhelmed and not to get frustrated. None of these emotional states helps you work more productively…

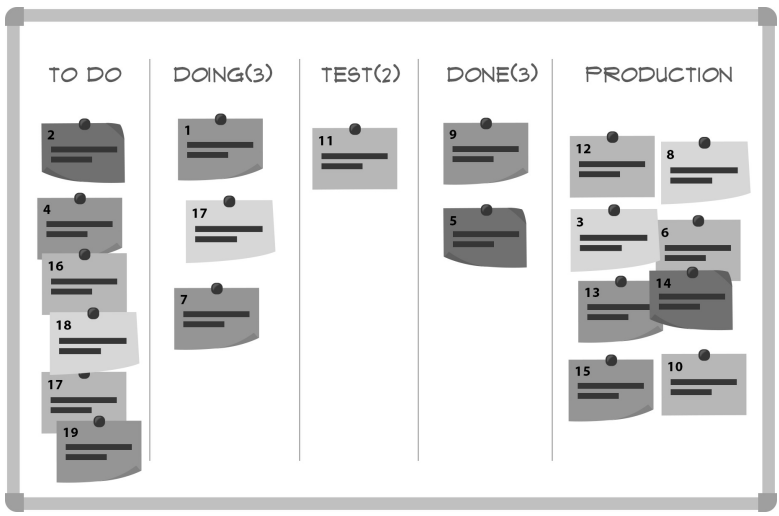
What is more, the client is frustrated as well! And so is your boss. And his wife, because she is annoyed when he is getting home and he pays no attention to her new haircut. There's a storm brewing!

# 6.3. WIP limits

What we said above clearly shows that a large number of tasks open simultaneously is not good. High WIP leads to time losses and has a bad impact on the employees' mental states. No wonder than that one of the major goals of Kanban is *limiting WIP*.

One commonly used method of pressurizing the team to limit WIP is imposing restrictions on the number of tasks that can be put in the individual columns of the board.
These are usually marked on the board by putting a number in brackets next to the name of a given column. These numbers set the limits, i.e. the maximum number of cards that can be put in the given column.

Let's have a look at the board shown below. The following limits are marked there: 3 for the *Doing* and *Done* columns and 2 for the *Test* column. They determine the maximum allowed number of cards in these columns.




The setup of the cards on the board indicates that no card can be added to the *Doing* column (because it already has 3 cards, i.e. equal to the limit). So, what should a developer do while looking for a new task upon completing a previous one? He should think about working on one of the tasks that can already be found in the *Doing*, *Test* or *Done* columns.

Note that this exactly repeats the considerations that we were engaged in when deciding how to deliver value to the client as quickly as possible.

The difference is that at that time we were appealing to the developer’s common sense, while here we have a clear delineation of his freedom. Sufficiently low WIP limits force team members to share work on tasks, to unblock blocked tasks and to work in pairs.

WIP limits are not meant to be a spiteful interference in work. They are meant to put pressure on the team to increase cooperation, to take responsibility and, in effect, to complete tasks earlier. They are meant to chafe in order to bring the expected results.

The limits, being put on board and thus clearly visible, are an example of the implementation of the Kanban principle: *"let the rules be simple and visible"*.