# When to Think about Shortening Your Sprint Length: A Toolset

## by Alexey Krivitsky and Ragnar Birgisson

There are multiple cases that come to my mind, when one should think critically about the current length of sprints:

- ***CASE "status reports"**:* customers ask for status reports in the middle of a sprint (usually a case for outsourced projects, especially during the start-up phase)
  - *CAUSES:*
    - lack of trust to the development team
  - *POSSIBLE SOLUTIONS:*
    - invite the Product Owner and other interested people as "chickens" to the daily Scrums (sometimes this may not be possible due to the language barrier);
    - share with them the burndown charts, so they have the same information and feeling of the progress as the team members do;
    - shorten sprint duration, so that you demonstrate the working software more often (this can hardly happen daily as it would have been with status reports, but anyway, agree that the presence of a status reports in a manager's mailbox cannot remove her doubts and build up trust); after 2-3 successful sprints, she should start trusting in your team and as a consequence should stop asking for the status reports.
- ***CASE "ever-changing priorities"**:* management tends to change priorities of the planned work items inside sprints or even already have stopped several sprints in the past
  - *CAUSES:*
    - the sprint is too long so that it is quite risky or impossible to predict the business needs and priorities for too far in the future
  - *POSSIBLE SOLUTIONS:*
    - shorten sprint duration, so that they wouldn't have to decide for too in advance anymore (for different projects working in different domains the definition of the "future", of course, differs, but two-week time should be OK for most of the situations, otherwise there is something wrong with the company's business strategy if they cannot build a solid plan for the next few weeks…)
- ***CASE "too much is in work"**:* too many developments tasks are 'in work' by the team so that interruption overhead becomes an overkill (you shall learn this very soon from the developers on the daily Scrums or retrospective meetings if this is case for you)
  - *CAUSES:*

    - the sprint is too long compared to user story size, so there is too high number of user stories planned for it, so that the product management makes lots business decisions causes fixes and re-fixes and this lasts until the sprint is finally over

  - *POSSIBLE SOLUTIONS:*

    - combine user stories making them bigger, but bare in mind that it will probably make it harder to estimate them;

    - shorten sprint duration, so that there are less user stories 'in work' in a sprint and the business decisions affecting the stories can be postponed until the next sprint (this is actually the reason why we do iterative development – to avoid too many changes occurring simultaneously)

- ***CASE "too much to change"****:* too many process improvements are identified after sprint is over, so that it is hard to implement them all in the next sprint

  - *CAUSES:*

    - the process or the project environment is not stable enough and the sprint length is too long to provide capabilities for the team for in-time project steering (process improvements are recommended to be applied from the start of a new sprint, not in the middle of the current one)

  - *POSSIBLE SOLUTIONS:*

    - shorten sprint duration, so that the changes can happen more often due to more frequent sprints

- ***CASE "too high risks"****:* level of project uncertainties/risks is too high

  - *CAUSES:*

    - it can be either related to unknowns in scope questions, technological aspects, buffer reservation concerns, etc – usually a case for a start-up phase of a project

  - *POSSIBLE SOLUTIONS:*

    - shorten sprint duration, so that the number of risks that can happen within the fixed period is decreased and the level of uncertainties becomes controllable allowing the team members to provide the estimates they are comfortable with;

    - development tasks with quite high level of uncertainties can be broken into two parts (a spike and development itself) that are planned for serial sprints limiting under/over-estimation risks – usually such tasks are the ones that to be started earlier in the project and obtaining the spike results sooner is preferred (may not be possible with longer sprints)

- ***CASE "too many bugs":*** level of bugs is too high by the end of each sprint

  - *CAUSES:*

    - the quality of the project is not under team's control (because the team is new to the domain or/

and the project deals with rework of legacy code without enough coverage of automated testing)

- *POSSIBLE SOLUTIONS:*
  - implement unit-testing, automation testing
  - reserve time and resources inside each sprint for fixing introduced and discovered issues (it is known that the sooner the issues are fixed after they have been introduced or discovered, the less effort is required)
  - shorten sprint duration, so that less code is reworked each sprint potentially triggering less issues to show up; also it should make the quality reviews (feedback loop) happen more often and the found bugs can be faster planned for the next sprint to be worked on

- ***CASE "delayed feedback"****:* client's feedback doesn't come in timely manner
  - *CAUSES:*
    - there can be plenty of things causing delayed feedback starting from a lack of resources on the client site, to broken communication between the development team and the clients
  - *POSSIBLE SOLUTIONS:*
    - force communications by using lighter communication channels: phone calls, video conferences, instant messaging
    - force building domain expertise within the team by switching to user stories, or, if already using them, leave more undefined points for them so there is an evident need for discussion
    - shorten sprint duration (so that the sprint demos are happening more often) which should push the clients to more often decision making, increasing the understanding of what the software being built should be

- ***CASE "planning difficulties"****:* clients have difficulties fulfilling the product backlog or/and the sprint planning sessions turn into never-ending meetings
  - *CAUSES:*
    - this may happen by the end of major development phase
    - the business situation changes unpredictably
  - *POSSIBLE SOLUTIONS:*
    - shorten sprint duration, so that smaller amounts of work items are enough to start a sprint

- ***CASE "drifting in the middle of a sprint"****:* there is a tendency for the team to start drifting in the middle of a sprint (should be visible as flattening of sprint burndown charts)

- *POSSIBLE SOLUTIONS:*

  - shorten sprint duration, so the team stays focused by feeling that the last days of the sprint are never too far in the future (for bi-weekly sprints, the demonstration day is always this or the next week)

There are cases when increasing of sprint length seems to be the right decisions, however, in most cases it is not the only possible one, consider the following example:

- **ANTI-CASE "no stable version"**: The overhead of making a stable version of software is too high, so that it affects overall development performance.

  - *POSSIBLE SOLUTIONS*: Increasing spring length can feel like the right solution, however, it can also make the things worse, since inability of the team to produce deliverable code each week or two can be caused by poor quality, too big user stories, lack of skills, or lack of build/test automation.

The causes listed above may be inapplicable for your situation (as, for example, the team is just facing complex tasks that cannot be slit), however there should be lots of far better solutions to try rather than increasing sprint length, as it will fix only the consequences, not the causes.

Here are some more hints on actions to be tried out prior to changing sprint length:

1. **ANTI-CASE "too big stories":** If you are to increase sprint length because the new user stories placed into the product backlog seem to be bigger and more complex than then ones you did before – try to sit with your Product Owner, users or user proxies and rework the stories so they are made more granular. Be aware, however, to avoid making them too cross-dependent (there is always a correlation between granularity and dependency).

2. **ANTI-CASE "moving bigger steps":** If it is your management that wishes to boost your team's performance and 'move with bigger steps' – try to explain them that doing less work in shorter iterations doesn't mean working slowly, prove that by playing a fake release planning session with twice longer sprint length.

3. **ANTI-CASE "work didn't fill into sprint":** If the team stops filling in into the sprint's length and tends to slip some of the sprint's backlog items – let them fail, learn and select fewer stories the next sprint (even if they have to fail once again before they find the right velocity to adjust planning with).

4. **ANTI-CASE "let's doing it right":** If your management wishes you to develop a set of functionality in a single but longer sprint – don't let them blind you, because you know that all the good things have been reworked at least once (even the Bible describes six one-day iterations…)