# Analysis of P22 project data

# Overview

This project data was made available by a company that wishes to remain anonymous. The conversation I had about the data analysis was with Mr A., was a developer working on the project.

This README is an edited version of the document used to log the analysis discussion between me, Derek Jones, and Mr A.

The project developed web based software to replace existing functionality, implemented by three people over eight months.

Jira was used to manage Stories.

The data contains 630 rows, of which 325 relate directly to the implementation. Non-implementation Stories include: on/off boarding employees, doing end of season tasks for the sales people, and just generally keeping the websites running.

## Summary

In summary, what would you say was the most useful part of the analysis, to you?

Mr A. The most useful part of the analysis? I think it was great to get an outsider's perspective on the data.

## Individual performance

While different people exhibit different levels of performance, project data is very unlikely to contain the information needed to answer questions relating to individual performance.

Unless the same, or very similar, Stories are implemented by different people, it's not possible to compare individual implementation performance. Differences involve lots of confounding factors, e.g., did a developer work on the harder items, were some developers interrupted more often (which slowed them down), was the implementation similar to work that a developer had recent experience (past experience is a big predictor of time).

Claims of evidence for factors of ten performance differences are sometimes made. This evidence derives from misinterpreting study data (e.g., the Grant Sackman study, see section 2.8.7 of my Evidence-based software engineering book), or based studies involving single, short experiment (e.g., time taken by slowest subject was an hour).

## Date/time data

Various columns contain date information:

- When a Jira story is first created it is given a number, this number is given by the *Branch* column. The relative ordering of these numbers does not provide any information about the order that Stories

were implemented. Stories are sometimes created long before they are estimated but they are almost always estimated before someone starts working on them (99% of the time).

- Various information specific date fields appear in the data. For instance: The *Updated* column lists the time that the Story was last updated, and the *Date Reported* column lists the data a defect was reported.

## Estimates

Stories are estimated in story points. Our definition of a Story Point is close to what other authors call an idealized engineering hour.

Stories created weeks before they will be actioned are often created without estimates. We gather up a group of candidate stories for the next sprint a few days before the current sprint ends and start making a plan of what we want to work on and who will do that work. We negotiate who will do what. The assignment of stories is usually straightforward and is based on who has been working in that area or who we want to get experience in a new area or with a new kind of work. Sometimes stories can be done by anyone, which works out well to balance out the quantity of work assigned to each dev for the next sprint. If someone ends up with too much or too little work, I'll step in to balance things out. People usually volunteer to take or give up work to balance the sprint; it's never a big deal.

The person doing the work picks an initial estimate. But, in the sprint planning meeting, anyone can object to any estimate. That 'anyone' is usually me encouraging the other guys to increase their estimates (my guys are eternal optimists).

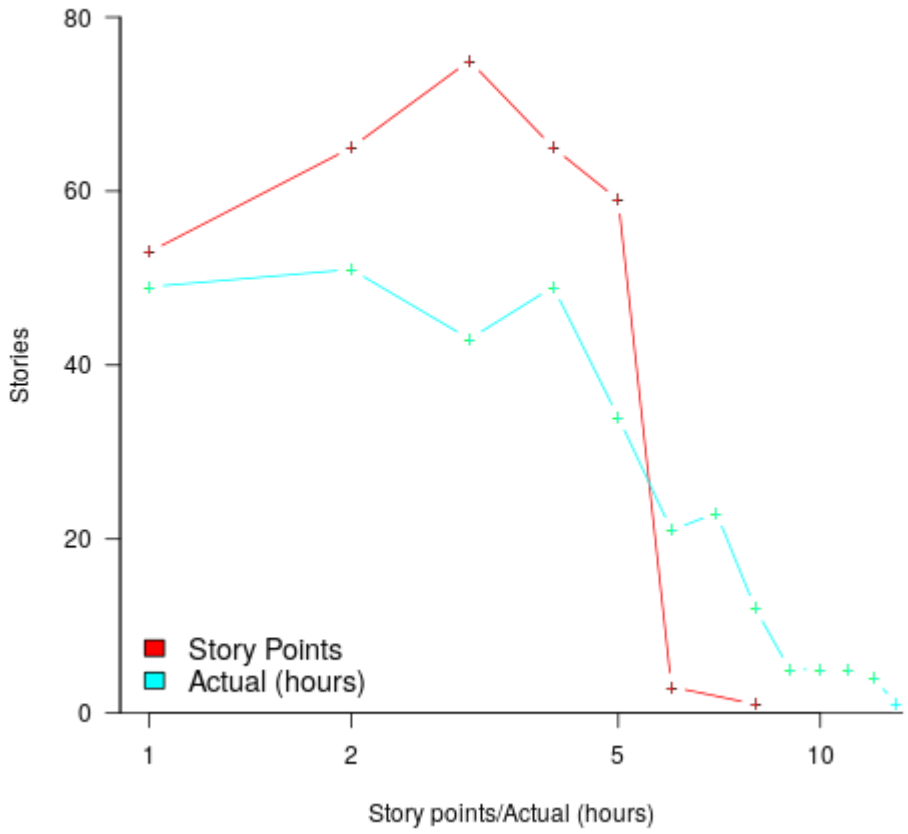
For non-programming stories we don't actually aim to hit the estimates for backlog grooming or meetings or even estimate them very carefully. Everyone gets assigned x hours for backlog grooming for example but the effort is usually spread unevenly among the team.

We care a little about estimation accuracy but it's just a means to an end (to make sure we can make reasonable predictions about when the project will end and make adjustments if we are at risk of being late)

# Modeling actual implementation time

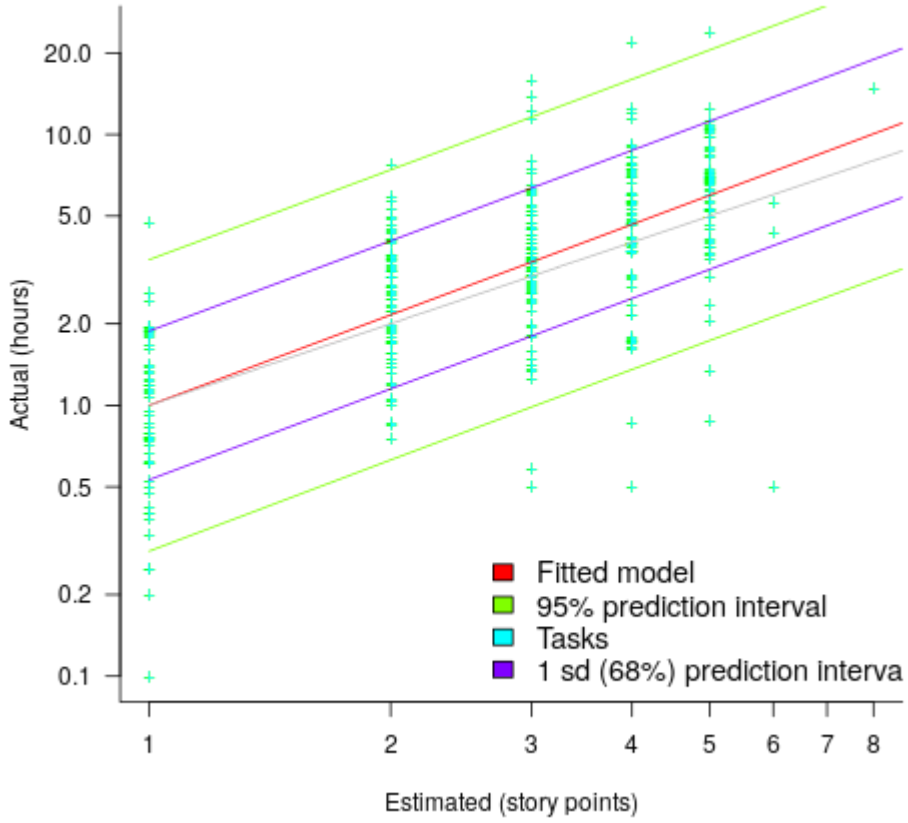
## Distribution of actual/estimate values

The following plot shows the number of Stories estimated to require a given number of story points (red), and the number of Stories whose implementation took a given number of hours (blue/green; actual time rounded).




## A regression model of implementation time

The plot below shows estimated story points against actual time (i.e., developer time plus review time), with the grey line showing `Estimate_story_points == Actual_hours`.




The red line shows the fitted regression model (variance explained 46%): $Actual=1.01*Estimate^{1.09}$

Estimates within 1 standard deviation of the fitted model mean were within a factor of 1.9 of the actual hours, while 95% were within a factor of 3.4.

The analysis of other estimation data finds corresponding multiplication factors of two and four. https://shape-of-code.com/2022/06/19/over-under-estimation-factor-for-most-estimates/

The following table shows the number of developers working on a given Story:

```
Devs        1   2   3
Stories   140 154  31
```

Including the number of developers who worked on a story improves the fitted model (variance explained 51%):$Actual=0.6*Estimate^{1.07}e^{0.3TeamSize}$

The following table shows the number of Stories that had a given number of developers working on them and the corresponding number of story points:

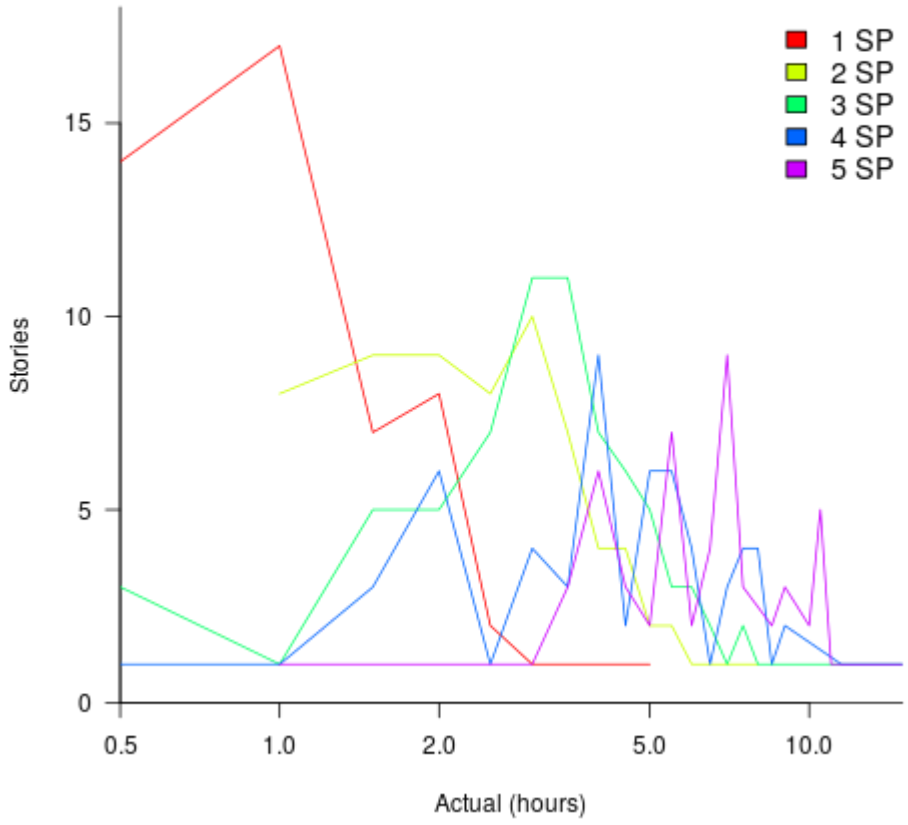
```
            Story points
         1  2  3  4  5  6  8
      1 23 28 31 38 17  3  0
Devs  2 31 31 33 21 37  0  1
      3  0  6 13  6  6  0  0
```

Based on their summary description Stories were manually categorized by activity, and the following table shows the number of Stories in each activity:

```
    None     coding     requirements/design   testing/debugging
      93        187                      17                  29
```

Adding this `story_kind` column in to the estimation regression model found that this information was not statistically significant.

The following plot shows the number of stories implemented in an a given amount of time (rounded to half-hour), broken out by estimated story points.




While the mean actual hours may be relatively close to the number of story points, the distribution is spread out.

## Developer estimation performance

How accurate were individual developer estimates?

Adding the anonymized ID of the developer to the estimation model produced the fitted equation:

$$Actual=0.6*Estimate^{1.11}e^{0.18D1}e^{0.35D2}e^{4.6D3}$$

where: $D1$, $D2$, and $D3$ have the value $1$ when that developer works on the Story, and $0$ otherwise.

When developer D1, D2 and D3 work on a Story, the multiplication factors are $1.2$, $1.4$, and $1.6$ respectively.

When the model is fitted to just the programming related Stories, the multiplication factors are $1.3$, $1.4$, and $1.7$ respectively.

### Does estimation accuracy improve with practice?

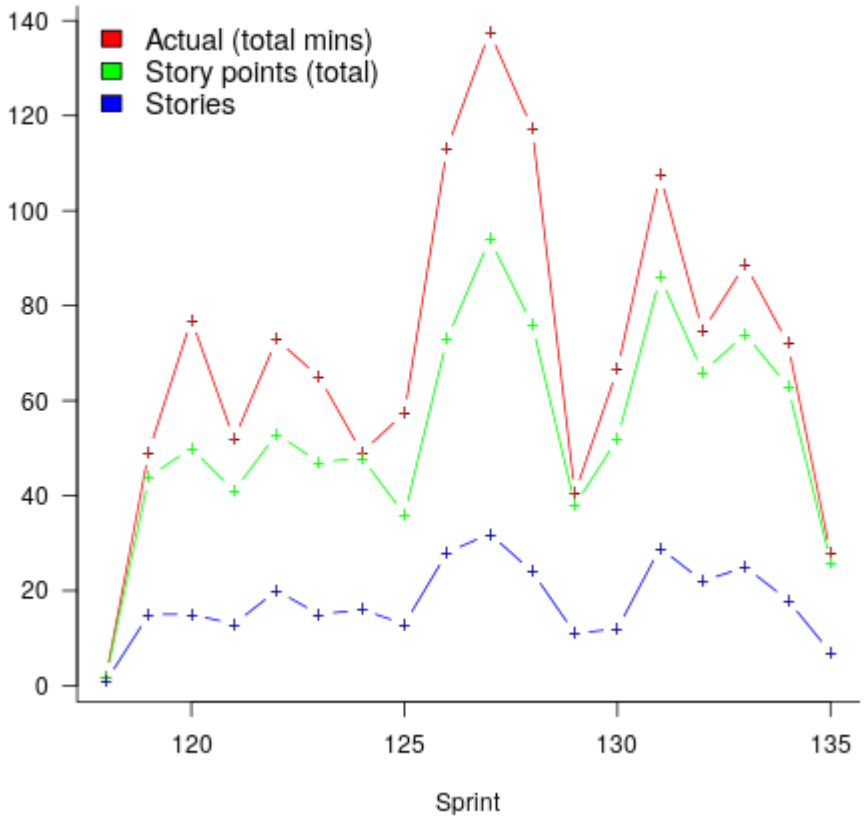
Did developer estimates improve with practice, i.e., as the project progressed?

The order in which Stories were implemented was inferred using code review dates. For each developer, the Stories they worked on were assigned a work-order, starting at 1 (for each developer) and incremented for each Story. Story work-orders are likely to be different for each developer.

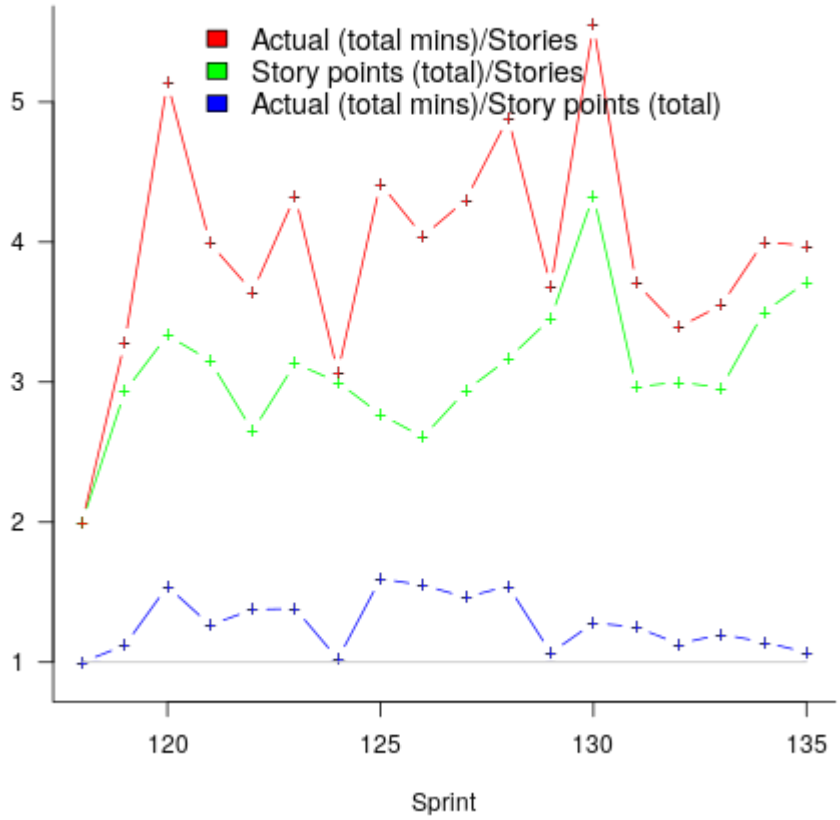
None of the regression models fitted for each developer, that included work-order information, showed a learning effect.

# Sprints

The following plot shows aggregated Story information per sprint:



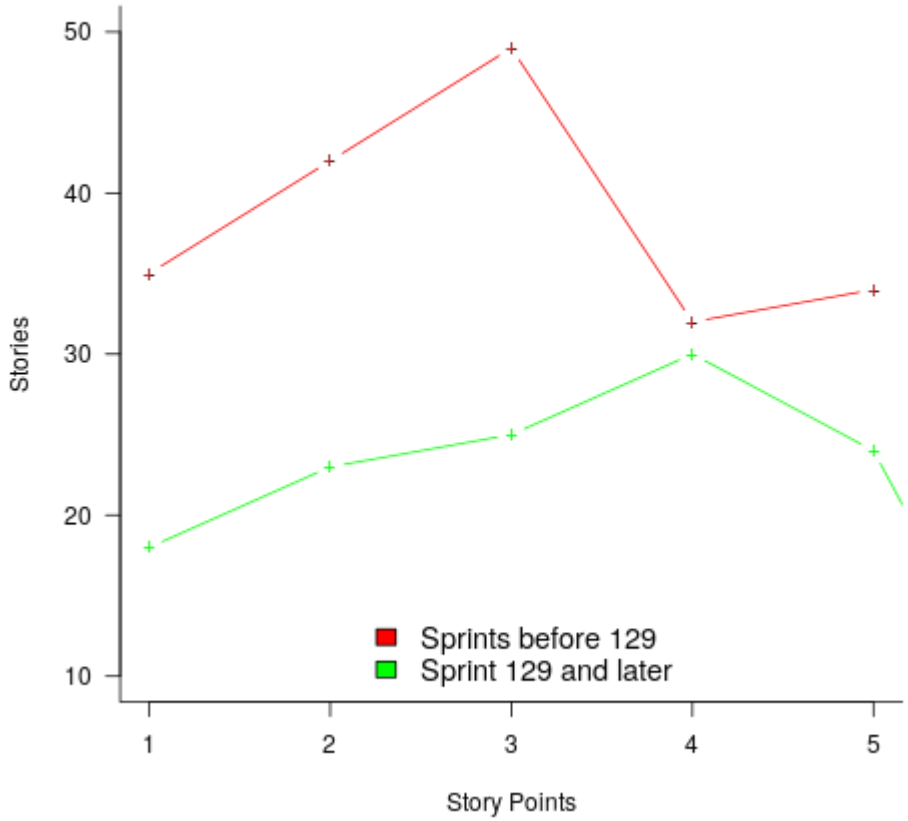

The following plot shows sprint summary information in ratio form:




Why are estimates more accurate and more consistent, starting at sprint 129?

Did these later sprints mostly contain Stories estimated at a few story points (which tend to be estimated more accurately)?

The following plot shows the number of Stories estimated to take a given number or story points, for sprints before 129, and sprint 129 and later:



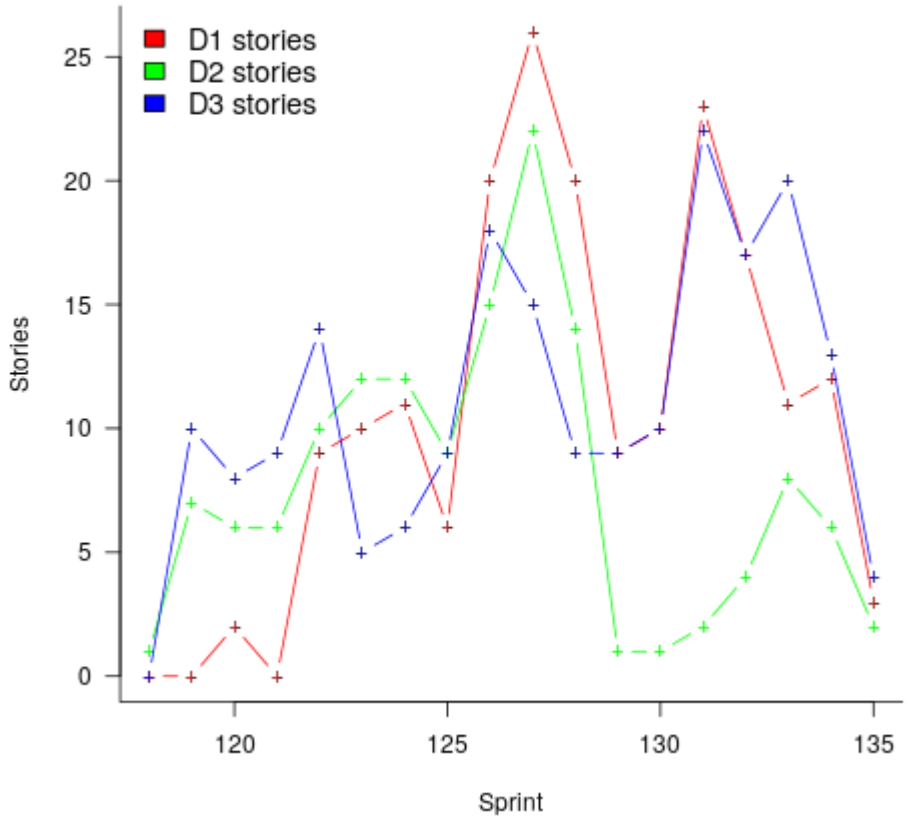

Overall, there are more Stories estimated before sprint 129. The relative number of Stories estimated to require a given number of story points is very similar up to three story points, however, later sprints include relatively more Stories estimated to require many story points.

The data shows that later sprints actually tended to contain longer Stories, i.e., not shorter, as hypothesized.

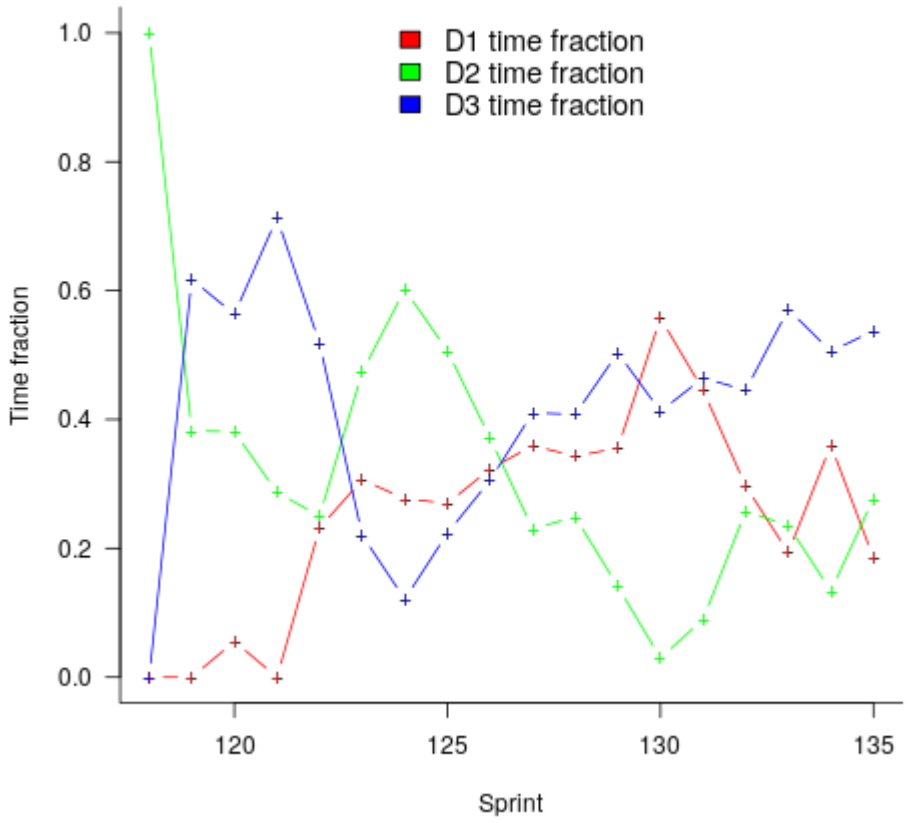
Have developers gotten better at estimating?

Fitting regression models to the before/since '129' sprints finds that developers estimation accuracy has not improved.

The following plot shows the number of Stories each developer worked on during each sprint:



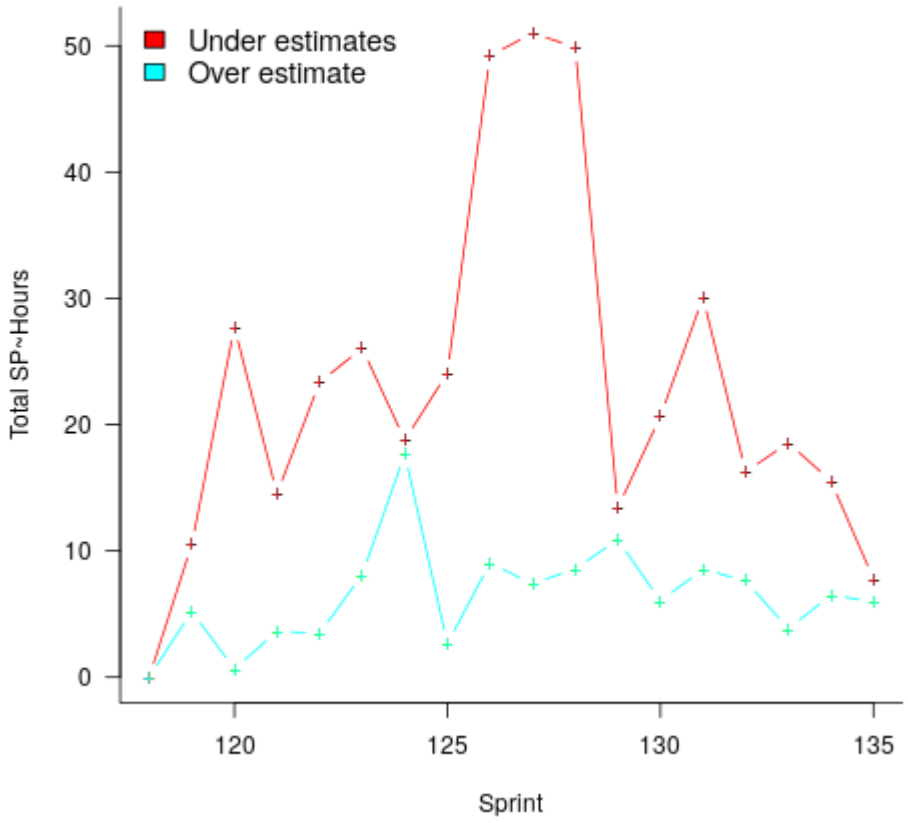

The following plot shows the fraction of total sprint time each developer spent working on Stories:



Have developers gotten better at balancing over/under estimated stories, when deciding which Story to include in a sprint (i.e., accuracy remains unchanged, but over/under amounts tend to cancel out)?

Perhaps estimation accuracy has not improved, but the amount of underestimation is being more equally matched by the amount of overestimation.

The following plot shows the total amount of under/over estimation in each sprint




Starting with sprint 126, there is a consistently greater total overestimation. The initial accuracy impact of this change is hidden by a spike in the amount of underestimation; once the level of underestimation returns to its previous levels, overall accuracy appears to improve.

The following table shows the number of sprints that each Story is assigned to:

| sprints | freq |
|---|---|
| 1 | 316 |
| 2 | 4 |
| 3 | 1 |

# Code reviews

Actual time will be impacted if there are review comments, in three ways:

1. comments on the review are first made directly on the pull request diff and/or pull request comments (it takes time for the reviewer to write them),
2. then the comments are sometimes summarized by the reviewer in the code review form we submit, which usually takes a minute or two. Summarizing comments is mostly done on failed code reviews. Most reviews that pass have no comments and the whole code review form takes < 1 minute.
3. if the code review passed, the author just has to read the comments. But if the code review failed, the author has to read them, make changes, retest, and resubmit for review, which means the reviewer has to go through all their steps again.

What are the most common durations for a code review?

The table below shows the ten most common code review durations.

```
mins  freq
 15   41
 10   22
 30   21
 20   20
  5   12
 25   10
  6    9
 14    9
 12    8
 35    8
```

We write in the review time in the form we fill out. That's different from the per minute logging we do with our time sheets. So there's some rounding happening. I, myself, try to log reviews to the minute but occasionally find myself writing in round numbers (to the nearest 5) if I lose track.

How many times is the implementation of a Story code reviewed?

The following table shows the number of code reviews associated with each branch:

```
  0   1   2   3   4   5   7   8
159 124  21   6   4   3   2   2
```

The following table shows the number of Stories whose code reviews involved a given number of passes and fails:

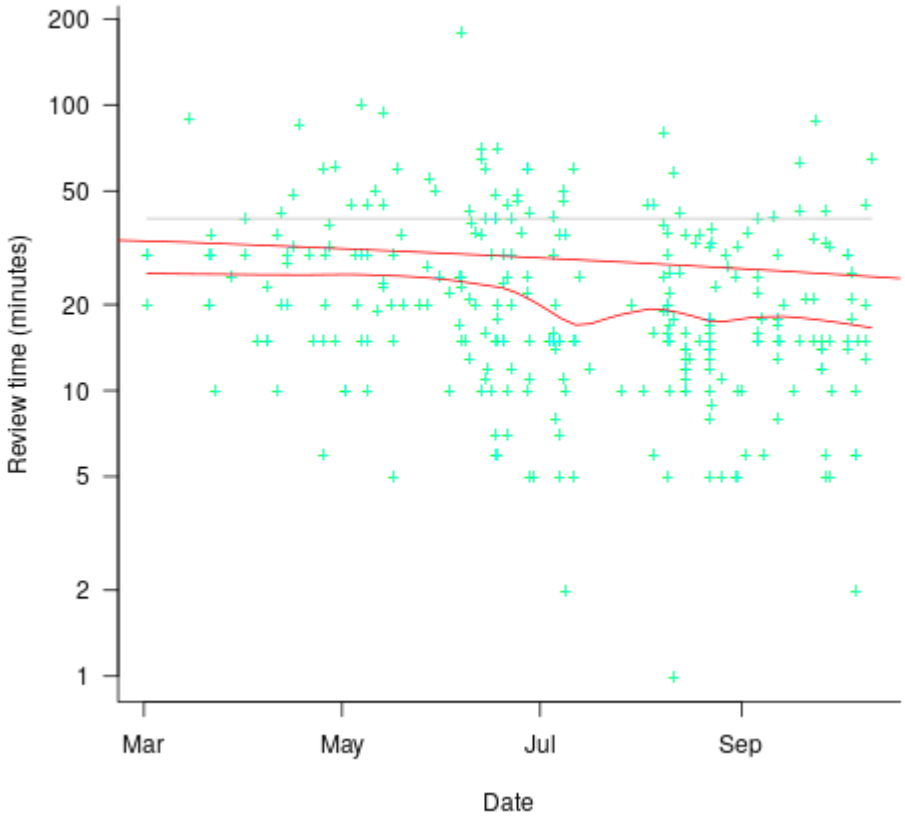
```
                PASS
F       0   1   2   3   4   5   7   8
A   0 159 120  16   3   2   2   2   2
I   1   4   5   1   2   1   0   0   0
L   2   0   2   0   0   0   0   0   0
```

The number of failed reviews was not large enough for a regression model to be fitted to an acceptable level of significance..

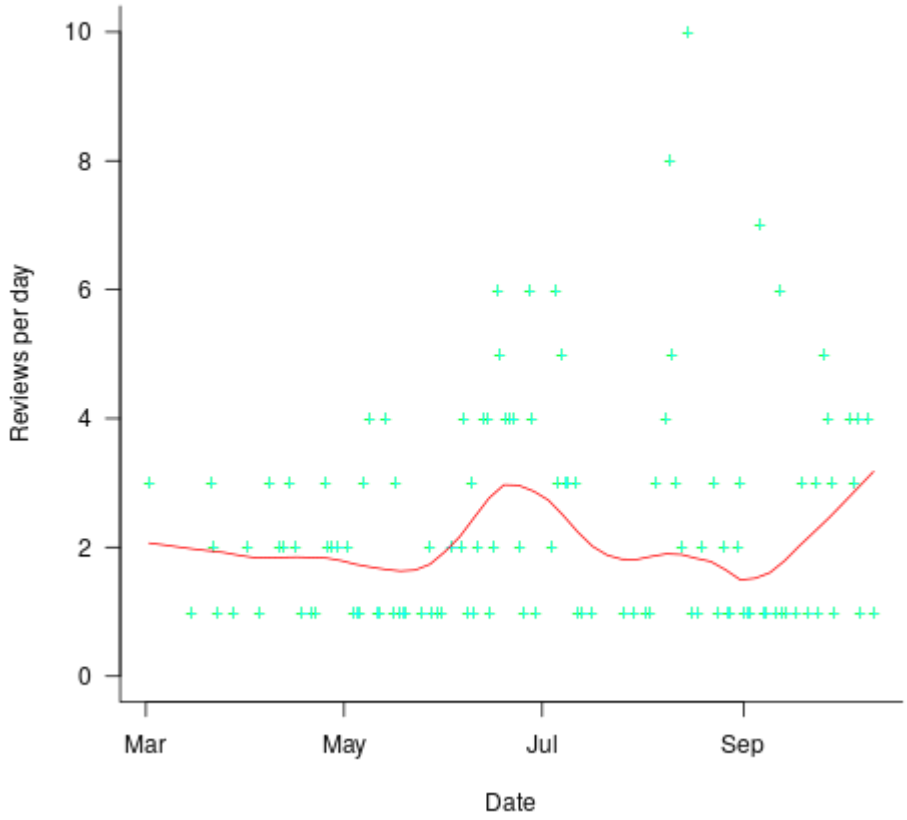
A single story can have multiple pull requests and subsequent reviews. And critical code can also have multiple reviewers. So the story with 8 reviews could have actually all been successful and it was just a story with a bunch of small pull requests.

How has the time spent on code review changed over time?

The following plot shows the number of minutes spent on each review of a Story, grey line is shows the 40 minute review time (the target maximum) , the lower red line is a loess regression fit, and the upper red line is a loess regression fit to the sum over all reviews of the same Story:

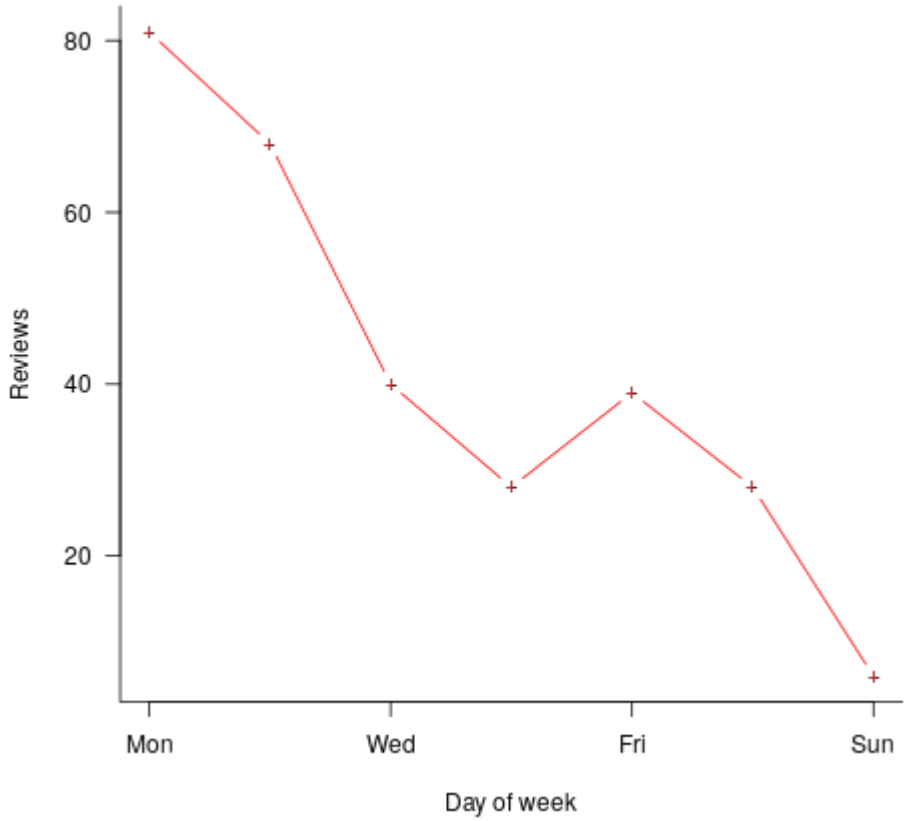



The following plot shows the number of reviews per day, the red line is a loess regression fit.



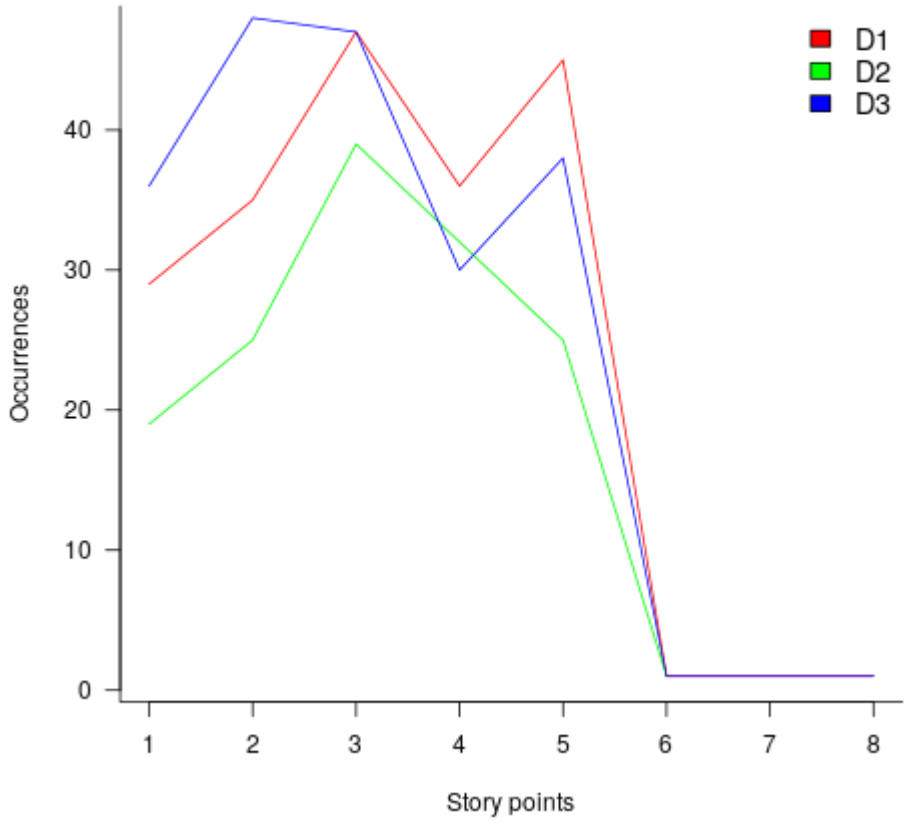

Mr A.: Yes, interesting. You can see a dip in May when one of the devs was on vacation for several weeks. Then there is a big peak in June-July. August was almost dominated by off boarding, hiring, and other non-MCO work. And then there was a big push to finish the project by the end of September.

The following plot shows the number of reviews on each day of the week:




# Story allocation

The following plot shows the number of Stories, estimated at a given number of story points, worked on by each developer.




The following table shows the number of Stories worked on by each developer and the mean number of story points of the Stories worked on.

```
           D1     D2     D3
Stories   194    141    201
Mean SP    3.2    3.2    3.0
```

The difference in the mean number of story points is not statistically significant. Differences in the amount of under-estimation by developers also skews the mean values.

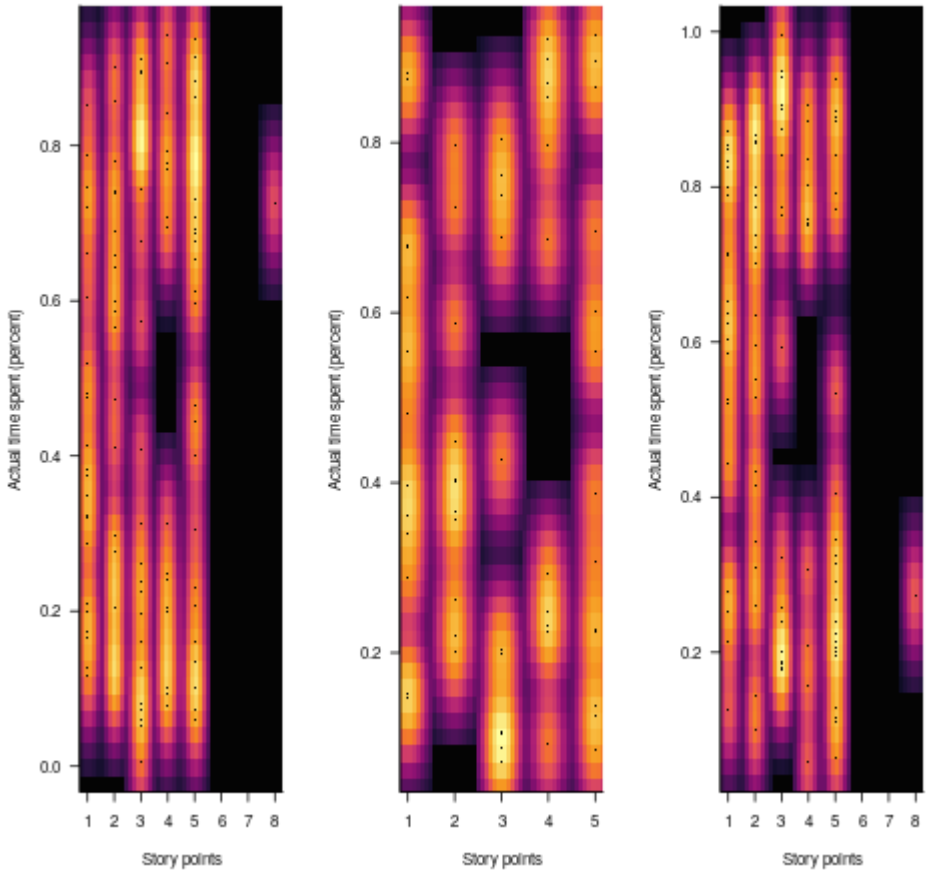
The following table shows the numbers of times developers worked alone, or with other developers, on a Story.

```
    D1   D12   D123   D13    D2   D23    D3
    29    48     31    90    47    16    64
```

What fraction of a Story does each developer implement (based on time spent)?

The following density plots show the fraction of time each developer spent on Stories estimated at a given number of story points, for all Stories they each worked on. Lighter colors indicate more Stories.

The following density plots are for Stories that had two developers working on them:




# Background on the data

## Estimating

For us, if we estimate a task as 3 SP, we are saying that we think it will take a total of three hours to move the story from TODO to DONE. That includes anything the author needs to do to prep, complete, and test the story as well as time for someone else to review the story (or for two people to review the story in cases where that was the plan) and also to merge the branch, log the code review(s), merge the PR, and deploy to production.

We have a maximum of 5 SP for any story. Some guys broke those rules from time to time.

Our choice of story points ranges has nothing to do with trying to match up to half days or anything on the calendar. We are just trying to break our work into small pieces and have them flow across the sprint board nice and smoothly as per kanban/lean principles. The tendency is for stories that are estimated at 10 to become bottlenecks that slow down our flow.