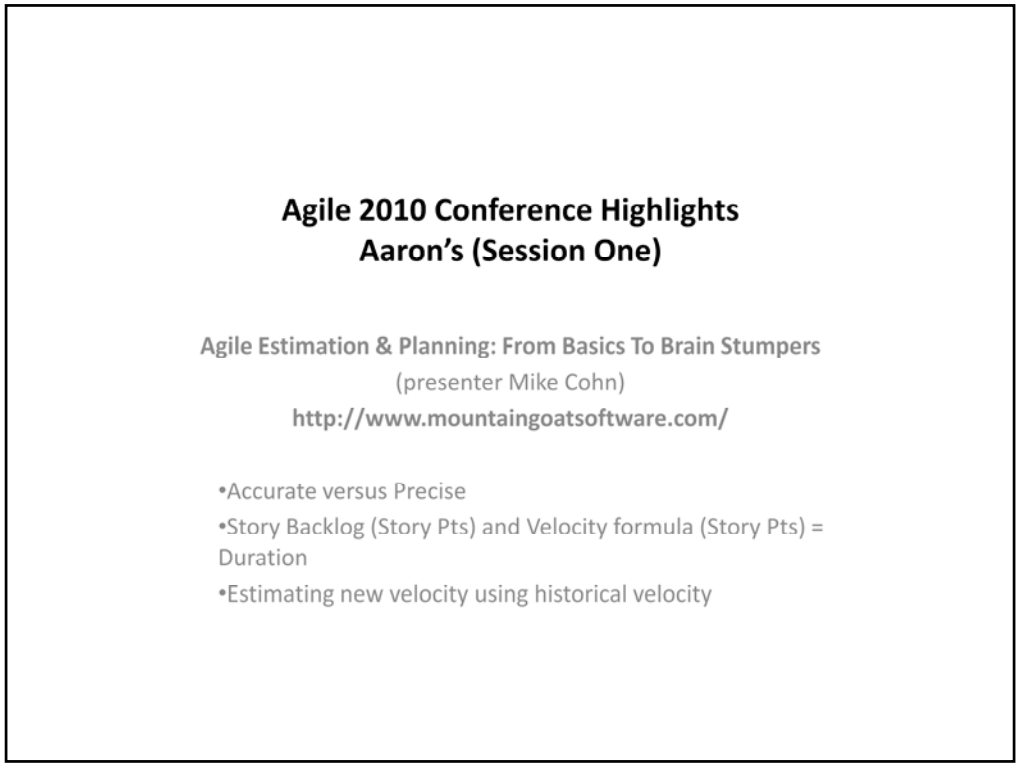
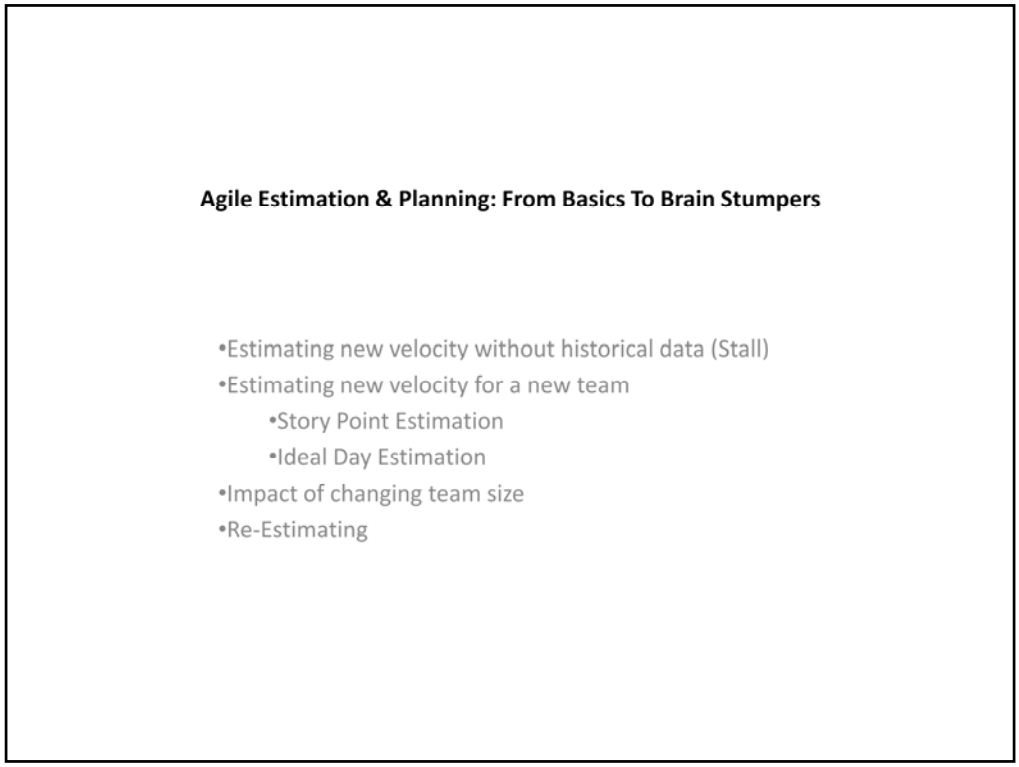
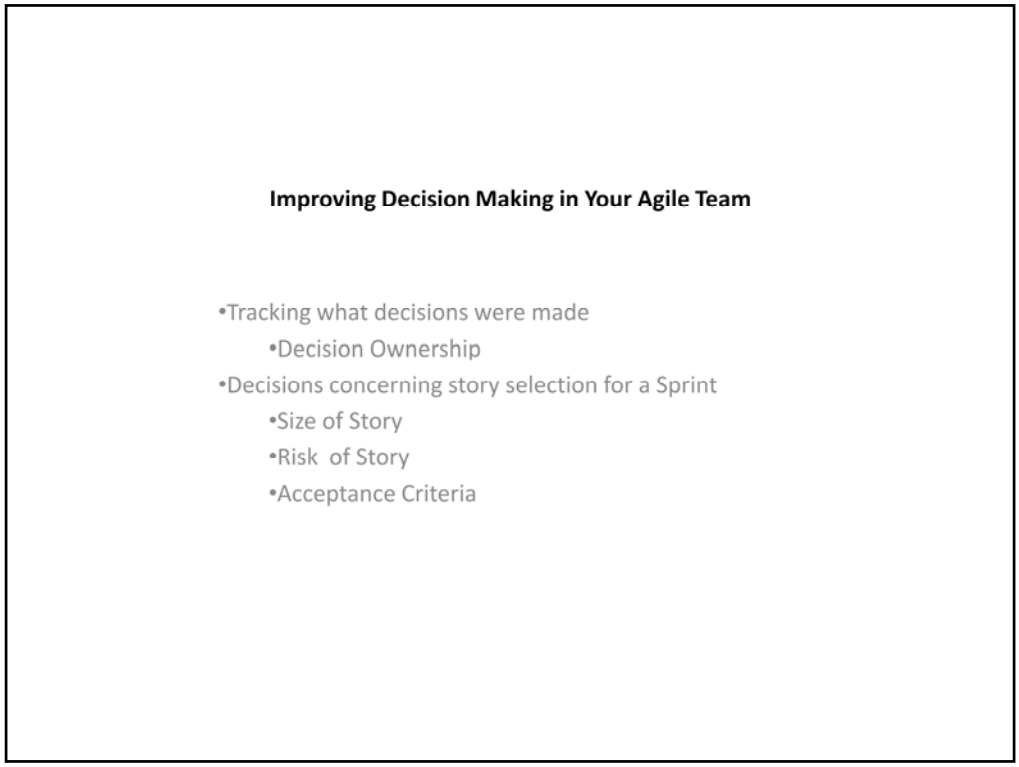


Accurate = before the end of the year
Precise = 14th of November
Velocity is calculated in Story Pts per Sprint
Estimate Using Historical Velocity is only permissible if the team in which the historical data was recorded against stays together and is taking on the next collection of story backlog.
Using the velocity for one team on another team is ok for a Low number but the high range number is risky.
Estimation given using ranges in order to increase accuracy
Take historical data through out high and low values then find mean and use it for low range number and then find high for the high range number.



Stall is a valid tactic for estimation, if you can start and complete a couple sprints before having to give an estimate you can gain some (even if not a lot) historical data
Story point estimation should be done with the whole agile team, if the team that will do the work is not available or unknown assemble a group that is similar to what the team would look like (1QA, 1BCP, 4DEVs)
Ideal Day Estimation to be done with the devs in a story to task breakdown manner
Use ideal day estimates and against the story point sizes to get en estimate story point velocity.
The team/devs doing the ideal day estimation should not be aware on any previously set Story Point estimation.
Track team size change along with your other historical data so that future team size chains can be estimated for based on the experiences of the past.
If you decided to re-estimate, the all related stories should be re-estimated so that historical velocities can be adjusted along with newly predicted velocities.

Sprint Planning: story ordering, story acceptance, story estimiation, expected velocity
Sprint Execution: who's pairing with whom, focus on closing, focus on new work
Sprint Review Meeting: did you meet your goals, demo, estimate review
Retrospective: what happed?, what didn't work?, what worked well?, what can we change to make better?



Decision Ownership: team made it, management gave it, deciding which stories to do (JIT, planning out)
Size of story: can it be completed in the sprint timeframe or does it need to be broken down,
Risk of Story: does is create more risk in the sprint or does it share the same risk we should put the two stories with a similar into two different sprints
Acceptance Criteria: does the story have adequate acceptance criteria

End Gate Taxing

Case: before checking in code you must do the following (all tests, all document, approval this and that, all tests not just those area effected)

Effect: small commit same cost as large commit more large multiday comment, simple refactoring never done because the cost is too high, no even comment being added.

Don't Break the Build

Case: breaking the build lead to the shaming of the developer with a hat or toilet seat necklace, etc.

Effect: developers made private build servers that would secretly run the build and test code without having to commit the code.

"Maybe it should be, try not to break the build but when you do take responsibility to fix it."

Reward Cleverness

Case: rewarding the individual by others including management for coming up with clever ideas.

Effect: everyone wanted to be clever so overly complicated solutions would be implement in order to show cleverness when simple solutions would have sufficed. Also the reuse of code began to go to zero.

"Once again individual rewards, created individual efforts and less importance of the team."

Strict Method Size

Case: Method can only be X number of lines.

Effect: if this is an imposed rule from outside the team itself then developers may start writing simple functional code in some nonsensical method naming breakdown.

Test Coverage Metrics

Case: using a code profiling tool test for percentage of code test coverage

Effect: writing any old code with test something, you don't even need asserts to bring test code coverage up.

"The group needs to believe that this metric is important and test code that has quality tests and is made with quality code"

Don't Have to Test Generated Code

Case: code that is in the source generated packages doesn't have to be tested

Effect: if I can write my packages into the right area I don't have to worry about writing tests for it.

"code that is not truly generated would not be in the generated source directory, the source code generator itself should be tested"

Reward Team Member Percent Tasks Complete

Case: reward developers with bonus for completing a large percentage of their tasks then others

Effect: developers don't like to work together because they are in it for themselves, developers all fight over the easy of less risky tasks

Reward High Bug Fix Rate

Case: developers that complete the fixes of more bugs get rewards

Effect: doesn't encourage bug creation prevention, developers fight over the easy bugs to fix and the easy bugs get fixed first.

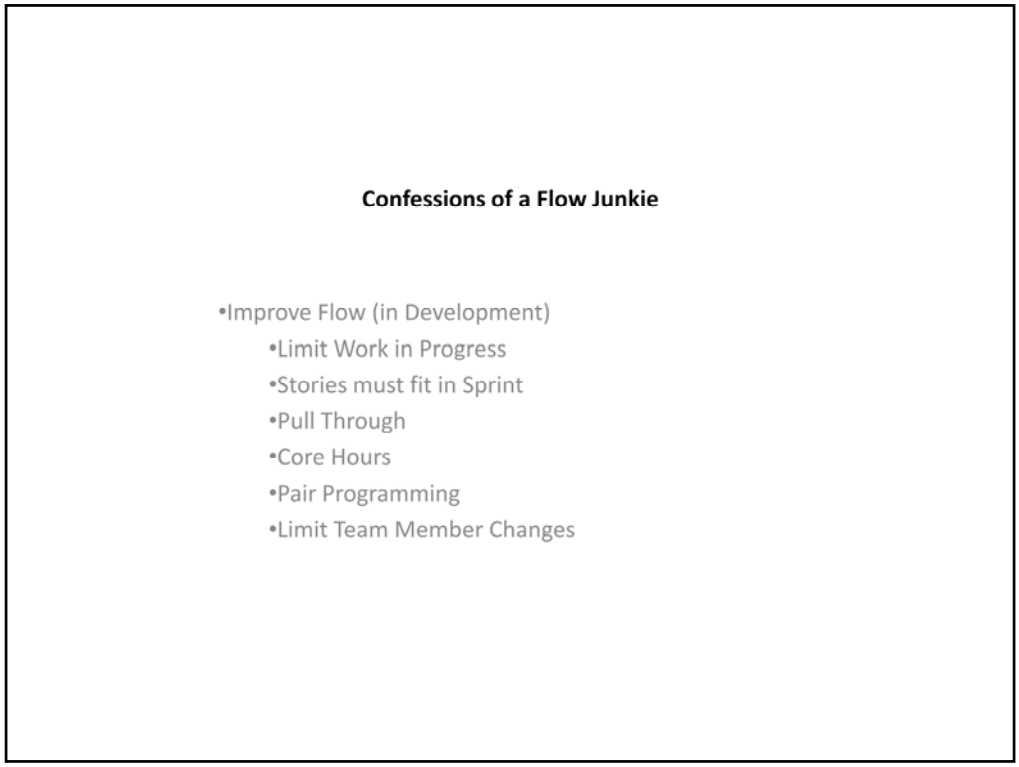
Conclusion:

Be aware when rules get imposed that are not agreed upon by the team.

If you can not agree as a team that there is value in the rule then remove the rule.

Incentives for the individual always ends with teams(groups of developers) without any unity.

What is Flow?
What does it feel like to be in the flow? (don't have to stop for lunch, don't want to call it a day just yet)
Personal Flow versus Pair Flow: an individual starting and completing before the next one begins may increase personal flow but if you can cooperate toghet the whole job will get done faster. (flipping coins example).
Improving flow : traffic roundabouts, public transit (underground), Toyota assembly line, etc.



Limit Work in Progress: Development complete not just Code Complete
Stories must fit in Sprint: Or break them down
Pull Through: a lean concept of not clogging the pipeline, go down stream to help pull things along (help out QA, help get work deployed to test environment)
Core Hours: hours in the day that you are no to be interrupted except for story related questions

- no personal email
- no non-story related email
- no web surfing beside story/technology related research

Pair Programming: if what you are doing it not acceptable to ask your pair to join you doing then you don't do it. Do you pair program on personal email…probably not.
Limit Team Member Changes: if team members are pulled on and off your team this disrupts flow. So fight to keep your team members and access carefully the adding of new team members (adding a team member for just two week may just be negative velocity)

Ribbons: scrum master, new to agile, coach, consultant,
Most vendors at the conference were of the electronic whiteboard with cards
Tons of masters and coaches, I heard a manager asked how to handle a problem with a team not being hyperproductive, "get a coach for them".
Interestingly "scrum master" certification only means that you have had the training and has nothing to do with mastering.
So many certifications so little time, so what not save yourself the time and effort and get ASS certified.