Working with Team

Petri Heiramo

Agile Coach, CST

Exercise: What to Do?

You are the ScrumMaster and are heading for the team room. The functional analyst runs past you crying and the lead engineer runs past you enraged, both on the way to their functional managers' offices.

You go into the team room. You can cut the tension with a knife it is so thick.

Apparently, the analyst has been writing specs and giving them to the engineers, who then change them as they see fit. Anger over this has been building for three weeks.

What do you do?

Exercise: Team Norming

A team is in its Sprint planning meeting. Half of the team is from a technical company that is running the project. The other half consists of contractors.

A usability engineer from the technical company demands to know how the contractors will ensure adequate user interface. A system architect from the technical company demands to know how the contractors will follow and not ruin the systems architecture they define.

What is wrong here? What do you do?

Exercise: The Olde Expert

Before becoming the ScrumMaster, you were a technical lead and respected for your design and programming skills.

In your new project, the team comes to you for advice on a challenging architectural choice. The team members have been arguing between two approaches, but could not agree which one to choose.

They want you to choose the approach.

What do you do?

Exercise: Limits of Self-Management?

You are the ScrumMaster. Everyone on the team except John meets with you. They tell you that John is not doing his work, is offensive, is difficult to work with, and they want you to fix the problem.

What do you do?

Exercise: Joint Responsibility

You are the ScrumMaster at the first Daily Scrum. There are two programmers, a tech writer, and two quality assurance people.

The programmers report that they were in a design meeting and will continue today. The tech writer says that they are working on the table of contents. The quality assurance people report that they are setting up the test bed.

You ask the tech writer and QA people why they aren't in the design meeting. They say they weren't invited. You ask the programmers why they weren't invited. They ask you what possible benefit these people would add to design?

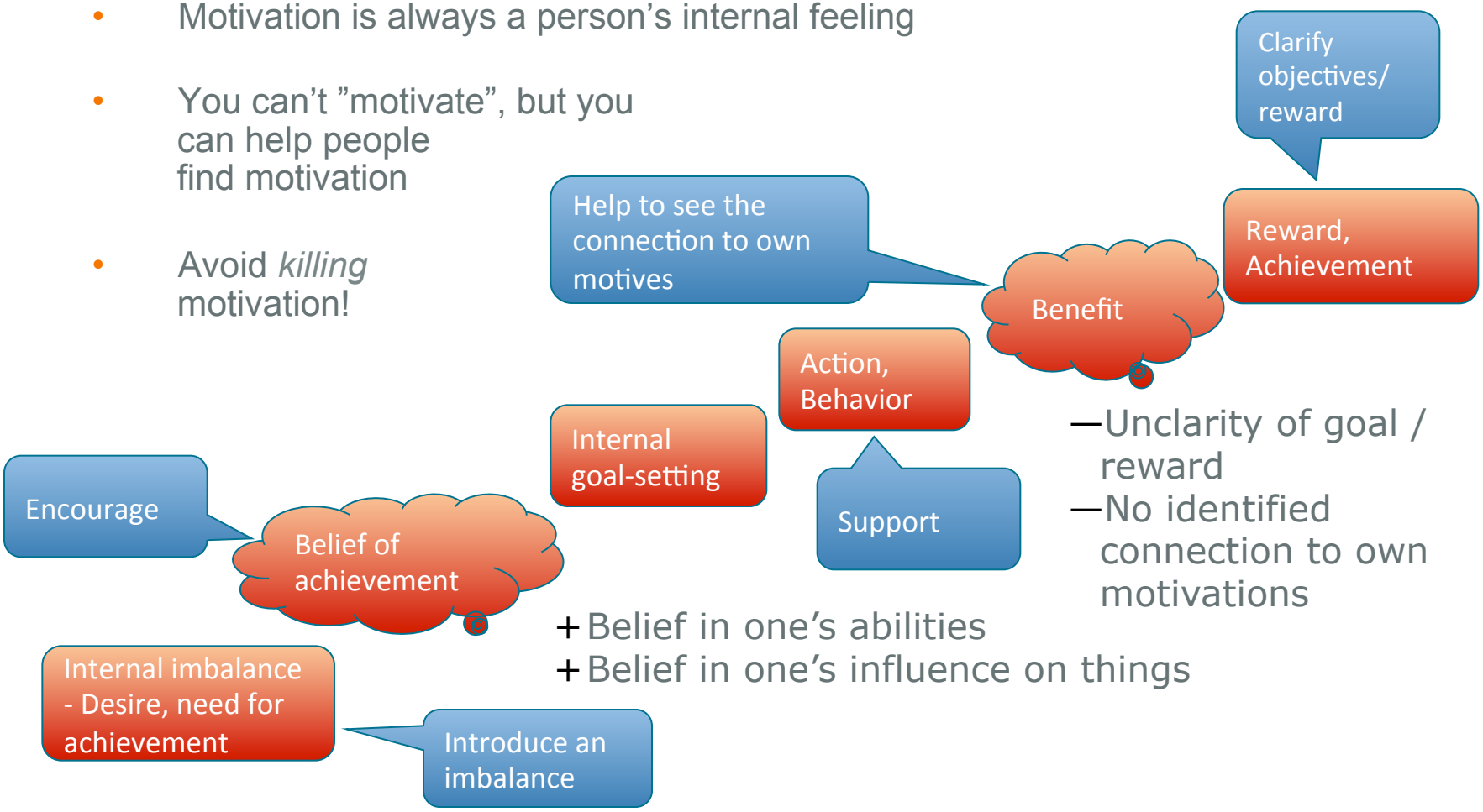
What do you do?

Basic Truths about Team Motivation

- People can be most productive when they manage themselves
  - But it requires the right approach
- People take their commitment more seriously than other people's commitment for them
- People have many creative moments during downtime
- Under pressure to "work harder," developers automatically and increasingly reduce quality

Can We Affect Motivation?

- Motivation is always a person's internal feeling
- You can't "motivate", but you can help people find motivation
- Avoid *killing* motivation!



—Unclarity of goal / reward
—No identified connection to own motivations

+ Belief in one's abilities
+ Belief in one's influence on things

Social Norms vs. Work Norms

- Experiment report:
  - People in testGroup1 were offered $5.00 to move boxes for 5 minutes. They moved 159 boxes on average.
  - People in testGroup2 offered $0.50; they moved 101 boxes avg.
  - People in testGroup3 offered no money, but asked to do it "merely" as a social request between colleagues. They moved 168 boxes.
- The point is not that money is not relevant, but that the power of social norms (e.g., in a teamwork context) on motivation or behavior is non-trivial -- more than some may have anticipated.
- What happens if you thereafter offer the social-norm people in testGroup3 money? They flip the context to work norms, and significantly lower output. The bloom is off the rose -- and for a long time. If you then again ask them to do the task as a social request, the output remains poor -- it's been stickily reframed as a work norm context.
- Example: what happens if you offer $ to kids to read books, after it was previously a social norm without reward?
- Or: 1€ penalty for being late from Daily Scrum vs. be there for others' sake?

When You Don't Know What to Do

- Ask the team!
- Examples
  - I noticed <situation> what shall we do?
  - I observed <situation> is that important?
  - I feel <feeling> do you share that?
  - Shall we try to find out why <situation>?
  - What do you think we should do?
  - Who has any idea about …?
  - Is this useful?
  - What have you decided?
  - What should I do?

When You Do Know What to Do

- If it involves the team, ask the team
  - If you have an idea what a solution could be, you might know to ask the right questions
  - Still, don't assume your answer is better than the team's answer – don't force it, accept the team's answer
    - They are responsible for the work, not you
    - They are committed to their own decisions, not to yours
    - The team must be allowed to make mistakes, otherwise there is no empowerment – you will then help them learn from them
  - Are you **sure** you're right? ☺
- If it involves the PO, ask the PO

"The Hardest Part of Being an Agile Project Manager"

"I've been leading a team of 5 devs, 1 tester, 2 designers, and 1 customer for the past month or so. During that time, I've learned a lot from watching this team grow, in their relationships to each other, and in their ability to work together as a team.

What has struck me about this is that all of this happens best if I keep my nose out of it. And this is the hardest part for me.

...

It is not my job to be a problem solver as many PMs feel is their role. It is my job to enable my team to solve problems on their own, and to create an environment where they feel safe to do so. This is much harder than solving the problems themselves, as I (as do most people) have the urge to grab those reins and lead away!"

– Brian Button
(http://www.agileprogrammer.com/oneagilecoder/archive/2007/02/11/22180.aspx)

Team vs. Group

A **team** comprises a group of people or animals linked in a common purpose. Teams are especially appropriate for conducting tasks that are high in complexity and have many interdependent subtasks.

A **group** in itself does not necessarily constitute a team. Teams normally have members with complementary skills and generate synergy through a coordinated effort which allows each member to maximize his or her strengths and minimize his or her weaknesses.

- Wikipedia on ”Team”
(Highlights are mine)

Team Formation

- **Forming – Storming – Norming – Performing**
  - The four stages of team formation, as suggested by Bruce Tuckman, 1965
- **Forming** – Coming together, accepting the goals. Independent behaviour
- **Storming** – Different ideas compete, potential conflict
- **Norming** – Team members adjust their behaviour to each other, common norms for working agreed
- **Performing** – Team functions as one unit, heterogenity is a source of creativity and strength, hyperproductivity
- Progress through these phases requires coaching and support (ScrumMaster)

One goal of Scrum is to create hyperproductive teams.

The First Day with the Team

- Set aside at least one day when team first gets together to form:
  - Introductions and backgrounds
  - Team name
  - Team room and Daily Scrum time/place
  - Development process for making Product Backlog done
  - Definition of “Done” for Product and Sprint Backlog items
  - Rules of development
  - Rules of etiquette
  - Training in conflict resolution
- In the case of distributed teams, get people to one place

Colocation – Team Rooms

- Colocation and team rooms are a powerful way to support collaboration and communication
  - Some studies have suggested 100% performance improvement from collocation alone
  - Some other studies have shown, that *with right arrangements* you can have distributed teams with equal performance
- Eliminate communication barriers
  - Separate rooms, separating walls
- People should be able to see everyone and the taskboard
- Enough wallspace for whiteboards, designs, information radiators, etc.

Distributed Team Recommendations

- Co-locate team as often as possible, especially at inception and key milestones. Rotate members around.
- Invest in (and plan for) tools that provide a shared environment. Plan to experiment.
- Establish a single global instance of project assets, easily accessible by all.
- Try virtual team building (team wiki w/bios & photos).
- Establish known hours, with as much overlap as possible.
- Apply high cohesion and low coupling to allocation of work to sites.
- Develop a shared team vocabulary.
- Don't let anyone go dark.
- Apply Scrum-of-Scrums concept when mass remote meetings unproductive.

Requirements in Practice

Writing User Stories

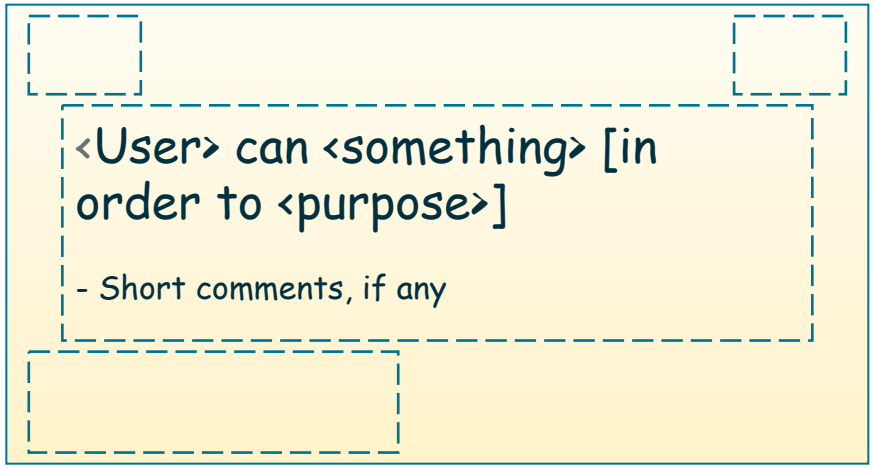
- Standard format:
  <User> can <something> [in order to <purpose>]
  - Exceptions to the rule can be made, if the clarity of the story requires it
    - Only one user at a time must be able use one floating licence
- A good story is (INVEST)
  - Independent
  - Negotiable
  - Valuable to users or customers
  - Estimatable
  - Small
  - Testable

Exercise: MyBooks.com website

- Vision
  - MyBooks.com allows internet users to record down books they own (”bookshelf”)and share their opinions of them. MyBooks.com creates a community for discussion, reviews and feedback. The site attempts to give a rich functionality
- User classes
  - Shelf owner – Someone who has recorded at least one book in the system and is maintaining their own bookshelf
  - Visitor – Someone who is viewing other people’s bookshelves
  - Admin – Maintainer of the MyBooks.com site

Exercise: Bookstore.com Website

- As a team, **brainstorm** functionality for the website
  - Aim for **at least** 15-20 stories
  - Get wild with the ideas ☺
- Write down the stories on index cards
  - Reserve space in the corners
  - One story per card
- 15 minutes



Independent

- For example, non-independent:
  - A customer can pay with Visa
  - A customer can pay with MasterCard
  - A customer can pay with American Express
- First implemented user story might take three days, the other two one day each
  - How do you assign estimates on the stories?
- Work-arounds:
  - Combine into one larger user story
    - A customer can pay with credit card [5 days]
  - Find a different way of writing the user stories
    - A customer can pay with one type of credit card (Visa, MasterCard or American Express) [3 days]
    - A customer can pay with two other types of credit card [2 days]

Negotiable

- In order to be effective, the details of the user story must be negotiable between the customer and the team
- Stories are reminders of the conversation
  - Keep the detail at discussion level for cost effectiveness
- Too much detail gives an impression of completeness
  - There is always too much detail to write everything
  - Impression of completeness leads to people not thinking
    - Stupid mistakes just because ”it wasn’t written on the card”
- Include key details and open items as ”confirmation”, i.e. key acceptance test

Valuable to Users and Customer

- Each user story should have at least one user or stakeholder who finds it valuable to them
  - Make the interested party clear in the user story
  - Avoid stories where interested party is the development team
- If you have a user story that is very difficult to write in the standard format, consider the entire validity of the user story
  - Is it a general quality requirement and better put in Definition of Done?
- The best way to ensure valuability is to have the customer write the user stories

Estimatable

- Three common reasons why a story may not be estimatable
  - Developers lack domain knowledge
  - Developers lack technical knowledge
  - The story is too big
- Lacking domain knowledge can be compensated for by having the customer or users explain the story
  - Not necessary to go through all details
- Technical knowledge can be generated through ”spikes” [XP term]
  - Brief experiment on the technology in the domain
  - Split to two stories: the spike and the real work
- Too large stories can be split to smaller stories
  - If the team doesn’t know how, do a spike

Small

- Too large stories are difficult to define and understand in sufficient detail
  - Differing priorities within the story
  - Does not fit into sprints
  - Difficult to estimate
  - Too much to remember from a single description
- Large stories are split to smaller ones until they feel comfortable for the team
- A user story can also be too small
  - Threat of micromanagement and planning overhead
  - Combine into aggregate stories

Testable

- Stories that cannot be tested cannot be proven to be successfully developed (or regression tested)
  - The application will be easy to use
  - The application responds quickly to user actions
- Untestable stories are typically linked to non-functional requirements
- Aim for automation
  - Strive for 99%, not 10%
  - Some stories are not automatable, but can be tested
    - 95% of novice users can send a message in less than 5 minutes
  - Automation provides clear yes/no answer and are powerful in regression testing
- Improving testability
  - Define concrete values and activities
  - Never say ”never” – impossible to prove
    - Instead, it is possible to show that something is rare

Splitting Stories

- Large stories (often epics) typically fall into two categories
  - Compound stories
  - Complex stories
- Compound epics consist of multiple smaller stories
  - Usually easy to split up
- Complex epics are harder to split
  - Often one very large feature or goal, e.g.
    - Complex algorithms, porting, etc.
  - Often research (spikes) are needed to find working substory structure
    - The inability to split often indicates insufficient understanding of the features
  - May require stubbing parts of the system

Split Stories Along...

- Operational boundaries
  - Subfeatures, operations (CRUD, ...), exceptional cases, error conditions, ...
- Data sets
  - Text / music / images / video, different types of invoices, message types, different statuses, ...
- Technology
  - Platform versions, hardware versions, protocols, ...

Combining Stories

- Sometimes the stories go too small for meaningful release planning
  - Very small user stories
  - Errors
- Combine several stories to one story for estimation purposes
  - Give one estimate of size
  - Allocate to one sprint for implementation

Exercise: Refactoring PBL

- As a group, go through your identified user stories for MyBooks.com and evaluate them in the light of INVEST
- Refactor the stories you feel could use an improvement
- Split and combine stories, if you feel it's appropriate. Split up stories that are obviously too large for estimation and planning

Thank you

For more info:

petri.heiramo@gmail.com