Team Management & SM Tools

Software Engineering II
WS 2020/21

Enterprise Platform and Integration Concepts

Image by Paul Albertella from flickr: https://www.flickr.com/photos/paulspace/10427636463/ (CC BY 2.0)
Tools in Software Engineering

**Noun**

**tool** (plural **tools**)

1. Equipment used in a profession, e.g., **tools** of the trade.
2. Something to perform an operation; an instrument; a **means**.
3. *(computing)* A piece of **software** used to develop software or hardware, or to perform low-level operations.

- **git is a tool**, so is your IDE: Means to manage source code
- **Brainstorming is a tool**: Means to generate varied ideas

▶ Processes and procedures are tools in a wider sense
  - We don't edit images with IDEs, **choose your weapons!**
  - Don't worry about labels, use what's needed in the context

Source: https://en.wiktionary.org/wiki/tool
Scrum Master Tools

SM: maximize the value delivered by the Scrum Team

- Tools and concepts focused on helping team members improve & practice Scrum methods and values
- Needs support to
  - Facilitate required Scrum events & workshops
  - Prepare Retrospective, gathering observations during the Sprint
  - Help the team to identify and remove impediments
  - Support PO and Dev. Team in making Scrum Artifacts transparent

Team communication patterns are valuable!
- Especially relevant in (half-)distributed teams
- Is everyone participating and is everyone's voice heard?

Scrum Master: Keep an eye on the team & process

■ Take the meta-view during meetings
■ What is currently happening in this team?

■ Time
  □ Time-box activities & stick to them
  □ Meeting agenda with breaks

■ People
  □ What team dynamics am I noticing?

■ You
  □ Facilitator vs. team member
  □ Focus on others, Let them talk
Scrum Master Tools

The (Scrum Master's) Diary
- SM main focus: team meetings
  - Active attendance
  - Organize and pay attention to the details
- Diary: help structure thinking & identify common anti-patterns
- No need to share the diary if you don’t want to

Source: https://medium.com/idealo-tech-blog/the-scrum-masters-diary-a-powerful-tool-to-derive-effective-interventions-ca5e5655ad49
The (Scrum Master's) Diary
Four questions for actively attending meetings:

- What did you observe? (describe what actually happened)
  - "Rick is confused"  "Rick asks about the next steps"
- How do you interpret and evaluate the observation?
  - Is this a normal process? How does it make you feel?
- What are questions when evaluating the observation?
  - “Does everybody still remember the last retrospective outcomes?”
- What changes (for yourself?) & interventions do you plan?
  - Interventions applied to systems not "onto" people
Scrum Checklists

- **Assess** the your team's **process state**
- **Compare** to Scrum Guide/experts/peers ideas, e.g.
  - Henrik Kniberg's "unofficial Scrum Checklist" ([https://www.crisp.se/gratis-material-och-guider/scrum-checklist](https://www.crisp.se/gratis-material-och-guider/scrum-checklist))

**Discuss!** Does the team agree with the dimensions?
Custom team maintenance checklist

■ Team members are experts for their own work processes
■ Build a **team-specific checklist**, refer back to it regularly
  □ Aspects that are **most important to the team**
  □ Minimal consensus

■ Idea: 3 categories
  □ “Is the **Product Owner** doing well?”
  □ “Is the **Development Team** doing well?”
  □ “How are our **engineering practices** doing?”
Retrospectives are easy

- **Just a fancy word for a meeting** with the goal of improving the future
- We can reflect on the past, because we were there

VS.

Retrospectives are hard

- Only effective if they lead to change
- **Change is hard**

Source: [https://www.retrium.com/blog/10-things-all-agile-teams-need-to-know-to-run-effective-retrospectives](https://www.retrium.com/blog/10-things-all-agile-teams-need-to-know-to-run-effective-retrospectives)
Retrospective Prime Directive
Regardless of what we discover, we understand and truly believe that everyone did the best job they could, given what they knew at the time, their skills and abilities, the resources available, and the situation at hand.

■ Base assumption and **foundation of successful Retrospectives**
■ Reminder to not focus on blame
■ Start the meeting by reading it out? (priming)
■ Refer back to it when related problems arise

The psychological concept of "priming" is disputed
Common Retrospective Structure

Five consecutive phases

1. Set the stage
   - Check-in with everyone
   - Gauge team spirit

2. Gather data
   - Gather perceptions & data points on last sprint

3. Generate insights
   - Identify common topics

4. Decide what to do
   - Decide on (few) changes for next Sprint
   - Find consensus

5. Close the retrospective
   - Wrap up the Sprint in a positive manner
Scrum Master Tools
Retrospectives

**Follow up** (the 6th Retrospective phase)

- **Share results (Action Plan)** of meetings transparently
  - With team (and entire project?)
  - Add changes to issue tracker/learning backlog?
- **Check the status of changes regularly (during Sprint?)**
- **Review** at next Retrospective meeting

- No point in running a Retrospective if nothing changes
  - **Focus on few (single?) actionable changes**
  - Few things are ever perfect
Retrospective agendas & activities

- Collections of popular activities for Retrospectives by Agile practitioners
  - Usually divided into Retrospective phases
  - Retromat: [https://retromat.org](https://retromat.org) (Recommendation!)

- Different ways to do things, e.g. voting:
  - **Dot Voting**—3 votes per person, distribute
  - **Roman Vote**
    - thumbs up (support), down (I want to speak), side (whatever majority wants)
  - **“Was that a decision?”**
    - Loudly ask in unclear situations, turn silence into action

Try out "best retro for beginners": [https://retromat.org/blog/best-retrospective-for-beginners/](https://retromat.org/blog/best-retrospective-for-beginners/)
Scrum Master Tools
Retrospectives

Retrospective anti-patterns

■ Common issues to watch out for:
  □ Reporting to management, little focus on collaborative improvement
  □ Few participants instead of the whole team
  □ Everybody is happy, nothing to discuss and no improvement
  □ Blame game, i.e. focusing on complaints instead of possible improvements
  □ Lack of facilitation, agenda and structure

■ Identified your own common issues?
  □ Share!
  □ What helped?

Organizing Work & Dev Team Tools

Software Engineering II
WS 2020/21

Enterprise Platform and Integration Concepts

Image by Four Bricks Tall from flickr: https://www.flickr.com/photos/fourbrickstall/48716532168/ (CC BY-NC-ND 2.0)
Development Team Tools

Software development tools
- SCM, issue trackers, CI, static code analysis, IDEs, plugins...
  - Have a favorite (that works well with the tech stack)? Share!
- Organizationals tools & techniquest
  - No such thing as a specific "PO Tool"/"SM Tool"
  - Adapt tools to your (technical) context!

Technologies and frameworks
- Your problem has probably already been solved somewhere
- Using something new means everyone needs to learn it (costs)
Relying on project data for organization
- Which stories are part of this Sprint? What to work on next?
- Who is working on which tasks? Who needs help?
- Which version should be presented (to stakeholders)?

User stories & tasks in (GitHub's) issue tracker
- Assign issues to developers
- Milestones to assign user stories to sprints
- Issue tags, e.g. to denote responsible teams or status
- Project management tools with overviews of issues, e.g. GitHub Projects, or Zenhub
- Tag versions that can be presented

Identifiable user names really help when assigning: "Who is Gronkh12?"

$ git tag --a v0.1 --m 'version after Sprint#1 without US #2'
Project data

- Especially valuable for large projects: **keeping up to date with project progress**
- What are the most interesting recent changes?
  - In the issues/stories of the current Sprint
  - In the history of the git repository
- Use visualization tools & tags to help
  - **Call out the highlights!**

Source: https://livablesoftware.com/tools-to-visualize-the-history-of-a-git-repository/
Task Board (Scrum Board/Kanban Board)

- Team’s home base, everyone has access and modifies (artifact transparency)
- Visual representation of tasks & progress
- Swim lanes/multiple boards by topic

This is where data for a Burndown-Chart could come from
Scrum Boards – Virtual vs. Real-Life
Communicating technical details

- How do I find out about architecture changes?
- How do I know how to use other people's code?

Tools that might help:

- (GitHub) wiki to (briefly!) document components
- **Code comments** explaining the larger context, common pitfalls
- Common communication channel for announcing (breaking?) changes
- **Automated tests** that show component functionality
  - Documentation stays current
Debugging

If debugging is the process of removing software bugs, then programming must be the process of putting them in.

—Edsger W. Dijkstra (allegedly)

- Debuggers & Stack Overflow are helpful
- Taking a walk is underrated

Using others as "rubber ducks": "cardboard superman"
Git — amend, interactive staging

Change commit message of previous commit
(Careful, don't do this if you already pushed the commit)

$ git commit --amend -m "new message"

Forgot to commit files?

$ git add [missing files]
$ git commit --amend #uses the previous commit's message

Undo the amending

$ git reset --soft HEAD@{1}
$ git commit -C HEAD@{1}

Interactive staging (also allows committing only parts of files)

$ git add -i
$ git add --patch [file]

Interactive staging (git add -i) is probably the most powerful git feature you're not using yet.
Git — reflog, diff, stash

Log of all recent actions

$ git reflog

What did I work on recently?
Show differences that are not staged yet

$ git diff

Shows differences between staging and the last file version

$ git diff --staged

Temporarily store/retrieve all modified tracked files

$ git stash
$ git stash pop

List all stashed changesets

$ git stash list

stash allows checking out another branch/commit, without losing your current changes
Git — log, blame, rebase

Shorter version of the git log

$ git log --abbrev-commit --pretty=oneline

Show pretty graph of git history

$ git log --graph --decorate --pretty=oneline --abbrev-commit

Show changesets in the log

$ git log -p

Show what revision and author last modified each line

$ git blame --date=short [file]

History is becoming cluttered with merge commits

$ git rebase <branch>

Do not rebase commits that others have worked with! "people will hate you, and you’ll be scorned by friends and family."

**Git — setup**

- Created "experiment" branch to try something out
  
  ```
  $ git checkout -b "experiment"
  $ git commit -a -m "C3"
  ```

- Easiest way to integrate the branches is **merge**
  - Will create merge commits
  
  ```
  $ git checkout master
  $ git merge experiment
  ```

[Diagram of Git branch structure]

---

Tips & Tricks — Software Engineering II

## Git — execution

- **git rebase**
  - Take all the changes that were committed on one branch and replay them on another one
  - Only do this with local commits

  ```
  $ git checkout experiment
  $ git rebase master
  ```

- Afterwards: fast-forward the master branch
  - No merge commits

  ```
  $ git checkout master
  $ git merge experiment
  ```

Git cherry-pick

- **Problem**: Quickly get changes from other commits without having to merge entire branches

- `git cherry-pick`
  - apply the changes introduced by existing commits

```bash
$ git checkout master
$ git log --abbrev-commit --pretty=oneline
  d7ef34a C3: Implement feature
  0be778a C4: critical change introduced

$ git checkout experiment
$ git cherry-pick 0be778a
```

Git Self-help Resources

- How to undo (almost) anything with git
  - [https://github.blog/2015-06-08-how-to-undo-almost-anything-with-git/](https://github.blog/2015-06-08-how-to-undo-almost-anything-with-git/)
- Git cheat sheet by Github
- Git FAQ – answers to common questions
  - [http://gitfaq.org/](http://gitfaq.org/)
  - [https://git.wiki.kernel.org/index.php/Git_FAQ](https://git.wiki.kernel.org/index.php/Git_FAQ)
- Git pretty – troubleshooting flowchart
  - [http://justinhileman.info/article/git-pretty/](http://justinhileman.info/article/git-pretty/)
Git tooling

  - Simplify complex interactions
  - Draw commit graphs, overviews of branches and merges

- **GitHub Integration**
  - GitHub also provides git tools: [https://desktop.github.com/](https://desktop.github.com/)

- **Git helpers**
  - Common git commands bundled ([https://github.com/tj/git-extras](https://github.com/tj/git-extras))
  - Git shortcuts for common operations ([https://github.com/scmbreeze/scm_breeze](https://github.com/scmbreeze/scm_breeze))
Further Reading

- **Learn Wireframing.** [https://github.com/dwyl/learn-wireframing](https://github.com/dwyl/learn-wireframing)
- Akif, R. and Majeed, H. 2012. *Issues and challenges in Scrum implementation.* International Journal of Scientific & Engineering Research. 3(8). pp.1-4. [https://communities.bentley.com/cfs-file/__key/communityserver-discussions-components-components-files/5917/Issues_2D00_and_2D00_Challenges_2D00_in_2D00_Scrum_2D00.Implementation_2800_1_2900_.pdf](https://communities.bentley.com/cfs-file/__key/communityserver-discussions-components-components-files/5917/Issues_2D00_and_2D00_Challenges_2D00_in_2D00_Scrum_2D00.Implementation_2800_1_2900_.pdf)
## Team Management & SM Tools
- Goals of SM (Tools)
- The (Scrum Master's) Diary
- Scrum Checklists
  - Build your own
- Retrospectives
  - Prime Directive
  - Phase Structure
  - Follow up

## Organizing Work & Dev Team Tools
- The cost of technologies and frameworks
- Project data
  - Task board
  - Issue tracker
- Communicating technical details
- Debugging
- git tricks