UNIVERSITY OF CANTERBURY
Te Whare Wānanga o Waitaha
CHRISTCHURCH NEW ZEALAND

Fabian Gilson

Software Engineering II

SENG301 - Planification to retrospection

Lecture 3 - Semester 1, 2020

University of Canterbury | Te Whare Wānanga o Waitaha
Computer Science and Software Engineering

The best way to get a messed up project is to start earlier than the basic requirements have been defined.

- Mario Fusco

Twitter – 2019

Scrum lifecycle

Scrum lifecycle focus on sprints

Scrum ceremonies

Initial startup

Some bootstrap effort is needed before starting

- start from a **rough vision** of the product
- **refine** its objectives
- **discuss** with the stakeholders
- create an **initial backlog** large enough for a sprint
- agree on **working mode** and **standards**

Scrum and Kanban are quite different here

scrum larger initial phase is needed before planning the first sprint

kanban task-oriented, so faster start-up

Remember to also agree on **coding standards**, **communication** channels and **tools**

Fresh restart at every sprints

In Scrum, every sprint must be planned

- requires some **cleaning** by the PO beforehand
- **other members may assist** cleaning/refining when needed
- **priorities** must be clearly stated (often with all stakeholders)
- set up a **global goal** for the sprint (*e.g "user profiling"*)

Priorities should be discussed on familiar ground

- chunking in **estimated tasks** may help
- every implementation task must be **sufficiently described**
  → mockups, design and updated acceptance criteria, if applicable

⚠ Sprint planning is no place for **deep requirement/architecture engineering**

Sprint planning

Part I - what are we going to do?

- the PO **presents** highest priority **PBIs**
- **first draft** based on previous velocity (*i.e.* number of *points* delivered previously)

Part II - who's going to do what?

- selected PBIs are **broken down into tasks** by the team
- **estimations** of tasks are set **collaboratively**, *i.e* reaching a **consensus**

Part III - how are we going to organise our time as developers?

- start from **higher priority PBI** and related broken tasks
- **iterate** until the planning is (almost) full

⚠ Developers are **not yet committing** to the set of tasks (*i.e.* no upfront planning)

Making estimates

" It is better to be vaguely right than exactly wrong."

– Carveth Read, Logic, deductive and inductive (1898)

Don't fall into the *student's syndrome*

- natural tendencies to **under-estimate** the job to be done
- think about an estimate and **double it for the few first sprints**
  → don't forget that daily scrum, reviews, emails,... and **design** take time

Be aware of cascading delays

- your **delay** is **passed to any dependent** tasks
- when massive delays occur, **re-evaluate the plan** as soon as poosible

Decide between **story points** (relative) *vs* **hours** (absolute)

Planning poker

- use Fibonacci-like progression to reflect **progressing uncertainty** in estimates
- usual values are **1/2, 1, 2, 3, 5, 8, 13**, 20, 40, 100 and **infinity (∞)**
- occur **after initial backlog** and when **new PBIs** show up

5 minutes game, *but it's no Texas Hold'em*

Team up with your neighbours in front or behind you (by 5 or 6)

- discuss about the two stories listed hereafter
- agree on an estimate
- keep in mind these are **relative** values, but have a **meaning** to you

Story 1

- As a traveller, I want to subscribe to email price alerts for a flight I have been just searching for so that I can (potentially) get the best price for it.

Story 2

- As a traveller, I want to see a chronological summary of all pictures I've added for one particular trip I made so that I can show all the nice spots I've seen to my friends.

Monitor, review and go forward

Daily stand up

Short synchronisation point every day

- for both Scrum and Kanban
- must be short and **straight to the point**

Three simple questions

- what did you **achieve yesterday**?
- what will you **do today**?
- are you **facing any issues** or impediments?

Daily stand up

- should last **5 minutes per member** at most
- is the place to identify impediments and **seek help** or **take actions**

Review the task or sprint

At the end of a task (Kanban) or sprint (Scrum)

- demonstrate **the outcomes** to the product owner and stakeholders
- gather **feedback** from them and keep trace of it

Be kind, be interesting, be prepared, be focused

- follow a **clear scenario** by combining the user stories
- use some **realistic test data**, but don't flood with data
- ensure you have the **necessary logistics** ready
- start by **setting the scene**, explaining the **impediments** and how you **tackled them**
- possibly include a *coming soon...* section

→ stakeholders are often far away from the team, so you have to **convince** them

Retrospect the sprint (I)

Remember the scrum core values

openess welcome comments and constructive critics

courage is needed by all team members to face issues

respect always be constructive and respectful in your comments

focus come up with SMART action points that improve your next sprints

commitment commit yourselves to resolve any issues

Location is important

- large and impersonal rooms enforce feeling of inquisition
- prefer coffee shops or break rooms

Come prepared with your issues and suggestions communicated in advance

Retrospect the sprint (II)

Discuss issues about

communication	with PO, stakeholders or in the team
processes	method, software, standards, versioning,...
scope	is the product vision still clear to everyone?
quality	is the product hitting the expected level of quality?
environmnet	is the team dynamic becoming toxic?
skills	is training or external expertise needed?

Just ask yourself these two simple questions

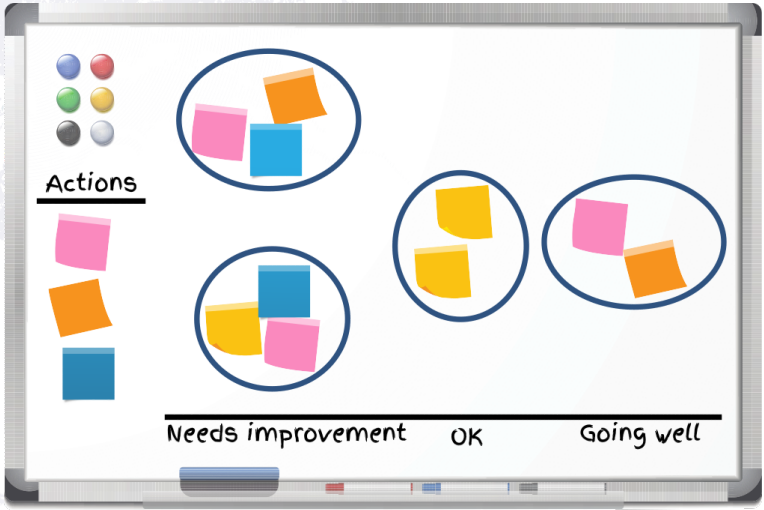
? what are we doing well and what can we improve **?**

Maintain a list of action items, **but do not try to solve them all at once**

→ The action points must be evaluated at **next retrospective**

Retrospect - Bubble method

Retrospect - Circle method



Sprint work

The fine art of breaking down tasks

INVEST in your stories

independent - negotiable - valuable - estimable - small - testable

Be SMART with your tasks (and action items)

specific everyone must understand the objectives and outcomes

measurable agree on what *"DONE"* means

achievable you are able to achieve it with the resource you've got

relevant tasks do relate to stories or "admin" work

time-boxed estimated or further breaking down is needed

Make it clear what *ready* and *done* mean

All members must agree on a definition of done and ready

- make clear the expected **level of quality**
- **minimise arguments** between members and the PO

Different definitions may exists

- story-level (regarding INVEST)
- code-level (regarding pair programming or code review and unit tests)
- documentation level (regarding pair review and availability/communication to others)

Don't forget to frequently review these **definitions of ready** and **done**

A definition of *ready* - proposition for stories

estimation story must be estimated

criteria acceptance criteria are clearly defined

ordered story has been ordered into product backlog

documentation any additional resources have been attached

sprint all tasks from the story will stay within one sprint

A definition of *done* - proposition for stories

acceptance	all tasks have passed the acceptance tests
regression	added feature does not break current system
deploy	any relevant build and deployment scripts have been updated
review	the product owner has reviewed the new functionality
documentation	end-user documentation has been written or updated

A definition of *done* - proposition for coding tasks

unit tests no code should be marked as done without attached tests

documentation code must be documented and commented appropriately

review code should be written in pair or (formally) reviewed by a peer

versioning code should be pushed onto the versioning repository

build new code may not break current build

board task board should be updated with $RemainingTime = 0$

A definition of *done* - proposition for releases

review product owner has reviewed the whole content of the release

deployment all code has been deployed successfully

smoke test subset of new functionality has been tested on production data

training target users have been trained on new functionality


TO BE
CONTINUED...

Tell me what you want

How do we understand the people and their work environment?

How to be sure we develop the right thing right?

How do you want your cake sliced, Madam/Sir?

... come next lecture