ADDENDUM: Agile Process Description

Content

1.1 Scope 1
1.2 Initial Effort Estimate 1
1.3 Agile Project Management Methodology 1
1.4 Specific Policies 4

1.1 Scope

The aim of this contract is to establish a common understanding, terminology, and fair basis for a joint agile development of software components and services for FAIR as principal client and the contracting body.
Software development projects implementing new functionalities are inherently difficult to manage with conventional waterfall-type contracts that specify high-level product requirements and in turn shall be used to provide precise time and cost estimates. Inherent shortcomings of these processes are that many specific requirements and their solutions often only evolve as new or more precise information becomes available. Initial effort and cost estimates are therefore commonly very imprecise and can only be improved after a number of design and development iterations.
The present 'agile development contract' aims to provide a better solution to the management of such software development projects by

A) mitigating the inherent shared risks of either over- or underestimating the time and resources required for completing the project, or to fall short of the actual required development targets, and at the same time
B) facilitating opportunities that efficiently improve the project value and overall outcome for all parties involved while minimising the required administrative overhead.

The contract defines common practices. Those common practices are largely based on management principles like lean management (Toyota Production System[1]) and continuous improvement, non-dogmatically borrowing elements from software development methodologies like Kanban, CI/CD, Extreme Programming and Scrum.

1.2 Initial Effort Estimate

Without prior in-depth domain knowledge or existing prototype implementations, there will be technical uncertainties for both parties that likely will lead to either an over- or underestimation of the project complexity. In the interest of both parties, this risk should be sufficiently addressed at the start of the project and before significant resources are invested into the project development.
In absence of an existing detailed analysis or design, the contractor shall thus perform and document a preliminary study with participation of the client *(*more precisely, the client's *product owner* and relevant *domain experts)* to improve the scope and mutual understanding of the functional user requirements. The resulting refined user requirements form the '*product backlog*' that is broken down into *epics* and prioritised *user stories.*
As an objective, this jointly developed product backlog shall define an improved estimate of the required total implementation effort in terms of *story points* (SP) for each backlog item, tentative lead-times in *working days* (WD) as well as an upper time-limit for completing the backlog.
After the initial analysis, both estimates and, in addition, a fixed price shall be agreed upon, kept as a fixed project target reference, and are part of the offer and awarding the CALL.

1.3 Agile Project Management Methodology

Continuous improvement as well as communication and respect for people are key ingredients for any successful cooperation, sustainable developments, and for providing lasting value. The following section provides guidance for the kind of agile practises that shall be employed.

1Ohno, Taiichi (1988)."*Toyota Production System*.", New York, NY: Productivity Press. pp. XV. ISBN 0-915299-14-3

Since following good rules or methods alone does not guarantee success, rather than being hard self-serving rules, the aim of these guidelines is to provide a common understanding of the flavour of the envisaged development process. These guidelines may be adapted as necessary in mutual agreement to minimise wastes that do not contribute to the overall product or collaboration value.

The software development process for FAIR follows a lean pull-driven workflow management methodology for defining, implementing and iteratively improving software functionalities. It relies on *Kanban boards* to

a) help visualise the individual work items as well as work-in-progress (WIP)
b) coordinate the workflow from *features* (i.e. functional user requirements), that are broken down into smaller *(user) stories* and *epics* (i.e. collections of *stories,* defining overarching goals and constraints), and are further tracked through the development, testing and deployment steps.

An exemplary kanban board is shown in

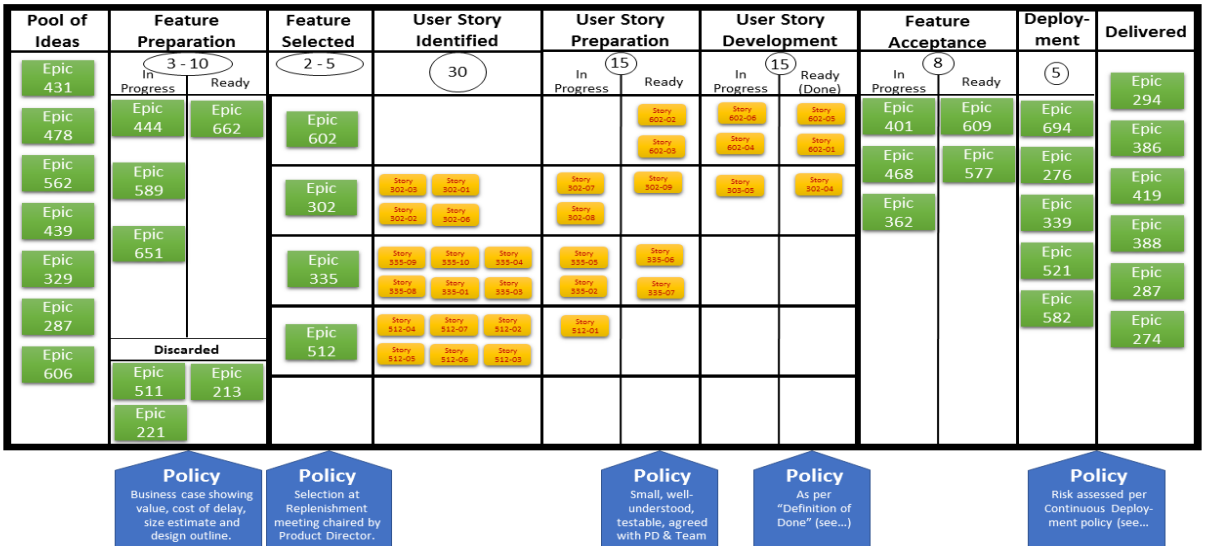

Figure 1: Typical Kanban board with 2 work item types (user stories and epics*). Source: Wikipedia*

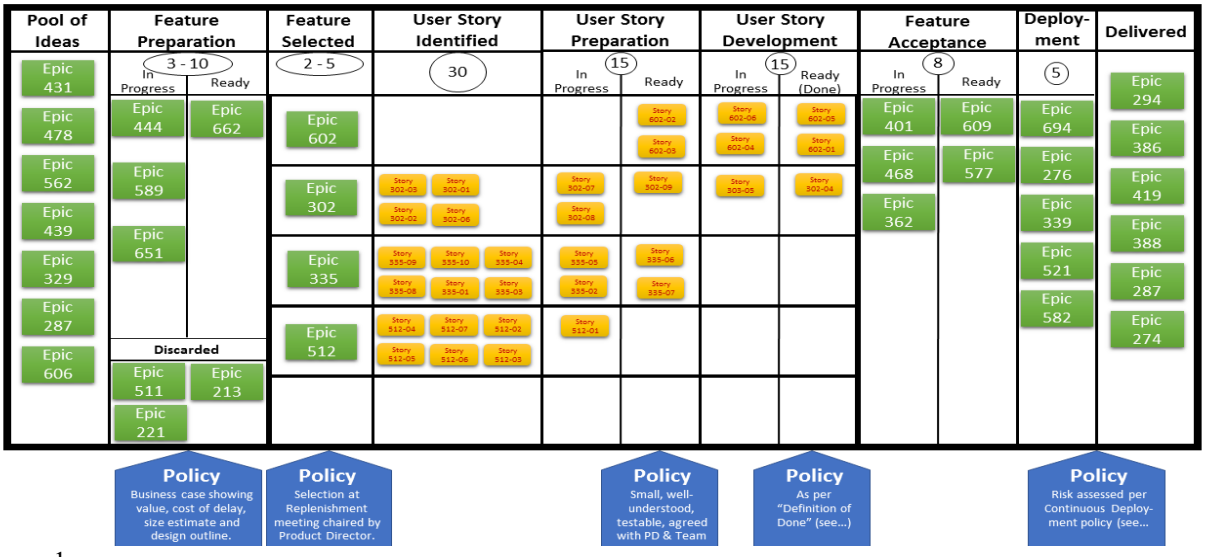

Figure 1.

Kanban boards describe and directly map the asynchronous workflow of the largely self-organised development team by defining *WIP limits* (number in ovals in in

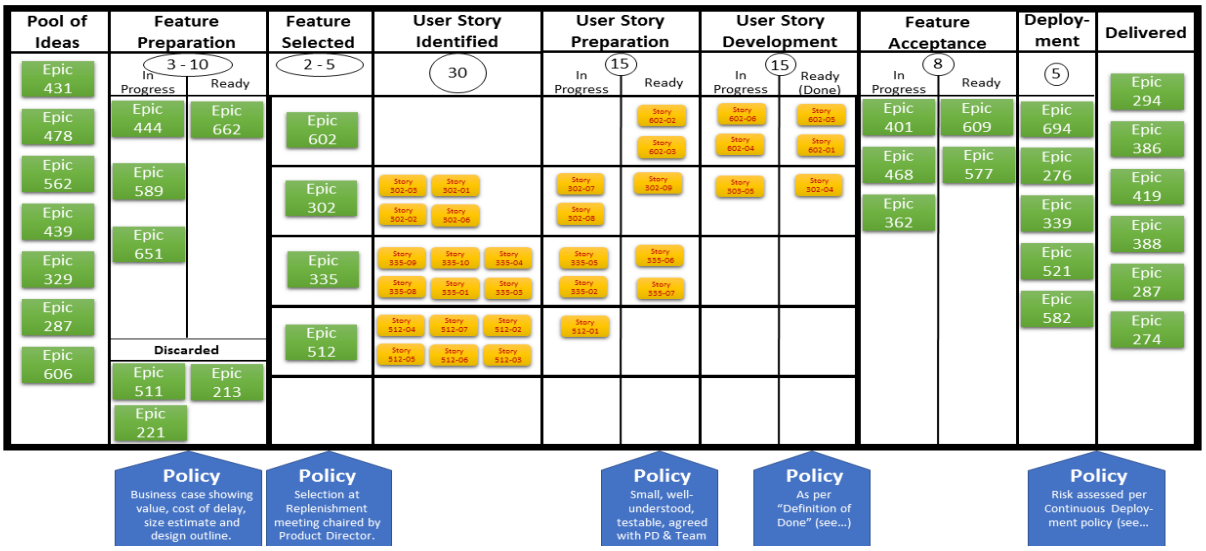

Figure 1) which sets the maximum number of active epics/stories in a column, and explicit *policies*, which work particularly well for teams that are physically distributed and/or that have different working hours (or time-zones). The WIP limits for development steps (typically linked to team capacity for preparing, developing, testing, reviewing of work items, etc.) provide team members immediate feedback on common workflow issues (tasks/epics/stories or workflow issues).

Where necessary, policies (blue boxes in

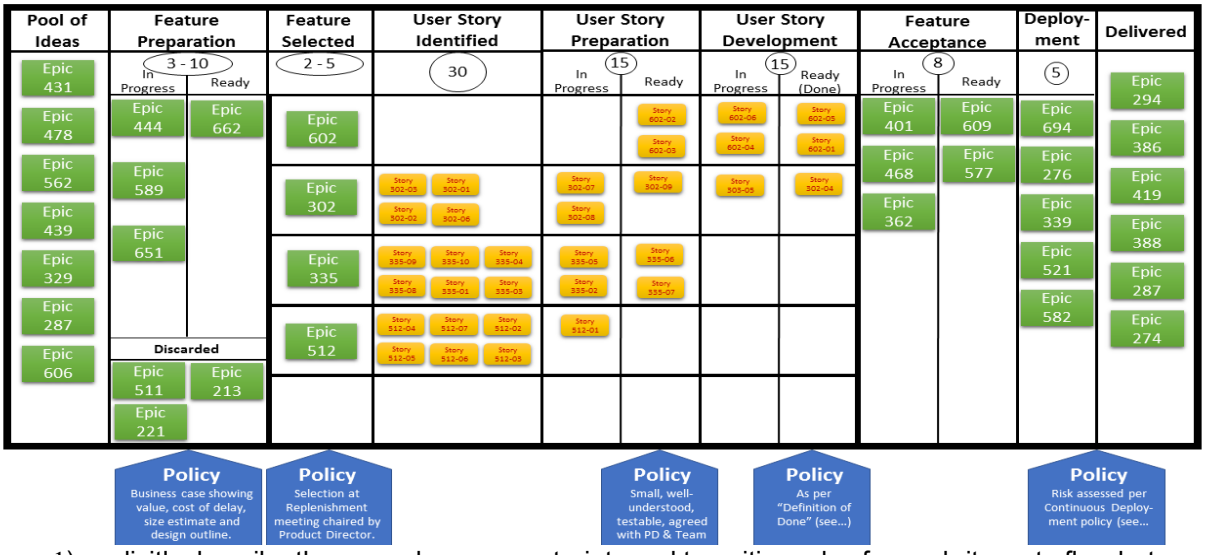

Figure 1) explicitly describe the agreed-upon constraints and transition rules for work-items to flow between the individual steps. Some of the policies commonly deployed at FAIR are outlined in Section 1.4 'Specific Policies'.

The Kanban-based process further uses a limited number of Scrum elements (aka. S*crumban*) to limit the otherwise needed larger daily meetings to, for example, bi-weekly or monthly *sprints* with its customary *sprint planning*, *sprint review*, and *product backlog grooming* steps and (optionally) very short *daily meetups*. An

exemplary Kanban workflow is shown in

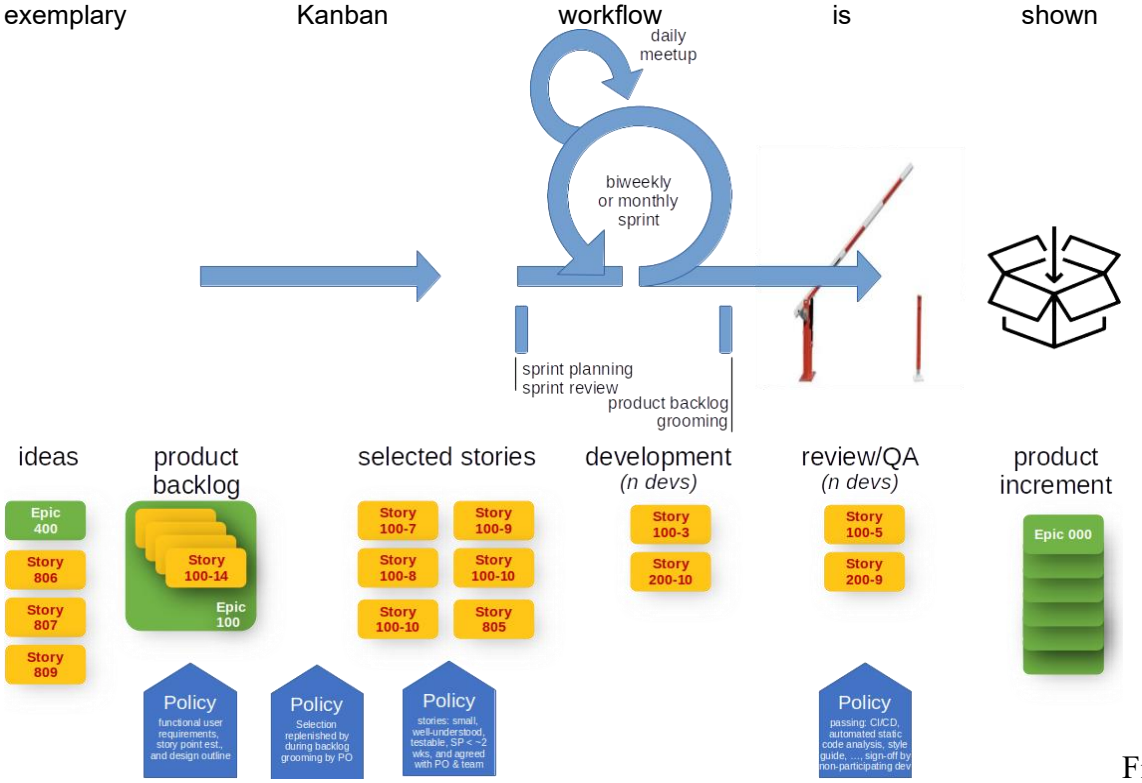

Figure 2.

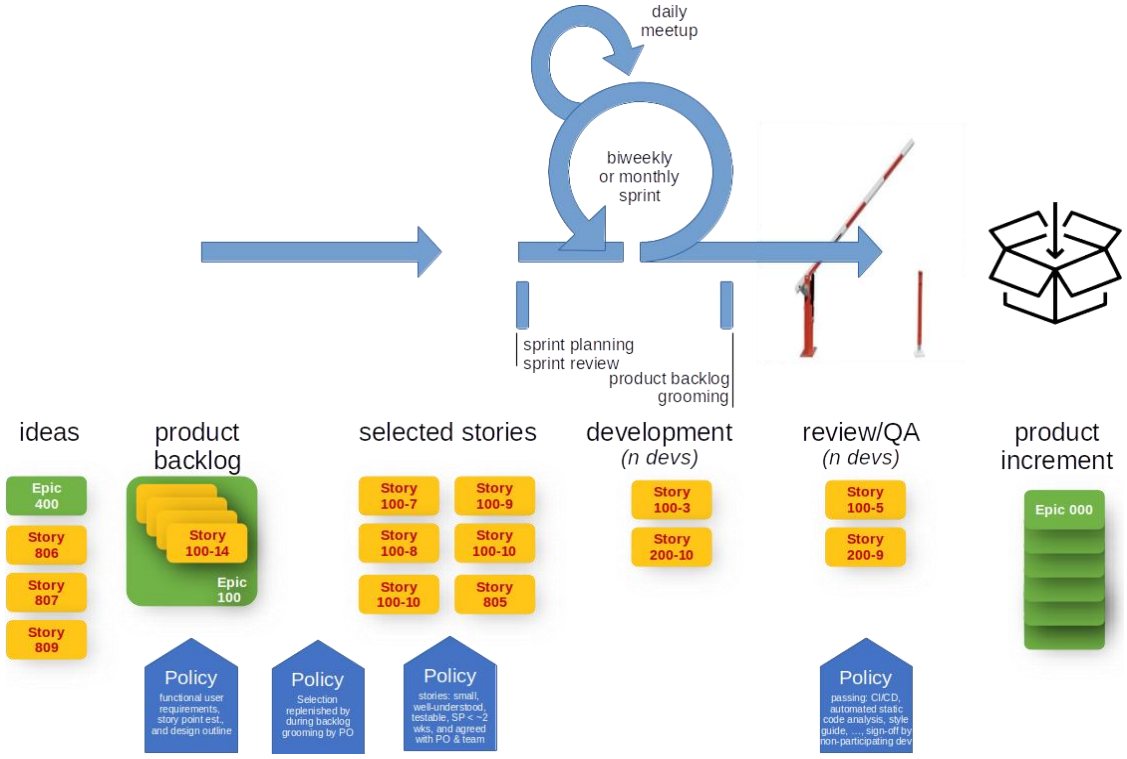

Figure 2: Exemplary Kanban workflow with policies and Scrum elements

While some Scrum elements are being used, there are some noteworthy differences:
sprints are largely planned and self-coordinated by the development team and steering board (i.e. there is no dedicated or formal-certified 'Scrum Master'). While it is recommended that work items are broken down into user stories that can be completed well within a sprint, this is not a strict necessity[2]. Work-items are not strictly time-boxed and may be larger (or smaller) if this is functionally easier or more efficient.

2 Experience has shown, that a too strict constant timing or control processes do not necessarily improve the quality or productivity, notably when tasks are intrinsically complex, cannot be meaningfully be broken further down without loosing core functionalities, require certain level of digestion period (e.g. waiting for expert feedback), may

As part of the sprint planning process, work-items are moved from *ideas* to the (accepted) *product backlog* (i.e. functional user requirements) by the steering board. Work-items in the product backlog are then prioritised by the team and steering board and moved to the '*selected features/stories'* categories. From there, each development team member can independently and freely select (i.e. *'pull'*) the next work item once they completed their previously assigned work item or while they are blocked from processing it. The purpose of the optional but pre-agreed *daily meetup* – that should typically last less than 15 minutes – is to detect and efficiently resolve blocking issues early on. To improve the development flow, the frequency of this daily meetup may be reduced or it may be skipped altogether if there are no blocking issues or if the issues are solved by other means. In any case, the team should maintain low-key communication channels for the cohesion of the team and to early detect and remedy arising issues ('*poka-yoke*' or '*error proofing*' principle).

in many cases require less time than the agreed sprint period (→ wasting time), or basic absence of team-member due to holidays or due to not working exclusively on only one project (common for cross-functional teams, e.g. DevOps).

Common definitions and terminology

- *(user) story*: informal, natural language description of features of a software system. They are written from the perspective of an end-user or user of a system.
- *epic*: grouping of functional user requirements consisting of multiple user stories.
- *product backlog*: ordered list of functional user requirements, grouped into *epics* and (user) *stories*.
- *product backlog grooming*: ongoing process at the end of each sprint to review/amending/updating/re-prioritise product backlog items in the light of new information, and to replenish the '*selected stories*' queue.
- selected stories: collection of priorities pending stories that can be tackled once the development capacities allow.
- *development or work-in-progress (WIP) queue*: stories that are actively worked upon. The queue size is typically limited by the number of available developers.
- *sprint, sprint- planning & review*: pre-agreed time-period after which the ongoing (N.B. not necessarily time-boxed), WIP items are reviewed as part of the *sprint review* (back-ward looking) and *sprint planning* (forward looking).

1.4 Specific Policies

- **Change for Free**: new requirements not included in the initial estimate may be added during backlog grooming by the steering board provided that they replace an existing requirement or set of stories that are not yet being implemented and that have an equivalent estimated effort. De-prioritised stories that exceed the initial target effort estimate are subject to the safety-margin.
- **Early Finish & Risk Share (Safety-Margin):** in case the contractor finishes the product backlog early with respect to the initially planned target reference, FAIR will pay 50% of the remaining story-points value. In case the actually required story points increase with the target reference due to – at the project start – unforeseen complexities, both parties will share the remaining risk and required remaining story point effort 50/50 up to a maximum of 10% each of the total project value (safety-margin).
- **Steering Board**: consists of
  - authorized representative of the client (FAIR)
    - must assure availability of FAIR-specific technical expertise if needed
  - authorized representative of the contractor
  - product owner (FAIR)

- **Sprint and Review Iterations**:
  - bi-weekly or as agreed by the steering board
  - Optional: 'daily Scrum' meetings at predefined times (usually <15 min),
    - often (mostly) only twice per week to quickly identify and resolve possible blocking issues.
    - Meetup can be omitted if there are no blocking issues, questions, etc. within the team or if these have been resolved beforehand.

- **Product Backlog**: if activities have been processed more quickly (i.e. <two weeks), activities from the pre-agreed and pre-scheduled list may and can be picked up and started independently by the developers. Rationale: reward of 'Early Completion' and 'Money for Nothing'.

- **'Extreme Programming' and Test-driven Design paradigm**: identify performance, API, or other bottlenecks early on through limited prototypes that the product owner can test and provide feedback (an important pre-requisite of the change for free principle)
  - development of technically critical items first
  - limited reduced prototypes (unit-tests) to identify issues or other bottlenecks at an early stage
  - strongly favour early narrow unit-tested vertical integration prototypes rather than widening the range of non-fully tested functionalities of the software stack

- **QA Process**: is largely automated, git-driven CI/CD[3]-driven development process largely following concepts outlined at https://githubflow.github.io/:

3 CI/CD: Continuous Integration/Continuous Delivery

- ‘main’ branch shall only contain and reviewed, QA tested, and deployable working code
  - N.B. the code does not have to be feature-complete though
- new functionalities shall be developed in 'feature branches' that are – after having passed the QA process – rebased and merged via pull-request (PR) into ‘main’
  - individual commits are usually squashed according to sub-functionalities before or as part of being merged into the ‘main’ branch
  - intermediate results need to be pushed as needed to the feature branches but at least once per working week to aid parallel-review and faster and more efficient PR reviews by FAIR.
- The QA process (‘PR review’) consist of two steps:
  - an automated CI/CD tests, and
  - sign-off by a non-participating developer of the other party which in turn is used to confirm the formal FAIR QA and triggering of the intermediate invoice/payment
- The automated CI/CD acceptance tests typically involve
  (*N.B. existing rules may be mutually adapted after a cost-benefit assessment. All developers are strongly encouraged to already test their ’feature branches’ with regard to CI/CD and QA requirements on a regular basis (e.g. daily)):*
  - static-code analysis (e.g. clang-tidy, linting, Coverity or similar)
  - automated sanity and style checks (e.g. clang warnings)
  - automated code-formatting (e.g. clang-format)
  - code-coverage goal: (strive for) that the code-base is covered by both class-level as well as functional (e.g. proof-of-concept) unit-tests for the primary and likely corner use case
  - untested code should be the exception and must be justified (e.g. often very complex in UI-level user input testing)