Requirement Management Process

Table of Contents

1. Abbreviations used 2
2. Participants 3
   2.1. Steering Committee 3
   2.2. Product Owner/Product Manager 3
   2.3. Developer team 4
   2.4. Architecture Committee & Quality Committee 4
   2.5. Test team 4
   2.6. Summary 5
3. Requirements 6
   3.1. Requirement template for Epic/Story 6
   3.2. Requirement Template for Bug Reporting: 8
4. The life of a requirement 9
   4.1. Creation 9
   4.2. Clarification 10
   4.3. Ready for development (aka sprint ready) 11
   4.4. Realization and internal review 11
   4.5. Approval & Testing 11
   4.6. Debts of former sprints 12
   4.7. Final remarks 12
5. Direct comparison of different suggestions 14
   5.1. Suggestion by the requirements management process service (Canoo) 14
   5.2. Suggestion by Sven Wittig 16
   5.3. Conclusion 18

This document explains the proposal and includes the discussion of the requirement management process of the OpenMDM project by demonstrating the life cycle of a requirement request. First the participants in the process are named. Afterwards the different stages of a requirement towards implementation and release are outlined. The thoughts and process presented here are proposed by Canoo Engineering AG and include the discussions and conclusion of the requirements workshop on December 8, 2014.

Chapter 1. Abbreviations used

- AC: Architecture Committee
- EWG: Eclipse Working Group
- QC: Quality Committee
- PO: Product Owner/Product Manager
- SC: Steering Committee

Chapter 2. Participants

The following people may influence or are directly involved in the processing of any requirement and are therefore briefly described:

2.1. Steering Committee

The committee members can file requirements like everyone else but are not directly involved in the concrete realization of a desired functionality. Instead the committee sets boundaries by defining the overall goals for each release. The product increment as being the result of each release is approved or denied as a whole by the steering committee.

2.2. Product Owner/Product Manager

Product Owner is the (see Scrum methodology). This role is responsible for the definition and priorisation of the functional and non-functional requirements of the product. She is responsible for shaping the vision together with the stakeholders and make sure that everybody building the product understands it (interface between stakeholders, sponsor and development team). The product owner has the competence to decide what should be build to create as much value as possible as quickly as possible. In Scrum requests for and issues of a product are managed in the so called **backlog** - the product owner is in charge of maintaining and prioritizing the backlog.

2.2.1. What does that mean for the OpenMDM EWG?

To get a successful product we think that is crucial to have a dedicated person to fulfill this product owner role. We propose to choose an independent person and pay it using a work group service. Very often the engagement of a product owner is underestimated, but since she is basically responsible for success or failure of the product, it is essential that this person has enough time and knowledge. The OpenMDM product owner is responsible to collect and evaluate requests from the community (backlog management). She is also responsible to bundle and select the most valuable and efficient issues needed to reach the roadmap goals defined by the steering committee. Ideally the product owner escorts the development of packages/blocks of development - as she is the only person who can make quick decisions (short feedback cycle). The OpenMDM product owner reports to the steering committee and signs off the delivered product increments. She is the primary person which defines **WHAT** (functional requirements) is implemented. But it is important that the product owner is a good team player since the architecture committee and the quality committee define the **HOW** (non-functional requirements) of the product - and the team (service provider) is in charge to estimate the effort and to develop the requirements in the required quality in the simplest way with the least effort possible.

Discussion of the workshop 8-12-2014 Sven Wittig: Sven only sees the need of such a role in the collecting and bundling of the request from the community (requirements service). The bundles/packages then are presented to the steering committee and provided they get a sponsor, are

then implemented according to the architectural and quality guidelines. The escorting of the service provider is the sole responsibility of the sponsor. Basically Sven Wittig assumes a **bottom up** (community driven) approach only.

Ulrich Bleicher, Andreas Benzig, Sibylle Peter (and others - please specifiy ???): While they see a possibility of a package product owner which is responsible for the implementation of such a package, they still think a global OpenMDM product owner is needed to provide conformity and congruence for the product. They think it is necessary to have a **top down** (steering committee provides initial vision and roadmap for the product) as well as a **bottom up** approach

This discussion led to another interesting and important question, again from Sven Wittig: What is the product? Is it only a set of building blocks, loosely coupled, configurable and working together or is the product a system consisting of loosely coupled building blocks, which allows the community users to execute the most common shared use cases?

IMPORTANT	these questions (what is the product and which approach(es)) should be discussed in the steering committee asap.

2.3. Developer team

The developers are responsible for the concrete implementation of the specified requirements. The team supports the product owner by pointing out costs, dependencies and risks linked with the realization of the requests. The developers are responsible for creating the shippable product at the end of each release iteration. In most cases a service provider will bring a complete team. At least one member of the team must be committer of the corresponding eclipse projects.

2.4. Architecture Committee & Quality Committee

The committees elaborate standards and guidelines the developers commit themselves to adhere to. The committees reflect about and revise their regulations in regular intervals. They are responsible to define **HOW** the product is to be build and thus support the product owner in the definition of non-functional requirements.

2.5. Test team

A designated test team verifies that the implementation meets the defined acceptance criteria and does not breach any regulations. This test team does **not** replace the automated unit, integration and functional tests implemented by the development team.

Discussion 8-12-2014: It became obvious that this part of independent testing was neglected so far. The discussion led again to the importance of defined and automated test compatibility kits (TCK). We established, that there need to be at least two kits: . A functional TCK, specific for each component to test the functionality (WHAT) of a specified component. . A general TCK that is applied to all components to assure that the non-functional requirements (HOW) of architecture and quality are met. Later, Sven Wittig suggested that the test team should consists of the members of the quality

service.

2.6. Summary

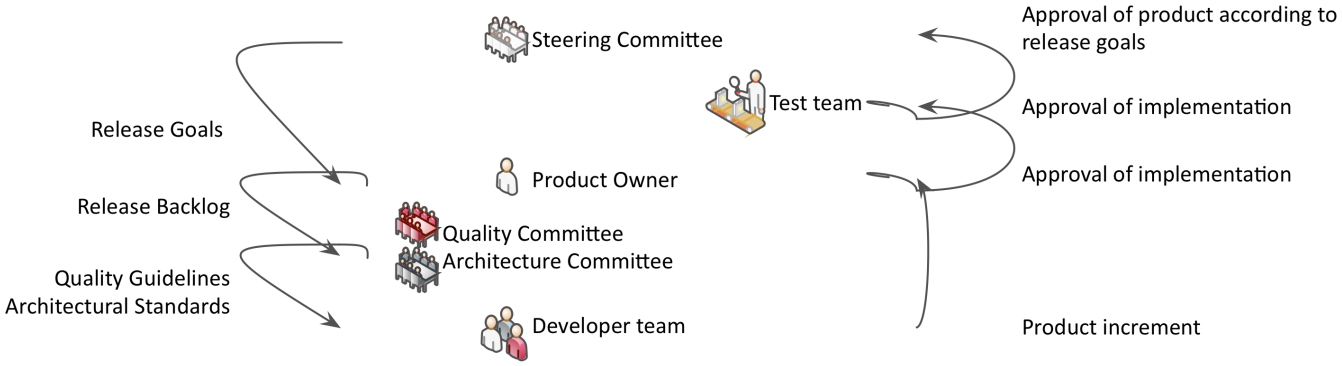


Figure 1. The participants and their responsibilities

Chapter 3. Requirements

We suggest a light weight way to manage requirements using a backlog. A backlog is place where requirements are collected, grouped, sorted and prioritized due to business value (for further explanation of the term backlog, see http://en.wikipedia.org/wiki/Scrum_%28software_development%29#Product_backlog). We suggest to use a sophisticaed issue tracking tool (like JIRA or YouTrack) to use as backlog as they offer advanced support by managing the backlog (filters, planning, different issue types etc.)

Dependent on the issue tracking tool used, we propose the following issue types (issue == a general entry in the backlog):

- Epic: a big user story, feature. Takes certainly more than one iteration to implement
- Story: a user story, broken down so that it is possible to implement in one sprint
- Improvement: An improvement (change) to an existing story/task
- Change request: A new story which is outside of the current scope - so either another story/epic of same size has to be removed or additional capacity is needed to keep the original release goal
- Bug: a problem which exists in the current code base and impairs the function of the product.
- Task: a task that simply needs to be done
- Technical Task: a technical task that simply needs to be done
- Subtask: a sub task of any issue

By using Epic and Stories as basic form to capture requirements, we accept that requirements are also subject to change. Therefore it is important to choose the right granularity for each context. After creation, the requirement should provide enough information to be understood by others and to be packaged by the PO (regardless if this is a person or the requirement management service) and prioritized by the SC. Of course to make estimations about the effort of implementation mostly more information is needed. We suggest that by creating a requirement the reporter (creator) agrees to deliver as much as needed initially (see template), but also agrees to be available for further detail specification and clarification once the requirement has been chosen to be implemented.

Since tasks are mostly so low level, that they are clear, we provide templates for epic ans story, as well as bugs only.

3.1. Requirement template for Epic/Story

In order to ensure a smooth realization the requirements should already provide as many details as possible right from the beginning. At least the following information should be given:

- Summary

  A short and yet precise summary eases the search for the requirements as it spares navigation effort in the issue tracking system.

- Description

  The description should express which user group acting in which role (rights management) shall be enabled to execute which actions on the component. In short, the description should stress the benefit (business value) of the requirement rather than telling how sth should be implemented.

Proven format of user stories in different context are the following (but other forms as goal directed use cases etc. are also ok).

```
In order to [benefit]
as a [role]
I want [feature]
```

or

```
As a [role]
I want [feature]
So that [benefit]
```

To make a requirement ready to estimate, further detail information is needed. In agile project, defining acceptance criteria extremely helpful for estimating. Acceptance criteria also help to define and narrow the scope of the requirement and support testing. Using examples to describe acceptance criteria is very useful, because they are easily understandable also by non-experts and allow quickly to gain a shared understanding. Acceptance criteria written in the following form are easily testable - with the right tools, an automated living documentation of what the product/component does can be built. This also eases the review and approval process of a component.

- Acceptance Criteria: (presented as Scenarios)

  ```
  Scenario 1: Title
  Given [context]
    And [some more context]...
  When  [event]
  Then  [outcome]
    And [another outcome]...

  Scenario 2: ...
  ```

Of course this description can be enhanced anytime by a specific specification document, ui mock ups,

sketches or whatever else is needed to implement the story correctly. These documents are attached to the main story/epic.

3.2. Requirement Template for Bug Reporting:

Given that a flaw (bug) in the software is reported, then the explicit steps to reproduce it should be mentioned along with the actual and the expected behaviour. Also the version of software the bug was found in has to be specified. The more precise this description the easier the developers can advance to the root cause of the failure. If an error message had been displayed it is strongly recommended to attach its content to the bug description. In some cases screenshots of the failing system are of great help.

Chapter 4. The life of a requirement

This section describes how a requirement progresses through several states from its creation to the incorporation into the product increment.

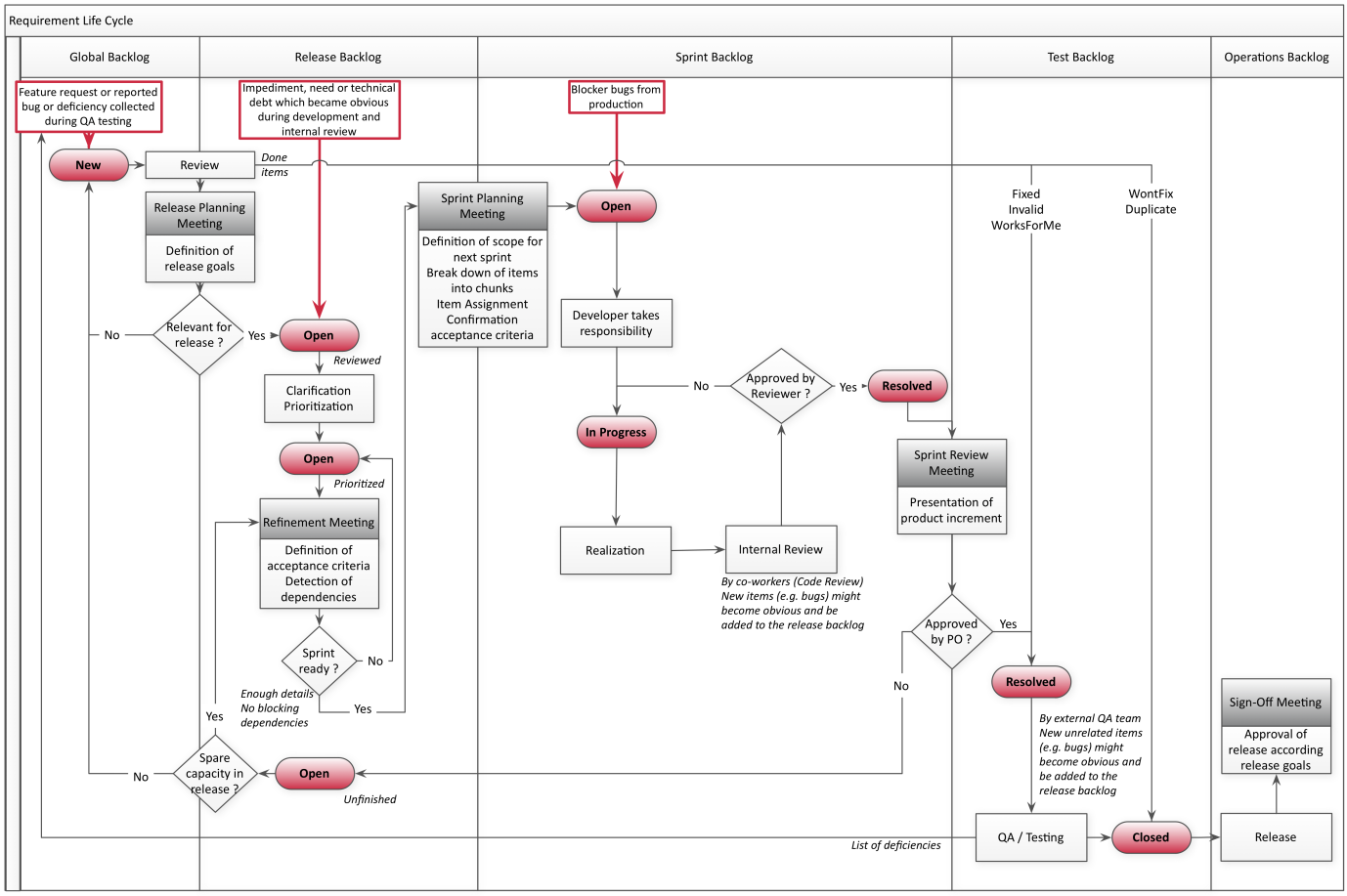


Figure 2. The requirement life cycle

4.1. Creation

New requirements can be filed by **any person** involved in planning, development and usage of the OpenMDM application. This includes the steering committee whose members want the application to provide a certain feature their employees are longing for as well as the developers who are adding functionality to the software and see needs for changes. Furthermore end-users from inside and outside the working-group are welcome to report their requirements and to inform about malfunctions or unexpected behaviour they experienced while using the application. Additionally the quality assurance/test team can file deficiencies they discover while testing the application.

NOTE

As mentioned before, the product owner is responsible to evaluate and classify these new requests. As she is also responsible for prioritizing the present requirements the reporter has no influence on the realization order. Priority or severity flags set by the reporter are only considered as recommendations.

At first the requests are added to the *Global Backlog*, a free accessible reservoir for all requests. As the authors differ some of the requests might already describe in great detail what kind of addendum or fix is desired whereas others lack precision or just state a more general need. While different granularity is possible, they need to be at least ready for priorization in order not to be rejected.

The product owner is in charge to manage the backlog and hence to review all requirements. The review does not necessarily be done by the product owner himself. He may ask a developer for example to confirm a reported bug. Duplicate requests and those decided not to be implemented are **closed** immediately. Requests which are already implemented, invalid or just cannot be reproduced are declared **resolved** indicating the corresponding reason (e.g. fixed, invalid, worksForMe). The review can be seen as a constant clean-up task for the global backlog.

The steering committee (SC) of the OpenMDM working group and the designated product owner (PO) regularly come together to a *Release Planning Meeting* in order to discuss and define the overall, coarse-grained release goals for the next product release. Based on the agreements of this meeting the product owner is then responsible for evaluating if a certain requirement is relevant for the upcoming release. If the issue is of relevance for the next software release then it is moved to the next *Release Backlog* (the Release Backlog is more a filter of the global backlog, not another physical repository) and its state switched to **open**.

4.2. Clarification

Once selected into the release backlog, the requirements have to be made ready for estimation as quickly as possible by adding acceptance criteria and other detail specification (mock ups etc.)The product owner contacts the initiator (reporter) of the requirement request in order to fix its content and scope. Furthermore the PO has to confer with the steering committee to clarify the relevance of the requirement from the business’s point of view. In addition to that the product owner prioritizes the backlog items according to their importance to release success.

In regular *Refinement Meetings* the product owner and the developer team inspect the requirements. The first goal of the meetings is to establish a common understanding of the tasks at hand. Furthermore acceptance criteria need to be reviewed and complemented where needed. Last but not least impediments such as dependencies between items need to be identified. Thereby quality guidelines and architectural standards have to be considered. The developers estimate the realization effort i.e. the costs for each item. The farther the project advances i.e. the more experience the team gains and the more often an item is discussed the faster the estimation will narrow down to a reliable value.

NOTE	Nonetheless estimates are no 'exactimates'. By definition they are prone to some degree of inaccuracy.

NOTE	In the following paragraphs we assume an iterative or flow based agile approach is followed. However, as stated in the discussion from 8-12-2014, it is open to the sponsor and the development team how they are going to implement the issues in the release backlog. Nevertheless regarding the involvement of many players and stakeholders we strongly recommend following an agile approach, which includes frequent integration of product increment. This ensures that learning happens as early as possible. **The later the integration of different parts and components, the later the learning the more expensive the corrective measures.**

4.3. Ready for development (aka sprint ready)

We recommend that content of the release backlog is realized in several iterations, also called **sprints** or several deliveries are made if using a flow system (Kanban). The term sprint is again derived from the Scrum framework and describes a fixed work phase. The items regarded as 'ready for development', i.e. detailed enough, provided with acceptance criteria and not blocked by any known issues constitute the basis for the *Planning Meeting* which is held at the beginning of each iteration/regularly. During this meeting the product owner and the developer team check the available manpower, define the scope of the next sprint, split large items into feasible sub tasks and finalize their acceptance criteria. Moved to the *Sprint Backlog* the issue remains **open** till one developer with free capacity signs responsible for it.

4.4. Realization and internal review

The requirements then enter the realization stage and are declared as being **in progress**.

The specified release backlog items are developed according to the architecture and quality requirements and guidelines.

When at some point the implementation gets blocked this immediately has to be communicated to the sponsor and the sponsor. The solution of these impediments takes priority. Another principle of agile development is that a story is either done or not done. So regardless the cause (impediments, wrong estimates if a story is not finished, it moves back to the release backlog (and most commonly into the next sprint backlog). This is another reason to work with an experienced product owner. She is familiar with story splitting and the smaller the items the bigger the chance they get done!

4.5. Approval & Testing

At the end of each sprint/iteration (but latest at the release if not followed an agile approach), the product owner, as representative of the stakeholders, approves the implementation and test result during a *Sprint Review Meeting*. Together with Architecture and Quality Committee she checks that the software increment fulfills the functional and non-functional demands. If the product owner is not satisfied with the provided solution then the task is **reopened** and put back into the release backlog.

NOTE	This does not prevent the product owner from verifying resolved items prior to the sprint review meeting. If she does not consent a certain implementation and the developer team has capacity left in the current sprint then she may reopen the issue and keep it in the sprint backlog. A developer, most preferably the developer who was responsible for the implementation in the first place (as she possesses most knowledge about it) will take care of it. A reopened task which cannot be fixed within the ongoing sprint is forwarded to the release backlog.

The product increment and the realized items are eventually handed over to the quality assurance and testing department. The testers check once again that all acceptance criteria are met and verifies that the implementation follows the agreed quality guidelines and architectural standards. The test team expresses its accordance by setting the item's status to **closed**. In case deficiencies are diagnosed then **new** issues are added to the global backlog.

NOTE	The advantage of involving the test team after the product owner has signed off the result of the sprint lies in the fact that the testers can work at their own pace (decoupling of work). They are not forced at the end of a sprint to rush through testing to keep the Sprint Review meeting deadline.

After the last sprint of a release has been concluded all then closed issues are integrated in a concrete release deliverable. This deliverable is presented by the product owner to the sponsor(s) of the release and to the steering committee in a *Sign-Off Meeting*. The stakeholders, considering the formerly defined release goals, accept or deny the given solution. Only in the former case the solution is put into production. Either way the next iteration starts with another *Release Planning Meeting*.

4.6. Debts of former sprints

As long as the last sprint for the current release scope has not been started unfinished tasks can be considered in the *Refinement Meetings* for the next iteration. It is the product owner's responsibility to sort the tasks into the prioritized list of backlog items. In case the last sprint is already fixed then the task is, along with all remaining release backlog tasks, moved to the *global backlog* and considered again as **new**.

4.7. Final remarks

The clarification meetings have to take place regularly. Items can only be considered for a sprint planning meeting when they are clarified, described in detail and prioritized. It is crucial that the product owner prepares the designated next sprint's items while the current sprint is still ongoing.

Lessons learned will be gathered in *Sprint Retrospectives* at the end of each sprint which may influence the realization process of upcoming sprints.

At any time, if an exceptional condition arises the product owner is allowed to notify the steering committee. In the following a non-exhaustive list of such conditions is provided:

- a high priority ticket cannot be solved in time
- an impediment requires the decision of the steering committee
- a new highly valuable capability has been discovered that could be turned into a requirement.

In case a blocker issue is diagnosed in the productive system, then it is added to the current sprint backlog at once. The developer team and the product owner have to negotiate which other items are then dropped from the current sprint backlog and postponed till the next sprint.

Chapter 5. Direct comparison of different suggestions

To make the discussion/decision easier, the initial suggestion by Canoo and alternative suggestion by Sven Wittig are presented below with a short summary and a SWOT analysis.

5.1. Suggestion by the requirements management process service (Canoo)

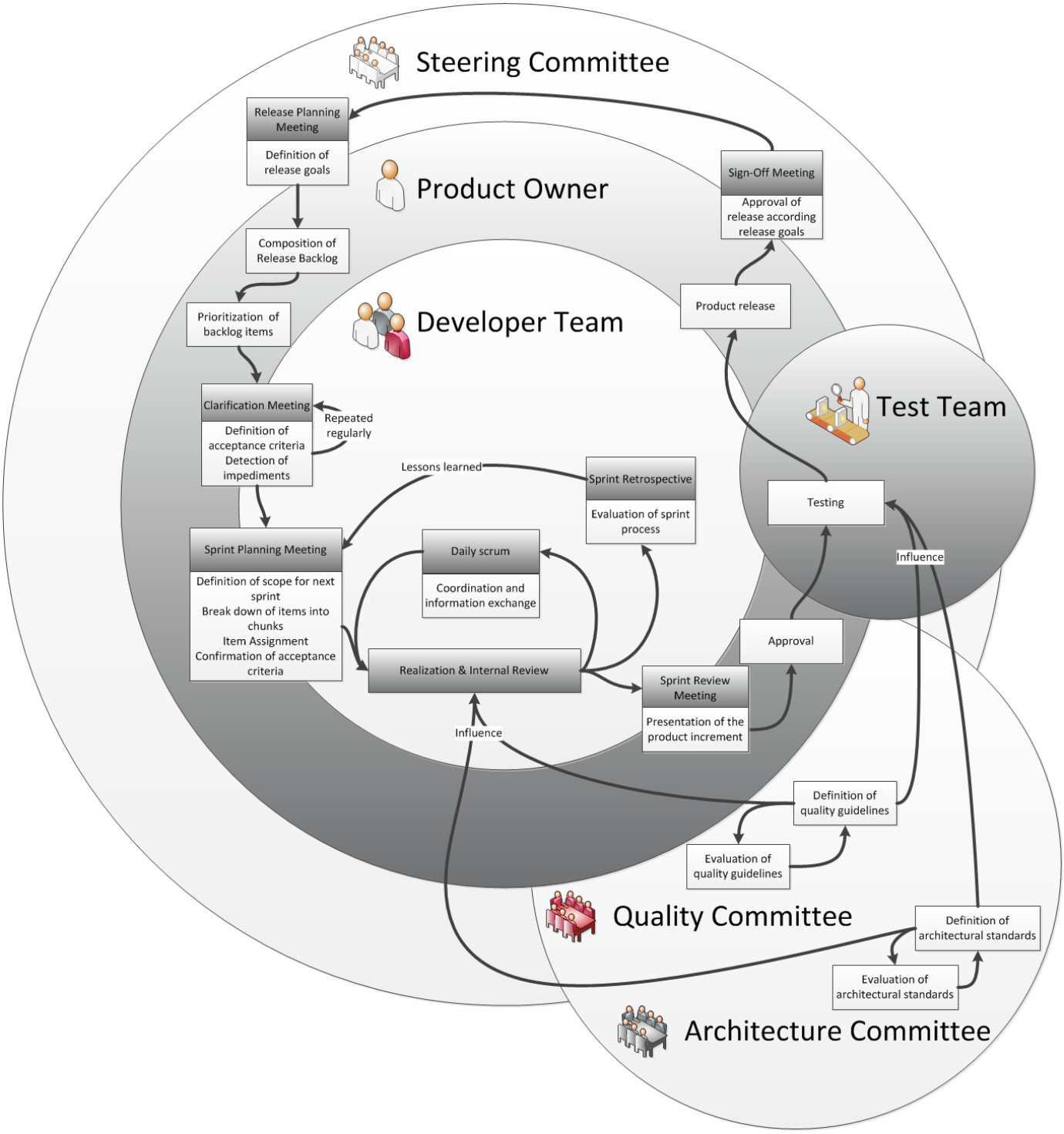
5.1.1. Summary

1. dedicated, independent product manager (Scrum role product owner) which is reporting to the steering committee

2. product manager coordinates all planned (top down) development around openMDM. S/he has know-how of lean product management and can support service provider implementing the product in iterations - product manager is involved in development (no blackbox). Of course the product manager can delegate some of the daily work, but remains responsible.

3. Development which is contributed by the community without having been planned by the work group is of course possible and welcome, the contributions are evaluated by the test team and the product manager

The process is visualized in the following image:



5.1.2. SWOT analysis

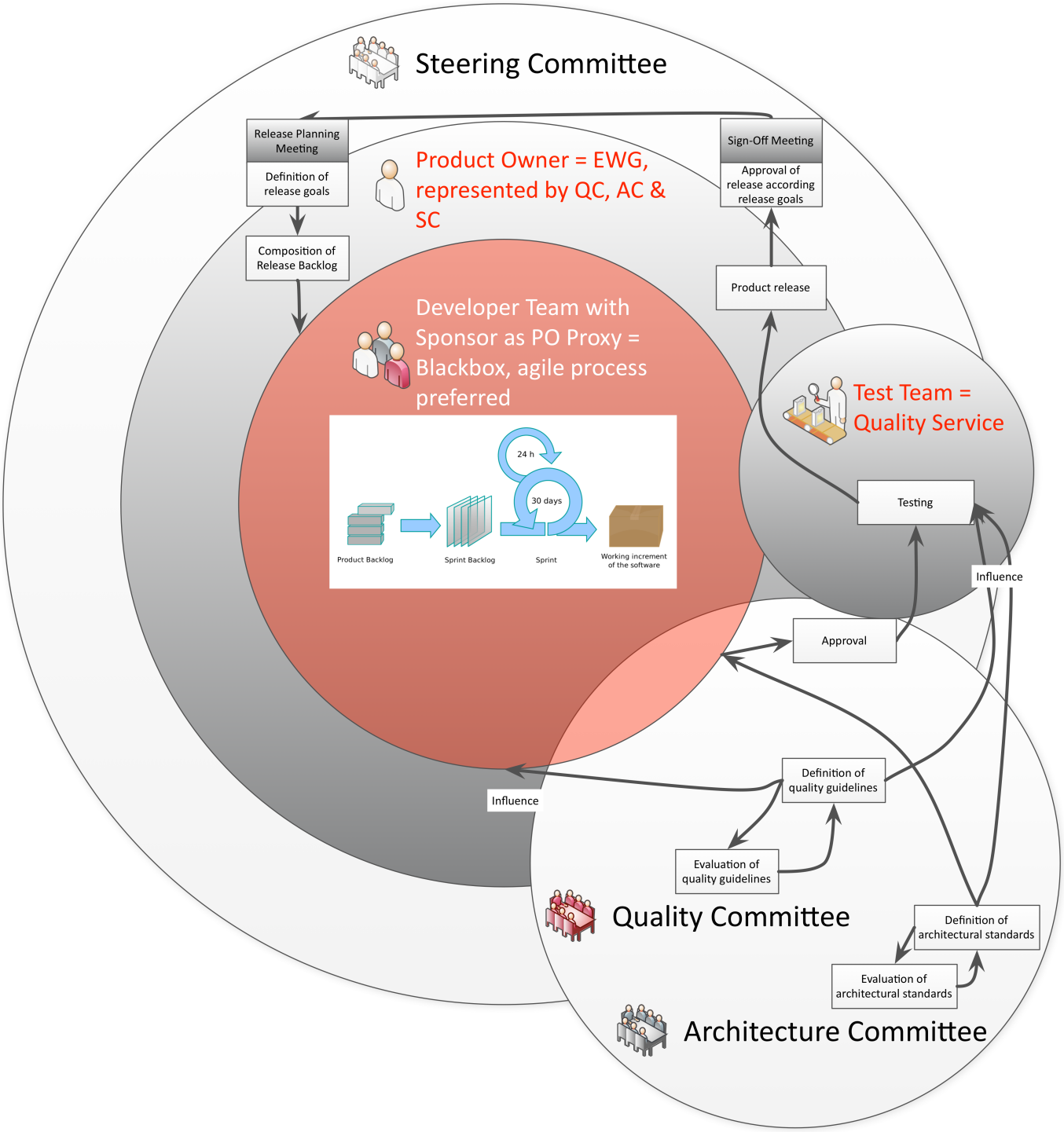
Strength: focused and lean product development possible - central roadmap and release planning help to lower time to market. High quality of comprehensive product (building blocks and example system) less overhead, rework because of short feedback/decisions cycles.	**Weakness:** full time job - increases cost for work group
Opportunities: shorter time to market of a comprehensive work product. This increases adoption of the product (aka Baukasten) of the automotive community, which then increases possible contributions (viral growth)	**Threats:** too much knowledge concentration at the product manager. product manager is not really independent.

5.2. Suggestion by Sven Wittig

5.2.1. Summary:

1. No dedicated, work group Product Owner, but product ownership is executed by whole EWG, represented by QC, AC and SC
2. The quality service execute the tasks of the test team
3. Black box of implementation: Service Provider takes the requirements (functional and non-functional which include quality and architecture guidelines), discuss everything internally with its sponsor and delivers the implemented results back to the workgroup. They are totally free which methodology to choose, the only constraint is that the issues are publicly available in the issue tracker
4. The acceptance of the delivery is decided by the sponsor and by the Quality Service (test team).
5. The requirements service is limited to collect and organize incoming requirements from the community, bundle them and present them to the work group.

The following picture should visualize the suggestion



Strength:	**Weakness:**
less coordination work which also means less costs for the work group only built what finds a sponsor	long time until decisions are made because of common product ownership lack of focus: danger of implementing similar components slightly different instead of focusing of providing the main components as building blocks

Opportunities: self-driving community emerging components which nobody thought of beforehand	**Threats**: lack of coherence of the individual building blocks (despite QA) individual interests of the sponsors are put above the interests of the work group. longer time to market prevents community adaption ("we rather build our own thing now")

5.3. Conclusion

Both models have their advantages and disadvantages. Because time to market seems essential for a success of OpenMDM 5 we therefore suggest to start with the proposed model and switch to Svens proposal when first deliverables are achieved and a certain momentum in the community has been gained.