# Results of Applying TracIMo to Introduce Traceability in a Mortgage Broker Company

Blinded for review

Blinded for review

## 1 Background

TracIMo is a newly defined traceability introduction methodology. It contains explicit steps on how to define a company- or project-specific traceability strategy, introduce the strategy, and evaluate the strategy. In this document, we describe how TracIMo was used to define and introduce a traceability strategy in an agile development team of a company in the finance domain. The company develops a website for mortgage applications that connects potential borrowers with money lenders.

## 2 Applying TracIMo at the Case Company

TracIMo consists of ten steps, which are split into two phases as shown in Figure 1. The first phase consists of steps of eliciting the traceability needs from the company and defining a suitable traceability strategy, while the second phase consists of steps for tool selection, deployment and evaluation of the traceability strategy. In the next subsections, we describe how each step was applied at the company and the results of the steps.

### 2.1 Step 1: Analyze existing software development process

The first step in TracIMo is to gather data on the development process of the company in order to understand the different activities and roles involved in the development as well as their goals. To collect this data, we conducted two interviews, one with the lead developer(LD) and one with the business analyst (BA). The interviews were conducted via Skype, since the researchers and the company were located in two different countries. We used an interview guide available as part of the supplemental material [1]. The interviews lasted around one hour each and were both recorded and later transcribed. We analyzed the transcribed interviews to identify the roles, goals, and tasks for these roles in the development process. This was done through thematic coding, where we had explicit codes for roles, tasks, and process goals. Form this information we derived a conceptual process model that shows all the development processes, the different roles and the different tasks associated with the roles. We also defined explicit process goals for each of the activities in the development process. The

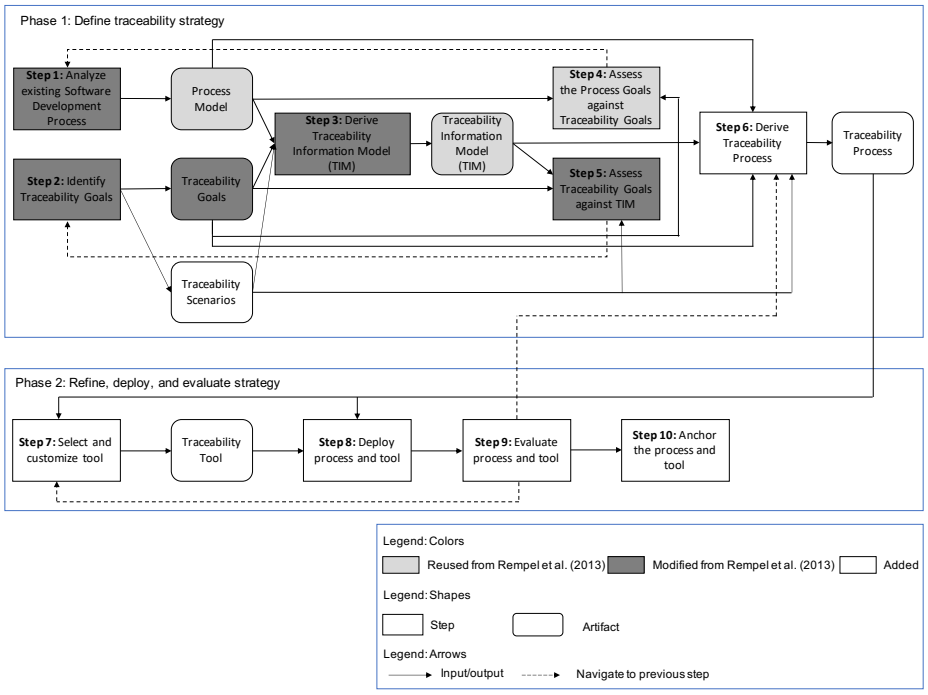


**Fig. 1.** The Traceability Introduction Methodology – TracIMo

conceptual process model is shown in Figure 2 and the process goals and the associated activities are shown in Table 1. We showed these results to the BA for member checking [2], to make sure that we correctly understood the development process at the company. In Section 2.1, we give a description of the current development process at the company in more detail. We also describe existing traceability practices in the company. Even through these practices are informal, they help us to understand which trace links the company already has and finds useful.

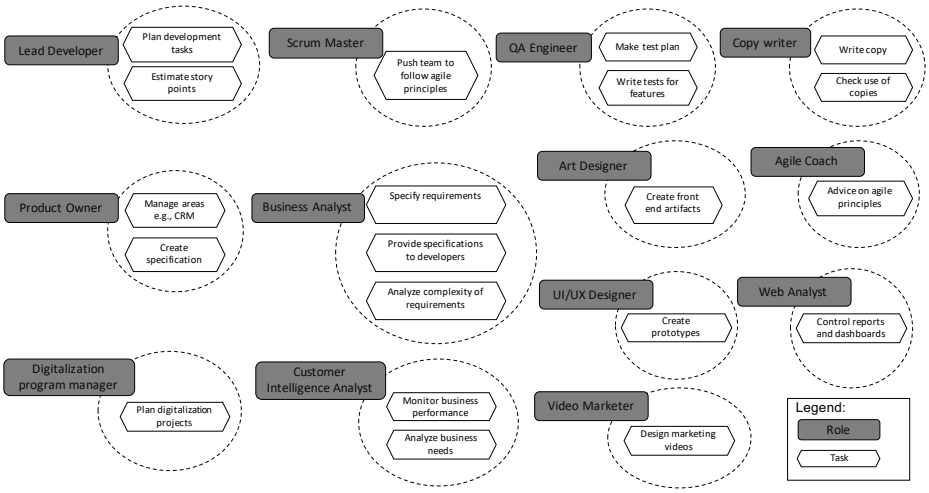


**Fig. 2.** Roles and tasks in the development process.

**Development process at the company** The company works in an agile way and uses Scrum as their agile method. As the scrum methodology instructs, the company has product owner, scrum master, developer and quality engineer (tester) roles in the team.

In the company, the development team is a horizontal group of developers distributed on different value teams. A value team is a group of people with different expertise (development, marketing, operations) that work together to achieve a defined goal. The company consists of four value teams that are depicted in the Figure 3. Each team has dedicated developers that will contribute to implement features to achieve the team's goal. In detail:

– **Operations Team** wants to increase the efficiency of the Operations department (advisors, mid-office, COO); their focus is the automation of advice and mid-office tasks and the improvement of the company's partnerships and commercial offerings.

- **SQL Team** – where SQL stands for Sales-Qualified Team – aims to increase the number of warm leads, which are the ones that will end up buying a house and, hence, closing a mortgage with the company; their focus is the communication area, from the landing pages that depict the company's services to the blog posts that explain different stages of the mortgage process and requirements.
- **Execution-Only Team** works to create the best execution-only customer journey, which differs from the advice-based customer journey by avoiding a direct conversation with advisors and could aim to narrowing the gap between customer and moneylenders; their focus is to enhance the customer journey by requiring only necessary information, to improve the underlying business logic, and to refine the document handling process using the national mortgage data exchange standard.
- **Analytics Team** focuses on analyzing the performance of the company, under every aspect – sales, website visitors, advices successfully closed, by extracting and processing useful data from Customer Module, Google Analytics, CRM, and other company-related resources.

Each team works autonomously and organizes itself in the agile Scrum fashion. Each of them has a Scrum Master that ensures that the Scrum principles are correctly applied and helps the team members to get over obstacles in that way. Each team has also its own Product Owner, depicted with a dark green box in Figure 3, who is responsible for defining the team direction by deciding what is the scope of each sprint, in order to achieve the defined goal. Given the reduced number of development resources, it might be possible that some of the development members can be assigned to tasks belonging to different teams.

Sprints last for two weeks and at the beginning of these two weeks a planning meeting is held to decide on which tasks need to be accomplished in the sprint and to assign the tasks to responsible developers. Every morning during the sprint, the team has a stand-up meeting. The sprint ends with a retrospective meeting to reflect on how the sprint went and find things that can be improved with the process.

Furthermore, at each sprint iteration, the Team Product Owners gather to coordinate the overall direction and to analyze which steps should taken further in the roadmap, by identifying high business value issues.

**Existing Traceability Practices** Before introducing traceability in the company, we analyzed existing traceability practices that were already in place. This is to ensure that we can leverage the existing links and understand how these links can be integrated in to the new solution.

Currently traceability is done on JIRA tickets. Bigger features are defined as epics in JIRA and within an epic, smaller tasks are defined. When the business analyst/product owner defines tasks tickets on JIRA, links to requirements, copy, wire frames, art designs, constants, code, other affected products and tests (not always) are created. These links can be made in the epic or if they are specific enough they can be added to the specific task. The aim is to make sure that the

| Current Practice | Process Goal |
|---|---|
| **Requirements Engineering**: The POs and the BA are responsible for the requirements engineering tasks which are to elicit requirements from the different value teams, document these requirements and make sure they are translated into actionable tasks. The requirements are written in Google Drive spreadsheets so that they can be easily shared. When requirements come from external entities, e.g., regulation boards, they are in PDF format. | **Process Goal 1**: Elicit all requirements from the product owner/BA's point of view.<br>**Process Goal 2**: Allow breakdown of all requirements into actionable tasks from a product owner/BA's point of view.<br>**Process Goal 3**: Improve the identification of related requirements from the product owner/BA's point of view. |
| **Software Design**: At the beginning of projects, the developers design a high-level overview either on the white board (stored as pictures) or in a UML modelling tool (stored in the corresponding format). | **Process Goal 4**: Improve the understanding of the software requirements from a developer's point of view.<br>**Process Goal 5**: Allow creation of a high-level design based on the requirements from a developer's point of view. |
| **Development**: The developers work on the tickets assigned to them and produce code. The code is written manually in PHP and stored in git repositories. | **Process Goal 6**: Allow implementing new features from a developer's point of view.<br>**Process Goal 7**: Allow implementing changes of existing features from a developer's point of view.<br>**Process Goal 8**: Improve the identification of artifacts that need to change from a developer's point of view.<br>**Process Goal 9**: Improve the understanding of the relationship between code and requirements from a developer's point of view.<br>**Process Goal 10**: Improve the planning process for future changes from a developer's point of view. |
| **Quality Assurance**: The developed feature is tested against its requirements to verify that it works correctly. The company has dedicated testers who write tests for implemented features. The tests are stored in git repositories together with the code they test. | **Process Goal 11**: Improve the understanding of requirements from a tester's point of view.<br>**Process Goal 12**: Allow verifying features from a tester's point of view.<br>**Process Goal 13**: Improve the understanding of which artifacts need to be tested after a change is made from a tester's point of view. |
| **Project Management**: This activity is associated with planning development and following up on the progress of development to make sure that features being developed align with the goals of the company. | **Process Goal 14**: Improve the understanding of software requirements from a PO/BA's point of view.<br>**Process Goal 15**: Improve effort estimation of requirements from a PO/BA's point of view.<br>**Process Goal 16**: Allow prioritizing requirements from a PO/BA's point of view.<br>**Process Goal 17**: Improve requirements' progress monitoring from a PO/BA's point of view. |

Table 1: The table shows the different activities in the development process and the process goals

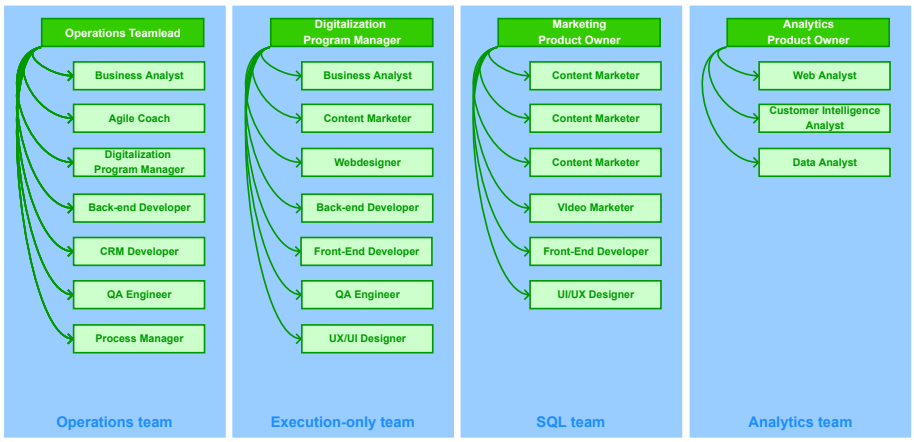


**Fig. 3.** Organization Structure

developer has enough information to carry out the task given. However, these links are transient as they are only valid for the current ticket/epic and for each task/epic that is defined the product owner/developer has to figure out which links should be established. The business analyst first identifies what needs to be changed, records this information in an excel file, creates tickets in JIRA from the excel file and add links to the tickets. If two tickets are related, a link is created between them by adding the ticket number of one ticket in the description of the other ticket.

To improve this current practice there is a need to have connections between requirements and other development artifacts e.g., models, implementation and code, and these connections should be reusable every time a change is requested.

### 2.2 Step 2: Identify traceability goals

We used the data collected in Step 1 to derive traceability goals for the company. Again we used thematic coding for the data analysis to identify these traceability goals. When all the goals were identified, we showed them to the BA for member checking. This led to updating of some of the goals. Our final results consists of seven traceability goals, which we formulated using GQM [4].

For each goals we derived questions and metrics that will be used to answer these questions. This was done in a workshop with the two researchers. After the initial list of metrics was complete, we had another workshop where the BA was available on Skype to discuss if the metrics makes sense and if the data for the metrics will be available from the company. We also decided when the metrics will be taken and that the BA will be responsible for taking the metrics. For each goal we also defined scenarios that we used later on to assess if the goals will be fulfilled. We present the goals, metrics and scenarios in Table 2 to Table 7.

The data sources to which access is required are listed in Table 8.

Table 2: Goal/Question/Metric to identify traceability goals and metrics

| | |
|---|---|
| **Goal 1:** | Increase the awareness of stakeholders about product changes from the business analyst point of view. |
| Rationale: | Different stakeholders are involved in the development of new features. Ideally, these stakeholders are able to shape the requirement as well as the implementation according to their needs. They also need to be aware of the schedule for the development of new features and give input for their prioritisation. Traceability can help to identify all the stakeholders concerned with a change early in the process by identifying which artifacts are connected to a change and from the artifacts identify which stakeholders (people who develop or use the artifact) should be involved. |
| Question 1: | How easy is it for relevant stakeholders to influence the changes? |
| Metrics: | – Number of comments on JIRA tickets from non-developers (analysis of JIRA tickets)<br>– Number of suggested changes to JIRA tickets by developers (analysis of JIRA tickets) |
| Question 2: | To what degree are relevant stakeholders aware of the changes? |
| Metrics: | – Likert Scale awareness:<br>1 – not aware at all;<br>5 – very aware<br>per task<br>(Questionnaire with BA) |
| Question 3: | How timely are announced changes deployed? |
| Metrics: | – Difference between time of announcement and time of deployment (analysis of JIRA tickets) |
| Scenario: | Given a requirement, it should be possible to identify corresponding development artifacts (design models, implementation, customer content, tests, copies, art designs and wire frames) and other artifacts such as change sets, by following trace links. By identifying the artifacts, one can also identify the stakeholders that use these artifacts. |

Table 3: Goal/Question/Metric to identify traceability goals and metrics

| | |
|---|---|
| **Goal 2:** | Improve the visibility of the decision rationale from the development team's perspective |

| | |
|---|---|
| Rationale: | The rationale behind a requirement is important information for the development team since it can help make decisions during design, implementation, and test. As such, the rationale should be visible and understandable to the team. If the team understood the task clearly, it should require less discussion and less rework. Traceability links can support this purpose by connecting the relevant artifacts describing the rationale of decisions to the requirement description. NOTE: For a few delicate projects (e.g., the ones that have to follow certain standards) rationales are recorded in a separate document |
| Question 1: | How understandable is the decision rationale? |
| Metrics: | – Likert scale understandability:<br>1 – not understandable at all<br>5 – very understandable<br>per task<br>(Questionnaire with developers)<br>– Numbers of traceability links per task (analysis of traceability model and JIRA tasks)<br>– Correlation between number of traceability links and the understandability of the rationale (derived)<br>– Number of re-opened tickets (analysis of JIRA tickets)<br>– Number of rejected tickets (analysis of JIRA tickets)<br>– Number of comments per task (analysis of JIRA tickets)<br>– Correlation between number of comments and understandability rating (derived) |
| Question 2: | To what degree is the decision rationale visible to the team? |
| Metrics: | – Likert scale visibility<br>1 – not visible at all<br>5 – very visible<br>per task<br>(Questionnaire with developers) |
| Scenario: | Given a ticket, it should be possible to create trace links from the ticket to the requirement(s) that is associated with the ticket. This is because developers can identify which requirement is related to the ticket and understand the rationale of the ticket. |

Table 4: Goal/Question/Metric to identify traceability goals and metrics

| | |
|---|---|
| **Goal 3:** | Improve the accuracy of effort estimations for tasks from the Lead Developer's point of view. |

Rationale: One of the main tasks for the lead developer is to estimate the effort a certain implementation task is going to have. This has a major influence on the sprint and on the schedule for the developers since it essentially determines how many tasks the team will tackle during a sprint and how much time they can devote to each task. Increasing the accuracy of the effort estimation is therefore a goal. Traceability links can support this goal by providing insight into dependencies between artifacts and requirements, and by helping to identify which parts of the code have to be touched for a change. Since an estimation can never be 100% accurate, an additional dimension is how confident the lead developer feels with his estimations. If traceability links do in fact support the estimation, the lead developer should become more confident in estimating over time and the high confidence estimation should become more accurate at the same time.

Question 1: How much does the estimated effort differ from the actual effort?

Metrics:
- Average number of tasks per sprint (analysis of Product Backlog/JIRA tickets)
- Average number of deviating tasks per spring (analysis of Product Backlog/JIRA tickets)
- Percentage of deviating tasks per sprint (derived)
- Initial estimation for each task in story points (analysis of Product Backlog/JIRA tickets)
- Updated estimation for each task in story points (analysis of Product Backlog/JIRA tickets)
- Average increase/decrease in effort per task (derived)
- Number of JIRA comments about effort per task (analysis of JIRA tickets)

Question 2: How confident is the lead developer in the estimation of tasks?

Metrics:
- Likert scale confidence
  1 – not confident at all
  5 – very confident
  per task
  (Questionnaire with lead developer)
- Number of low confidence tasks that required a change (analysis of Product Backlog/JIRA tickets)
- Number of high confidence tasks that required a change (analysis of Product Backlog/JIRA tickets)

Scenario: Given a ticket, it should be possible to create links to requirements, model elements, implementation and tests that are associated with the ticket. By identifying all artifacts that need to change using trace links, the LD can make better estimations of the tickets.

Table 5: Goal/Question/Metric to identify traceability goals and metrics

| | |
|---|---|
| **Goal 4:** | Increase the efficiency of identifying artifacts relevant to a change from the developers' point of view. |
| **Goal 5:** | Increase the efficiency of identifying artifacts relevant to a change from the business analyst's point of view. |
| Rationale: | These two goals are closely related and only differ in the viewpoint. Therefore, they are treated together and the questions and the metrics are valid for both.<br>At the moment, it is a significant effort to identify all artifacts that are affected by a change. The business analyst creates an initial change impact analysis, but that does not contain the code. The developers thus need to identify the relevant code parts themselves. Therefore, there is effort for both roles. If traceability links are established between all relevant artifacts, this effort should be significantly reduced and the information recorded for use in the future. |
| Question 1: | How well are the different artifacts connected to the code? |
| Metrics: | – Number of identified code sections per task (Analysis of traceability model and JIRA tickets)<br>– Number of tasks (analysis of JIRA tickets)<br>– Average number of identified code sections per task (derived)<br>– Number of missed sections (analysis of commits) |
| Question 2: | How easy is it to identify artifacts connected to a change for a given task? |
| Metrics: | – Likert scale difficulty of identifying artifacts connected to a task<br>1 – very difficult<br>5 – very easy<br>per task<br>(Questionnaire with developers), (Questionnaire with business analyst)<br>– Number of artifacts added after initial change impact analysis (analysis of traceability model and JIRA tickets) |
| Question 3: | How efficient is the creation of the change impact analysis? |
| Metrics: | – Initial estimation for each task in story points (analysis of Product Backlog/JIRA tickets)<br>– Updated estimation for each task in story points (analysis of Product Backlog/JIRA tickets)<br>– Average increase/decrease in effort per task (derived)<br>– Duration of creating a CIA (time sheet of BA)<br>– Effort of changing an existing CIA (time sheet of BA) |
| Scenario 1: | Given a ticket, the BA should be able to create trace links to all artifacts associated with the ticket, including the code. |

| | |
|---|---|
| Scenario 2: | Given a ticket, the developer should be able to identify all artifacts associated with the ticket using trace links created by the BA. |

Table 6: Goal/Question/Metric to identify traceability goals and metrics

| | |
|---|---|
| **Goal 6:** | Improve the visibility of the dependencies of the process steps from the lead developer's point of view |
| Rationale: | Hidden dependencies are a major source of added effort. If dependencies are not detected during the planning process, they can become evident when code is checked in (in the form of merge conflicts), when it is compiled (in the form of failed tests or builds), or during design. Traceability links can help identify dependencies and thus, a system with a high number of traceability links should be less susceptible to this kind of mistake. |
| Question 1: | How often do dependency conflicts manifest in development? |
| Metrics: | – Likert scale ease of dependency identification<br>1 – very difficult<br>5 – very easy<br>per task<br>(Questionnaire with lead developer)<br>– Number of merge conflicts (analysis of commit history, continuous integration server)<br>– Number of backed-out patches (analysis of commit history)<br>– Number of JIRA comments about dependency issues (analysis of JIRA tickets)<br>– Number of failed builds (build history on continuous integration server)<br>– Number of failed tests (build history on continuous integration server) |
| Scenario: | Given the trace links related to a requirement, it should be possible to identify which processes are still needed for the requirement to be completed. For example, if a requirement that requires copy and is not yet linked to copy, it means the copy does not exist and needs to be created. |

Table 7: Goal/Question/Metric to identify traceability goals and metrics

| | |
|---|---|
| **Goal 7:** | Improve the visibility of progress from the Product Owner's point of view |

| | |
|---|---|
| Rationale: | One of the stated goals, in particular of the lead developer, was to use traceability information to get an insight into the current progress of the development. The business analyst also emphasised the importance of having insight into the "entire lifecycle of a feature" and to get insights at all stages of the process. This kind of progress tracking is connected to traceability since traceability links allow, e.g., to identify if all relevant code segments have already been touched or to see if the relevant artifacts have already been updated. |
| Question 1: | How easy is it to gauge the progress of a task? |
| Metrics: | – Likert scale visibility of progress<br>1 – very opaque<br>5 – very visible<br>per task<br>(Questionnaire with Product Owner)<br>– Number of traceability links per task (analysis of traceability model and JIRA tickets)<br>– Correlation of visibility and number of links (derived) |
| Scenario: | Given a ticket, the PO should be able to identify all the artifacts that need to be changed for the ticket to be completed through trace links. The PO should then be able to investigate if these artifacts have already been changed to get an idea of the progress of the ticket before the sprint ends. |

### 2.3 Step 3: Derive Traceability Information Model

Based on a definition of the traceability goals and the process goal, we defined a traceability information model (TIM) to describe the links needed and the semantics of the traceability links. First the two researchers went through the traceability goals, to identify all links types that are needed and went through the development process to identify which traceable artifacts correspond to the needed traceability links. We created a model of these links using Xcore [], which is a textual language for specifying EMF models. Xcore also allows to have the graphical representation of the model. As suggested by TracIMo, having a graphical representation of the TIM, makes it easy to discuss with involved stakeholders. It also makes it easy to detect missing links and duplicate links. We iterated over the design of the TIM several times through emails and Skype calls with the BA. The TIM designed is represented in Figure 4.

| Source | Comment |
|---|---|
| JIRA tickets | The JIRA tickets represent the epics and tasks and contain all relevant information about the implementation of a change. In particular, they contain traceability links to, e.g., commits and requirement documents and will contain the traceability visualisations done in Eclipse Capra [1]. The comments are a particularly valuable source of information and will be used to identify what the developers discussed while working on the task. |
| Product backlog | If a product backlog that is separate from the JIRA tickets exists, this can provide information about the estimation and prioritisation of tasks. |
| Commit history | Commit messages usually contain a link to the task they were intended for. They will allow us to see which code was changed to address which task. |
| Questionnaires | Qualitative data about the satisfaction of the different stakeholders is going to be collected with questionnaires. |
| Build history on continuous integration server | The build history gives insight into failed builds that can be caused by compile errors, failed tests, or dependency issues. |
| Time Sheets | All activities where effort is recorded will need to be accompanied by a time sheet. Since we can not expect developers to record all times accurately, this data collection tool will be limited to the BA. |

Table 8: Data sources for metrics

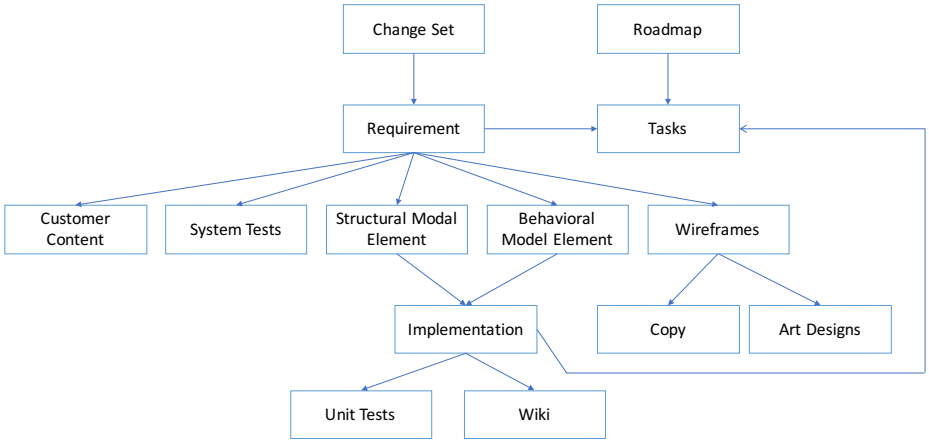


**Fig. 4.** Proposed Traceability Information Model.

### 2.4 Step 4: Assess the process goals against traceability goals

In this step, we used the traceability process goals defined in Step 1 and the traceability goals defined in Step 2 to analyze if all the defined process goals that require traceability to be fulfilled, have at least one corresponding traceability goal associated with them. The two researchers went through a list of all the process goals and identified traceability related goals. Then investigated the list of traceability goals that matched the process goals. The result of this step is shown in below.

### 2.5 Step 5: Assess traceability goals against TIM

In this section, the traceability information model defined in Step 3 will be evaluated in terms of the traceability goals that have been defined in Section 2.2. Whether these goals can be achieved or not depends on the expressiveness of the meta-model as well as the traceability practices that are put into place. In this section, we will only consider an evaluation of the meta-model, whereas the evaluation of the practices will be left to the empirical validation in the evaluation step (Step 8). Since we focus on the TIM, the object of the analysis are the artifacts that are connected via traceability links and the semantics of these links. In this step, we also used the scenarios defined in Step 2 in the assessment. The analysis here relates to the meta-model shown in Figure 4.

All links are undirected and establish 1:1 relationships between the artifacts. No additional meta-information is captured in the links.

*Support of goals* The following list contains the goals and how the final version of the traceability information model helps to achieve them:

**Traceability Goal 1:** Increase the awareness of stakeholders about product changes from the business analyst point of view.
The traceability links and in particular the change impact analysis that is enabled through their existence allows the business analyst to communicate product changes to the stakeholders and to give an indication which impact they have. By including the CIA in the tickets, all stakeholders that have access to those get an immediate impression of the elements that are affected by the change.

**Traceability Goal 2:** Improve the visibility of the decision rationale from the development team's perspective.
Links between requirements and tickets as well as between the roadmap and tickets indicate the rationale behind them.

**Traceability Goal 3:** Improve the accuracy of effort estimations for tasks from the Lead Developer's point of view.
Links between the requirements and the model elements allow identifying all aspects of the system that are affected by a change. The transitive links to the implementation indicate the code elements that need to be changed. This change impact analysis improves the overview and should support the Product Owner in estimating the task.

**Traceability Goal 4:** Increase the efficiency of identifying artifacts relevant to a change from the developers' point of view.
This goal can be achieved due to the same reasoning as for Goal 3.

**Traceability Goal 5:** Increase the efficiency of identifying artifacts relevant to a change from the business analyst's point of view.
This goal can be achieved due to the same reasoning as for Goal 3.

**Traceability Goal 6:** Improve the visibility of the dependencies of the process steps from the lead developer's point of view.
The process steps correspond to different activities that need to be performed by different stakeholders. For instance, copy needs to be provided before the web page can be programmed. The existence of a traceability link between a requirement and copy thus indicates that the step has been done.

**Traceability Goal 7:** Improve the visibility of progress from the Product Owner's point of view.
The traceability information model makes it easier to track progress since it clearly identifies the affected elements of the system that are affected by a change. When comparing with which elements have already been changed (e.g., tests, customer content, models) to which have to be changed, a notion of completeness can be derived. Notably, however, traceability does not help establishing to which degree the different elements have already been completed, just if they have been touched at all.

*Changes to the model due to evaluation* The first revision of the analysis revealed that the concepts related to requirements were not clearly delineated. Since product changes impact the requirements and must be broken down to be recorded in JIRA tickets (Goal 1), it is essential that these are traced correctly. The first version of the model contained the concept of a "task", however. This was ambiguous, since a task could be a user story or something more fine-grained. It was also ambiguous since the terms "ticket" and "task" were used interchangeably. For the purposes of traceability, however, the concept "ticket" is most suitable, since requirements (in the Excel document) will be linked to tickets in JIRA, no matter what level of abstraction they are.

This line of thought also revealed that traceability links that are maintained by Eclipse Capra need to be distinguished from those maintained in JIRA. For instance, wireframes, copy, and art designs are going to be added to the JIRA ticket as images or text, respectively. Therefore, the traceability link is going to be handled through the containment relationship in JIRA. Likewise, links between tickets (e.g., from an epic to user stories) are handled by JIRA. This has been made visible in the conceptual model.

Other than that, no obvious missing connections have been identified. It is not entirely clear if the links between tickets and implementation are necessary. Since the requirements are transitively linked to the implementation, it might not be helpful to have an additional link originating in the ticket. However, if commit messages are augmented with ticket numbers, this connection becomes relevant again and should be included. It can be helpful to determine progress (Goal 7).

The end result of this step is the TIM depicted in Figure 5.

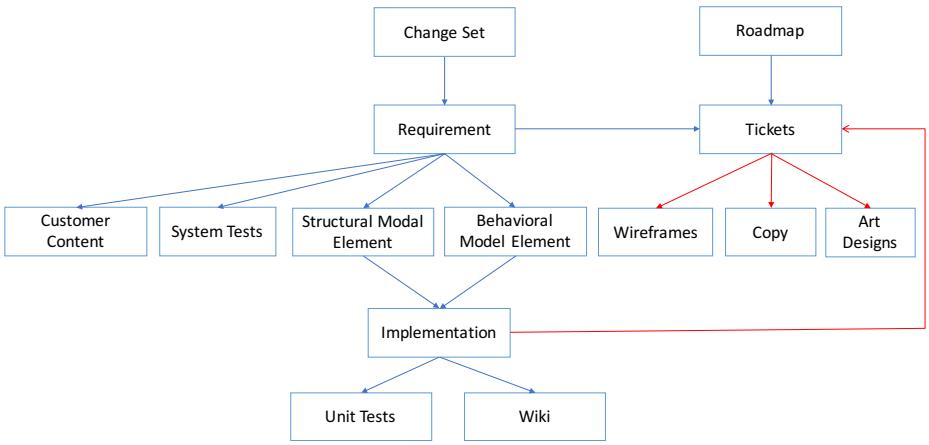


**Fig. 5.** Updated Traceability Information Model. Links in red not supported by Eclipse Capra but through JIRA. Tasks have been renamed to Tickets.

### 2.6 Step 6 : Derive a traceability process

In this step, we defined how trace links were going to be created, maintained and used. The inputs we considered for this step were the process model the company, the traceability goals, metrics, scenarios and the defined TIM. Since the BA was already responsible for conducting the manual impact analysis, we decided that he should also create the trace link because he knows the system well and was already creating links implicitly in the existing development process. The BA is also responsible for updating the links when artifacts evolve.

The end users of the trace links will be the development team, the lead developer, the product owner as well as the BA. Due to the fact that there were no existing links and the systems developed at the company already had a large number of artifacts, the links will be created in a retrospective manner. To reduce the load for the BA, the links will also be created incrementally. For each sprint, the BA will create links to tickets planned for the sprint and make these links available to the developers. This is a lightweight approach for creating links as the BA can focus the effort on the links that yield immediate benefits. Furthermore, links between development artifacts are also created incrementally, e.g., links between model elements and implementation and between implementation and tests. These links can be reused the next time a change involves an artifact that already has trace links.

We used the metrics and measurement plan defined in Step 2 to define a data collection strategy that should be included in the traceability process. The data from JIRA, e.g., average number of tickets per sprint did not need extra effort in collecting as they can be automatically obtained from the JIRA system. For the data that needed the Likert scale, it was agreed that the BA will collect this data from the rest of the involved stakeholders.

### 2.7 Step 7: Select and customize tool

From the defined traceability process in Step 7, we searched for a traceability tool that would support the process. We investigated the existing traceable artifacts and identified which tools used. The traceability management tool therefore needs to be able to support linking to and from artifacts in the specific formats provided by the different tools. We used the traceability tool characteristics defined in [3] as suggested by TracIMo and selected Eclipse Capra[2] as our traceability tool. While other alternatives e.g., Yakindu traceability[3] where available, the company had no budget for a new tool and we therefore had to select an open source tool that we could customize. Additionally, we had to select a stand alone tool, or a tool that could easily integrate into the existing development process without disrupting the existing process. This meant that complete life cycle management tools such as Rational DOORS [4] were out of the question. The disadvantage of selecting Eclipse Capra is that links had to be created out of JIRA, the main system that the developers use. However, it was possible to extract traceability graph and embed them in the JIRA tickets. While this was an extra task for the BA, it meant that the developers did not need to learn any new tool.

The existing Eclipse Capra implementation did not support links to and from PHP code or spreadsheets. But since Eclipse Capra is extendable, we customized it to add this functionality. We also customized the visualization to show directions in links and added filtering mechanisms to select which elements should be shown in the traceability graph. Lastly, we implemented the TIM defined in Figure 5, so that the company can create their custom trace links.

### 2.8 Step 8: Deploy process and tool

To deploy the designed process and tool, we first created a schedule together with the BA for when the deployment would take place. This schedule was communicated to the team before the arrival of the researchers. The detailed schedule for the one week the researchers were present at the company is given in Table 9. Part of the original schedule was updated on-site, to accommodate arising issues. On the first day, one researcher installed the tool on the BA's machine, explained how links were going to be created and the BA started to create links using one project that was selected. During the week, the researchers attended the daily stand-up meetings of the developers to understand more how the team works. In the stand-up meetings, the BA showed the created links to the team for feedback. The feedback from the team was taken into account by the researchers, and the tool and process was updated as required. One of the major update that we did was to change the TIM, so that links are created from tickets to model elements, instead of to requirements. This is because developers mostly interact with the tickets and not the requirements. The resulting TIM

[2] https://eclipse.org/capra
[3] `https://www.itemis.com/en/yakindu/traceability/`
[4] `https://www.doorsng.com`

is shown in Figure 6. Additionally, we changed the granularity of the links from tickets to model elements. At first the BA created links to specific methods and properties in the UML classes, but the developers thought that this was too detailed and made the traceability graph complex to understand. To solve this the links were created to the specific classes only.

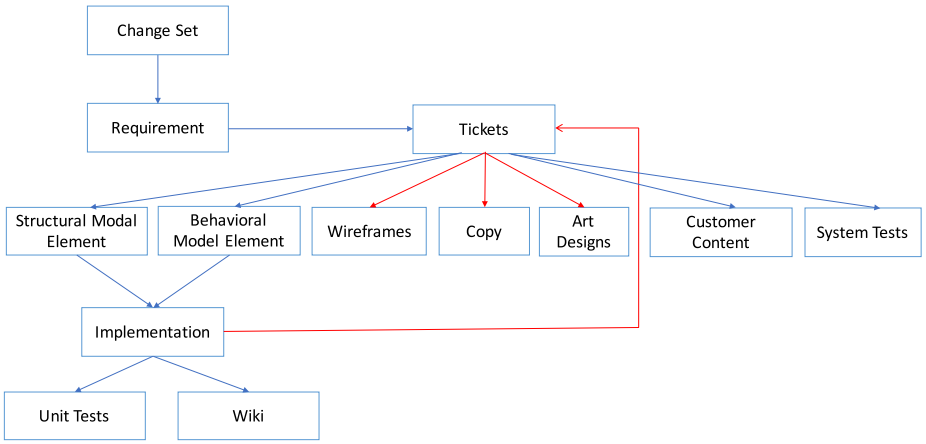


**Fig. 6.** Updated traceability information model. Links in red not supported by Eclipse Capra but through JIRA. Note that all development artifacts are now linked via the *Ticket*. The roadmap has been removed.

Table 9: Time line for the study at the company

| Activity | Day | Schedule | Persons |
|---|---|---|---|
| Stand-up meeting with Dev team | 1 | 9.45 to 10:00 | BA, SM |
| Update metamodel and fix issues | 1 | 10.15 to 11.15 | BA, BD, SM |
| Finalize tool environment | 1 | 11.15 to 12.00 | SM |
| Install Eclipse Capra for the business analyst and backend developer | 1 | 13.00 to 13.30 | BA, BD, SM |
| Configure and test Eclipse Capra (e.g., connection to Git, JIRA, Google Drive) | 1 | 13.30 to 14.30 | BA, BD, SM |
| Demo the tool for the business analyst and backend developer | 1 | 14.30 to 15.00 | BA, BD, SM |
| Create traceability links for a specific project | 1 | 15.00 to 17.00 | BA, BD, SM |
| Prepare presentation for QA Engineer, the Frontend Developer, and Product Owner | 1 | 17.00 to 18.00 | SM |
| Discuss and reflect on the day | 1 | 21.00 | SM, JS |

| | | | |
|---|---|---|---|
| Stand-up meeting with Dev team. Show the Dev team what the links will look like in the JIRA ticket (use this as a focus group to get feedback) | 2 | 9.45 to 10:00 | BA, SM |
| Meeting the the BD and BA to discuss the links created on day 1 | 2 | 10.15 to 11.00 | FD, PO, QA, SM |
| Interview with the PO about his experience with mainly task planning | 2 | 11.00 to 12.00 | BA, SM, PO |
| Update second researcher on current status and ongoing process | 2 | 13.00 to 14.00 | SM, JS |
| Revise traceability links and approach based on feedback and common action plan | 2 | 14.00 to 15.00 | BA, BD, SM |
| Discuss granularity and traceability goals with BA | 2 | 15.00 to 16.00 | SM, JS, BA |
| Create a UML model of the current implementation | 2 | 16.00 to 18.00 | JS |
| Business analyst continue creating traceability links and augmenting JIRA tickets | 2 & 3 | ongoing | BA, BD, (SM, JS) |
| Let the QA Engineer, the Frontend Developer, and the Product Owner use the links | 2, 3 & 5 | ongoing | FD, PO, QA, (SM, JS) |
| Discuss UML model of current PHP code | 3 | 9.30 to 9.45 | JS, BA |
| Stand-up meeting with Dev team | 3 | 9.45 to 10:00 | BA, JS, SM |
| Discuss feedback from focus group with business analyst and formulate action plan | 3 | 10.00 to 10.30 | JS, SM, BD, BA |
| Update the meta-model and package Eclipse Capra | 3 | 10.30 to 11.00 | SM |
| Write bug reports for new Eclipse Capra features | 3 | 10.30 to 11.00 | JS |
| Revise traceability links and approach based on feedback and common action plan | 3 | 11.00 to 12.00 | SM, JS |
| Get feedback from the Backend and Frontend Developer | 3 | 13.00 to 13.30 | FD, BD, SM, JS |
| Install updated Eclipse Capra on the BA's and the lead developer's machines | 3 | 13.30 to 14.00 | FD, BD, SM |
| Collect data to establish baseline for metrics | 3 | 14.00 to 16.30 | SM, JS |

| | | | |
|---|---|---|---|
| Discuss the new traceability information model with BA and ensure that graphs are added to tickets | 3 | 16.30 to 18.00 | SM, JS, BA |
| Stand-up meeting with Dev team | 5 | 9.45 to 10:00 | BA, JS, SM |
| Resolve any open issues | 5 | 10.00 to 12.00 | BA, BD, SM, JS |
| Agree on next steps and schedule follow-ups | 5 | 13.00 to 14.00 | BA, BD, SM, JS |
| Interview the Business Analyst and Backend Developer about their experience with the tool | 5 | 14.00 to 15.00 | BA, BD, SM, JS |
| Collect data to establish baseline for metrics | 5 | 15.00 to 16.30 | SM |

### 2.9 Step 9: Evaluate tool and process

The evaluation of the deployed tool and process started on the second day at the company. We collected feedback from the stand up meetings, when the developers were shown the links. In the week we spent at the company, we also conducted one focus group meeting with the BA, LD, and two back-end developers to get their feedback on the deployed process and tool. We also interviewed the PO. We used the post-deployment interview guide provided in the supplemental material [1]. The interview with the PO was recorded and transcribed, while in the focus group meeting, the researchers took notes. Additionally, the researchers took field notes that are included as appendix to this report (c.f. Appendix A – C).

After two weeks, when the first sprint where the developer used the links was over, we conducted another interview by Skype with the BA to understand how the links are used, their benefits and the challenges associated with the new traceability strategy.

We conducted two interviews again after five months. One with the BA and one with a back-end developer to understand the long term benefits of using the current traceability process and tool. At this point we also collected measurements from JIRA to understand how the current process has impacted the development. Some of the metrics we collected can be found in appendix D.

## A Appendix: Field Notes

### A.1 Assorted Notes Day 1:

- Introduction to members of the team
- Participated in standup meetings for 3 teams
- The meetings were short and clear. Members discussed what they will do that day, no reference to past tasks because it was the beginning of the sprint.
- Sat down with the BA to look at the metamodel and the following were suggested:
  - Change of task to tickets
  - Adding arrows to the diagram
  - Adding Class and Method distinction for PHP resources
  - Made links one to many instead of one to one
- Implemented the change
- Installed the tool to the BA's machine
- Connected to Jira, spent time importing tickets to Eclipse
- Connect to Google drive – Discovered a bug (see below)

Immediate enhancements in Eclipse Capra:

- Google drive should show all Excel sheets
- Google Drive first column missing
- Task ID instead of name in Jira

– Naming of links to many targets as From “source”

Future Enhancement for Eclipse Capra:

– Show connections from Selection view
– Many to many links
– Better editing of links (`https://bugs.eclipse.org/bugs/show_bug.cgi?id=506899`)

Creating links

– Discussion on what should and should not be linked.
– Long discussion on what the mapping Excel sheet means. The conclusion was that there are several systems that need to exchange data. The mapping sheet shows how the data fields in the different systems are related to each other. This can be supported by Eclipse Capra later on as a separate model containing links between two systems. The resulting graph can be added to the ticket as a mapping diagram (instead of the Excel sheet)
– First set of links to one requirement took around 30 mins, for about (7 links (not all one to one links)):
– Observations:
  • Locating what needs to be traced takes time
  • Deciding if something should be linked or not also takes time
  • Links were created in a perspective of what will need to change, if I change X. This leaves a question of if these links would have been the same if the tasks were different but involving similar elements. E.g, if the task was delete instead of modify – (We can test this in an experiment where we give two sets of instructions on how to create links (requirements-based Vs Task-based) and see if the links we get are different.)

– Planned for next day
– Got access to Jira account

### A.2 Assorted Notes Day 2:

Daily stand up meeting with Dev team

– Showed the metamodel and links created on day 1
– Their concern was on the granularity of the links and they though that it was too detailed
– One person asked if they will get more complex graphs in the end — This seemed interesting to him
– One developer asked if they could be able to see the graph of the entire system. This is currently not possible, only links from and to selected elements are shown

Meeting with the lead developer and BA

- How he can install the tool – zip and export did not work
- Needs to download and install Eclipse and all required plugins
- Discussion on which model to link to the requirements. There is a model from HDN (an external client dealing with compliance to money lenders), this contains the data fields that the company needs to have in order to comply with money lenders. However, this model is not a true reflection of the code base, the model contains come dutch words and some English words. The dutch fields are translated to English in the code. The code also contains additional fields that are not in the HDN model. So the agreement is to find a way to get a model that reflects the code instead of using the HDN code, however, I think that we can introduce a link type called model to model that will map the fields from HDN to the fields in the actual code base, this will make it easier for the BA to make the connection when creating the task, and (may be for the developers?)

Meeting with Product Owner

- Introduced him to what we are trying to achieve with the tool
- Asked about his role as a product owner
- Asked about his opinion on traceability, even though this was very hard because he does not deal with the code directly and also has only two weeks as a product owner
- He thinks having traceability will mainly help developers and give him an overview of how artifacts are connected, because this is currently hard
- Has some concerns on the amount of time it takes to create trace links

- Investigating what is wrong with Google Drive filtering the files shown to only recent files
- Investigating why task IDs do not show

Granularity

- Different levels will be needed
- This will also require having different visualizations to accommodate the different granularity levels

### A.3 Meeting with BA, FD (2), BD, and JS, SM, Day 3, 13.00

Deployment whenever the feature lands, not necessarily when the sprint ends. BA goes through changes that were decided yesterday:

- Granularity: different cases in which tracing went through properties instead of via classes. Granularity level will be decided by BA.
- Feedback BD: better maintainability with coarser granularity, detail was better in more fine-grained version; when building traceability diagram, it is difficult to be complete, so tests will still need to be run; therefore the effort of having something complete is thus not necessary.

- Question: how will you communicate about missing links or wrong links? Not clear yet, should be done in process with feedback to BA or BD. Could be via notes in the ticket.
- FD: Not all issues are covered by diagrams in the beginning. BA: focus on tasks in the backlog in the beginning
- Goal is to involve Kalkuli and Frontend part, new project will change the customer journey and affect the calculation engine, this will co-exist with the new advice flow; traceability will be helpful
- Changed metamodel to ensure more targeted view via the ticket to the developers. This seems to be positively received.
- Progress tracking: detailed the issues with it. BD refined goal to mean cost of changes (effort involved, places that need to be changed). This was also related to adding a weight to the elements in the graph. However, this can be accomplished by the graph visualisation and the number of connected elements. A form of highlighting would be interesting.
- Later on, seeing elements that are not linked will be helpful
- UI change to show parent
- BD: if it's too late in the project lifecycle, adding unit tests or something like that is too costly and it's no longer useful. Project is four years old, so the number of nodes is huge. BA: definition of scope is to address this issue, will also be used for new project
- BD: JIRA ticket is description with links to Google Drive etc. Is it possible to add the links to the artifact description so you can click the link and browser will open? BA: Graph will be in the ticket, requirement will be seen; SM: mapping for different systems is necessary, it is more static and does not change that often; therefore, a separate trace model will be used and the image can be added to the ticket.
- BD: How will the UML model be updated? BA: Current version will be manually and continuously updated. First UML, then implementation. JS: Who is going to update? BA: Me BD: Could also be the architect (himself)
- Name worksheets in Google documents to make the graph prettier.

### A.4 Granularity and Progress Tracking

Feedback from developers was that the traceability links are too fine-grained. One of the reasons for this level of granularity was Goal 7, since it is necessary to have links on a very low level in order to be able to track which parts of the implementation have already been added. (This is the only thing that could be tracked since changes in an existing method would not be visible this way.) However, the level of granularity means more work for the business analyst and is not necessarily helpful in tracking progress since the graph only shows what is already there and never what is missing (e.g., a class or method or unit test). However, effort estimation might be easier with a more fine-grained view (Goal 3).

Progress that the PO wants to check is how far the development has come. This can be checked via JIRA and the connection between the commit messages

and the ticket. However, even commits are not sufficient to do that since it is unclear how many commits are needed to complete a task. Additional information is coming through personal communication and the stand-up meetings. Information from the traceability model would not be helpful.

Granularity: if you link to a property, you have no idea which class to look in. Therefore, a higher granularity in the traceability model is useless to the developer and thus not helpful in achieving Goal 4. If granularity is set to the class level, some information is lost. Fix: include the parent in the display name for all EMF model elements.

Trade-off between BA effort in creating links for different purposes. BA wants to be able to model in both ways, either on the class level or on the property/method level.

### A.5 Semantics of the traceability information model

Semantics: ModelToImplementation is not really carrying any semantics about the concrete relationship of the elements that are connected by it. Introducing more fine-grained link types (implementedBy, usedBy, etc.) would make the semantics more explicit but also introduce more work and more potential for error.

The original version of the traceability information model (cf. Figure 4 used the requirement as the starting point for connecting development artifacts. That has the disadvantage, that, if a requirement is split over several tickets, the developers do not get the traceability information specific to the ticket, but information for all tickets related to that requirement. This is potentially confusing. Therefore, the relationships were changed so that the tickets link to the development artifacts.

### A.6 Working with UML Models of the Source Code

The class diagram is exported from PHPStorm and uses a non-standard format. We have no way to open that format right now in Eclipse, so it is not possible to create links directly to this. Also, the file format does not contain full information about the classes, but rather links to source code. It is therefore not particularly useful in any case.

Fix: reverse engineer the current source code and create a model (as well as diagrams from that). Done by using BOUML to create an XMI file, then removing the XMI header from that file and importing it in Papyrus. New diagrams can then be created there.

Since the models are re-engineered from the source code, the BA needs to introduce new elements that are necessary to fulfill a requirement. This is necessary to show the new elements in the traceability graphs, anyway. The company will thus make the models the gold-standard and show new elements in the model before it is implemented. A potential drawback of this approach is that the model and the source code might get out of sync.

### A.7 Tickets and Estimation

PO: Traceability can increase the understanding the PO has of the system. The PO has only been working in this role for two weeks and his technical understanding of the system is still limited. Therefore, the traceability graph is supporting him in understanding how the system works and which elements are connected.

The PO estimates tickets together with the BD and currently relies on the technical expertise of the BD. Tickets are usually over-estimated.

Tickets are created when the backlog is filled. BA can thus create links for the tickets in the backlog to help in the estimation.

### A.8 Eclipse Capra Features

Showing the parent of a model element in graph would allow identifying where it is located. If classes with identical names are located in different packages, it would also be helpful to see the parent. (Should be configurable)

Links between child elements should be liftable to the parent level (link between property in model and method in implementation should imply a link between the model and the implementation).

Support for several traceability models would enable traceability links between different types of models to be handled separately. There are models that would benefit from traceability, but they are a bit detached from the day-to-day development (HDN model in this case). Maintaining these links in the same traceability model as the rest is not desirable. A quick fix could be to have separate workspaces for these different purposes.

### A.9 Lessons Learned

We need to make sure to differentiate the artifacts that are introduced only for traceability and those that we introduce because they have practical advantages in other areas.

There are remote developers that are relying on the BA's analysis. We should interview them about their view as well.

Due to time constraints, we are pushing the interviews for the immediate benefits to the week after the introduction of the links to give everyone the chance to work with the links for a little while.

Company has recently been acquired by another company. This was the driver for the change of process and is now driving changes in the architecture as well. The impact on the use of traceability is unclear.

### A.10 Mapping of Enumerations in different data exchange standards

There is a need to map constants in enumeration between different models. There are data exchange formats that have enums that need to be mapped to

the ones used internally. Since there are several formats and each format has its own nomenclature and a different set of entries, the mapping of the internal models needs to be defined for each. At the moment, the mapping is done in a spreadsheet that is linked in tasks where this is relevant.

The question whether Eclipse Capra can be used for this purpose was discussed. We agreed that this might not be the most prudent thing to do since this information is relatively static and the visualisations Eclipse Capra offers are not particularly suitable for this purpose and it is difficult to change these links once they have been established.

### A.11 Release Engineering

Once the developer is done with a ticket, they send a pull request. If this is accepted, it goes to the testing system. The acceptance shows up in the ticket. Once testing is complete, the feature is released via a separate pull request (marked with "release"). This does not show up in the ticket, but the developer can see it on Github.

### A.12 Prioritisation of Traceability Goals

Based on the discussions at the company and the work with the traceability information model, a priority of the goals has been established as follows.

**Traceability Goal 4:** Increase the efficiency of identifying artifacts relevant to a change from the developers' point of view.

**Traceability Goal 5:** Increase the efficiency of identifying artifacts relevant to a change from the business analyst's point of view.
Goals 4 and 5 are directly achievable by introducing traceability links and generate the most benefit for the development team during the development.

**Traceability Goal 3:** Improve the accuracy of effort estimations for tasks from the Lead Developer's point of view.
Since the estimations so far were mostly relying on the expertise of the developers, this goal is also

**Traceability Goal 6:** Improve the visibility of the dependencies of the process steps from the lead developer's point of view.
This goal is closely related to goals 3 to 5. Since process steps are signified through different artifacts, making all relevant artifacts visible can be helpful.

**Traceability Goal 2:** Improve the visibility of the decision rationale from the development team's perspective.
As long as the decision rationale is captured in a linked artifact (e.g., as a separate document, the requirement or the roadmap), the impact of introducing Eclipse Capra is unclear since these artifacts were already linked through the JIRA tickets. Traceability Graph could give information about where to look in the file with the rationale. Also, Eclipse Capra can help in checking if each requirement has a rationale.

**Traceability Goal 1:** Increase the awareness of stakeholders about product changes from the business analyst point of view.
The original intent of the goal was to identify the stakeholders that need to be involved in the change. This is partially achievable through traceability links since different types of artifacts can be assigned to different stakeholders. Thus, if all relevant artifacts are identified, then the stakeholders can be identified as well. However, this was already done in the task breakdown in JIRA. This requires, however, to find all relevant artifacts in the CIA. Since all relevant stakeholders are now part of one team, all stakeholders are involved in the estimation and the breakdown.

**Traceability Goal 7:** Improve the visibility of progress from the Product Owner's point of view.
This is tightly coupled to the discussion of granularity.

### A.13 Feedback from Stand-up Meeting on Day 5

The fact that there is no test that is directly associated to a class does not mean that the class (or method) is not tested. There's a chance that there are indirect tests (tests of another class that also test the method in question) or special tests (e.g., for the Twig templates), that are stored elsewhere. The developers use a code coverage report to identify whether the tests are complete.

However, it was agreed that a missing link to a test can be an indicator for the developer to double check the presence of tests. It was unclear if links to indirect tests should all be recorded since this might be redundant information to the coverage report. This should be followed up in the interviews.

### A.14 Feedback from Summary Session

Goal 3 is relevant for many stakeholders (PO, BA, Developer) since everybody is affected by the estimation. Quality of estimation depends on the quality of the traceability graph. Feedback from developers is necessary to keep the links consistent and complete (e.g., if test is not in Class where it is expected).

Goal 6 is still relevant, copy for a feature could not be available or not final. That means that the feature can not be merged. Linking the ticket to the copy artifact will help in that. However, this should happen in JIRA. Copywriters have own template that is linked to the ticket. Unfortunately, this process is not yet optimal.

Estimation of a feature is about the complexity of the feature.

Since all tasks have already been planned and estimated, it does not make sense to introduce traceability graphs right now. Therefore it would be better to gauge this after the next sprint, where we also can gauge the impact on planning and estimation. We will instead get feedback from the BA after the next planning poker session.

Next time, we should go through all the traceability goals with the stakeholders and verify them before the study.

# B Appendix: Notes on follow-up meeting with the BA, May 31

- Link creation going well
- All tickets have graph now
- Some additional links that are not related to current tasks are being created now
- Helped in estimating one task already
  - There was a task in which some of the logic had to be copied to another entity across two files
  - Specified with the links the correct version
  - Links represented the change that needs to be done
  - External developers saw what needed to be changed: it was clear enough what the complexity was and understood clearly what needed to be changed by following the links
  - The task was easy/intuitive, though
  - Diagram had a total of four links (AU-332.png)
  - Developer asked if only following from highlighted task was sufficient
  - Developer is a freelancer, so he will have to find some time
- Issue with new Eclipse Capra version, name of links seem to be different or artifacts can not be resolved any longer
- Created links for tickets from this sprint and something for the new sprint
- Next sprint will be focused on some content tasks for EXO team, focus groups and changes on the frontend and presentation side
- There is one backend tickets for HDN, but will most likely be scheduled for the sprint after that
- Not sure if traceability links are helpful for the frontend, right now the BA works mostly with backend issues. Sees potential for the presentation itself, though, particularly for validation and flow logic because for that one can refer to methods and classes
- For next sprint, there is a task to remove a controller function and some components on the frontend side. However, it is only possible to link to a template file, instead of a method or something more concrete.
- No plans to work on that yet, will rather be focused on HDN and calculation engine; reason is that these changes to the business logic will be very difficult and impact a lot of business decisions in the next two or three months; would thus prefer to maintain the scope on that; there are also some non-negotiable rules from the regulator, and Eclipse Capra will give a good hint of these rules for the calculation are complied to. Adding frontend might add overload in terms of tasks and things to manage;
- Compliance was not an original goal; has connection to frontend since certain panels show certain calculations that are mandated by the regulator
- Eclipse Capra can give information about where the compliance is respected; this is a form of requirement; having this information would be great

– Models in the presentation logics are unclear, where to point the links? Could be fixed by extending the meta-model with a Ticket-to-Implementation link; might be a bad idea since that would mean that some people will bypass the model because that what they care about; that might impact the big picture.
– So far, tasks were not so complex, but what they had to cope with worked.

# C Appendix: Metrics

| **Before traceability** | | **After traceability** | |
|---|---|---|---|
| **Sprint name** | **Deviating tickets** | **Sprint name** | **Deviating tickets** |
| Jumbotroll | 5 | Sprint 1 | 4 |
| Ocean's 11 | 2 | OPS: Never ending football | 3 |
| OPS: Stock syndrome | 1 | EXO: Intervention | 3 |
| OPS: Never ending football | 1 | EXO: Judgment day | 3 |
| Revenge of the sixth | 2 | Sprint 4 | 2 |
| Alternative facts | 3 | EXO: Buzzy stuff | 1 |
| Alternative reality | 2 | EXO: Firewalls | 1 |
| King kong returns | 4 | OPS: Summer time | 1 |
| Fool's day | 2 | | |
| Sprint 4 | 1 | | |
| Paana cotta | 5 | | |
| Crazy bamboo | 1 | | |

Table 10: Number of deviating tasks per sprint before and after the introduction of traceability

| **Roles** | **No. of comments before traceability** | **No. of comments after traceability** |
|---|---|---|
| Jenkins | 67 | 3 |
| Developer | 18 | 8 |
| Non Developer | 59 | 2 |
| Total comments | 144 | 13 |

Table 11: Number of comments before and after the introduction of traceability

| **Ticket ID** | **Trace links** |
|---|---|
| OPS-174 | 4 |
| OPS-160 | 4 |
| MYV-80 | 6 |
| MA-2967 | 5 |
| MA-2950 | 3 |
| MA-2947 | 3 |
| MA-2941 | 2 |
| MA-2924 | 3 |
| MA-2923 | 3 |
| MA-2919 | 10 |
| MA-2916 | 2 |
| MA-2915 | 5 |
| DH-346 | 6 |
| DH-345 | 2 |
| DH-343 | 2 |
| DH-341 | 3 |
| DH-330 | 3 |
| CON-1021 | 3 |
| CNV-149 | 3 |
| CNV-65 | 4 |
| AU-334 | 3 |
| AU-332 | 5 |
| Total trace links | 84 |

Table 12: Number of trace links created for different tickets

## References

1. Authors, A.: Supplemental Information for "TracIMo: A Traceability Introduction Methodology and its Evaluation in an Agile Development Team" (2020), https://doi.org/10.5281/zenodo.4160569
2. Seaman, C.B.: Qualitative methods in empirical studies of software engineering. IEEE Transactions on software engineering 25(4), 557–572 (1999)
3. Steghöfer, J.P.: Software traceability tools: Overview and categorisation. In: Report of the GI Working Group "Traceability/Evolution", pp. 2–7. German Informatics Society (GI) (October 2017), http://pi.informatik.uni-siegen.de/gi/stt/38_1/01_Fachgruppenberichte/ARC_AKTE/ARC_AKTE_2017_p2_steghoefer.pdf
4. Van Solingen, R., Basili, V., Caldiera, G., Rombach, H.D.: Goal question metric (gqm) approach. Encyclopedia of software engineering (2002)