Release Planning with Feature Trees: Industrial Case

Samuel Fricker[1] and Susanne Schumacher[2]

[1] Blekinge Institute of Technology, School of Computing
Campus Gräsvik, 371 79 Karlskrona, Sweden
samuel.fricker@bth.se
[2] Zurich University of the Arts
Ausstellungsstrasse 60, 8005 Zurich, Switzerland
susanne.schumacher@zhdk.ch

Abstract. [Context and motivation] Requirements catalogues for software release planning are often not complete and homogeneous. Current release planning approaches, however, assume such commitment to detail – at least implicitly. [Question/problem] We evaluate how to relax these expectations, while at the same time reducing release planning effort and increasing decision-making flexibility. [Principal ideas/results] Feature trees capture AND, OR, and REQUIRES relationships between requirements. Such requirements structuring can be used to hide incompleteness and to support abstraction. [Contribution] The paper describes how to utilize feature trees for planning the releases of an evolving software solution and evaluates the effects of the approach on effort, decision-making, and trust with an industrial case.

Keywords: features, abstraction, release planning, roadmapping, case study.

1 Introduction

Software releases are planned by allocating requirements to development projects [1]. A strategic release plan aligns the development of an evolving software solution with market and stakeholder needs, company objectives, and constraints such as time and resources. Release planning is a central concern in iterative development, where multiple iterations, rather than a single project, are defined [2].

Release planning involves the following steps [3]. Requirements are elicited and specified. Criteria [4] are defined to evaluate and prioritize requirements [5]. Releases are then scoped by allocating the prioritized requirements to development projects. The resulting release plans are implemented, delivered, and analyzed with post-release reflections [6].

Requirements that enter release planning are often of low quality [7]. Their homogeneity [8], completeness, and understanding [9] are hard to ensure due to the limited effort invested before a development project is funded. This situation contradicts with the assumptions of release planning approaches that scope projects simply by prioritizing and allocating available requirements. Consequently, the results are not trusted and not used for guiding ensuing development steps [10].

This paper describes in detail how to hide the requirements-related problems by structuring the release planning inputs. The approach, whose initial ideas were introduced in an earlier position paper [11], is based on variability modeling [12] that allows abstracting from requirements with AND, OR, and REQUIRES relationships [13]. Variability is here used to structure decision options [14] for product evolution. This paper then introduces an industrial case [15] to understand how to use variability modeling in a real-world context of continuous agile product management [16]. Evaluated were feasibility of the approach and its effects on effort, decision-making, and trust were evaluated.

The paper is structured as follows. Section 2 describes background and motivation. Section 3 introduces variability-based release planning. Section 4 describes, analyzes, and interprets the industrial case. Section 5 discusses and concludes.

2 Background and Motivation

Release planning for software products is a key practice of software product management [17]. Software releases are planned to answer a stream of requirements that approach the product development organization [18]. The requirements are first homogenized [8] and pass triage [19] before they enter release planning [3]. Release planning then involves evaluation and selection of requirements to scope development projects [4]. The requirements that are closest to implementation are those that are detailed most [16].

Current release planning approaches fit well into this context of continuous requirements inflow. They require a complete catalogue of comparable requirements that are evaluated, prioritized, and selected for implementation [20]. Known prioritization approaches include manual techniques such as top ten, numerical assignment, ranking, and 100$-test [5], and computer-based techniques such as Integer Linear Programming [21, 22] and the Analytical Hierarchy Process [23].

Prioritization allows evaluating requirements in a controlled way and leads to requirements ordering that suits development projects [10]. However, scalability is limited; and the results are mistrusted and perceived inadequate to guide how to act [10]. Post-release reflections help improving decision-making over time [6].

We investigated release planning in an organization that developed innovative software as a service for managing media such as text, sound, pictures, and movies. The solution provided first-of-its-kind features, was in an early stage of its evolution, and had a small, but rapidly growing user base.

Responsible for the development was a product manager, a project manager, and a team of up to five developers. They reported to a company-internal steering committee with management of the development organization, of the product-owning organization, and of departments that used the solution. A product reference team was used to coordinated development with important stakeholder groups.

Surprisingly, there was no stream of requirements that the product organization was confronted with. No homogenization and triage of incoming requirements was necessary. Instead the requirements were based on ideas that originated from the product manager who was an expert in the product's application domain and on feedback from pilot users. Ideas were made explicit during product planning and specified in detail when communicating with the development team.

The requirements catalogue was managed in a word processor document and used as a basis for release planning. It contained 108 requirements. The requirements were grouped into 12 sections and 19 subsections or themes. In average, a group contained 3.6 requirements and was allocated to 1.93 releases. The grouping, however, did not show a relationship with requirements allocation to development releases.

The requirements were not prepared and analyzed in a form that was expected by current release planning approaches. A key concern to the practitioners was development efficiency. Effort was only put into requirements when the return of such an investment was obvious.

Requirements that were not likely to be implemented in near future were not specified. Some requirements were specified with descriptions of up to 245 words, others only with a few words in a declarative manner, again others were completely omitted because not relevant within a practical planning horizon. Many requirements were discovered while development progressed.

Requirements were not evaluated. Isolating a requirement from its context would have increased the risk of misunderstandings. For example, the requirement *thumbnails of variable sizes* would have carried the following ambiguities: *When would thumbnails be shown? For what purpose? Which sizes? What (photos, videos, documents, etc.) would be depicted by these thumbnails?* The many potential interpretations of such a requirement would have led to different interpretation of importance, dependencies, implementation cost, and risk.

Requirements were not prioritized. The product organization avoided to compare requirements. For example, questions like "is the requirement *thumbnails of variable sizes* more important than the requirement *storage of search results*?" have not been posed. Such comparison would have led to detailed evaluation results. However, details irrelevant at the given product evolution stage would have been sub-optimized.

The organization wanted to transition from implementing the whole solution with a single large project to incrementally evolving the solution with short development iterations. They considered improvements in their release planning capabilities as a key enabler and asked *how release planning can be implemented by abstracting from the detailed requirements and by focusing on the key product evolution decisions*. The desired approach had to support decision-making, maintain flexibility of how the solution evolves, and keep effort to be invested at a low level.

3 Feature Trees for Release Planning

The lacking stream of requirements and the tendency of not specifying and evaluating individual requirements motivated us to identify alternatives to current release planning approaches. The alternative had to fit the described organization with the innovative product and the strong leadership of the product manager. Release planning should remain a low-effort activity, however with improved decision-making support and flexibility.

Feature diagrams are a widespread approach to document and analyze variability of software products [12]. They are used to specify how features vary for the products of a product line (variability in space). Applied to release planning, variability models can be used for defining the evolution of software (variability in time) [24]. How feature trees are utilized for release planning, has been proposed in this line of research for the first time [1, 11].

We use AND, OR, and REQUIRE dependencies [13] to structure a solution's requirements as a feature tree. Figure 1 illustrates the feature tree of a solution, *Online Shop Sales*. A feature is a named group of requirements that are implemented in the same development increment (AND dependencies). E.g. the *Sales* feature in Figure 1 refers to six such requirements. To enable acceptable implementation of the feature, the feature's requirements are elicited [25] and refined until they comply with the solution's environment and design [26].

Sub-features extend a feature. They can only be implemented after their super-feature has been implemented (REQUIRES dependency). E.g. *Enhanced Cart Display* is such a sub-feature to the super-feature *Sales*. A chain of REQUIRES dependencies that connects the root with a leaf is called a feature vector [27]. Such a vector captures the foreseen levels of implementing a functional or non-functional concern of the software solution. E.g. the *OnlineShop Sales* solution may support just *Sales* or support both *Sales* and *Enhanced Cart Display*.

The implementation order of a feature's sub-features is not constrained a-priori (OR dependency). E.g. the root's eight sub-features can be implemented in any order.

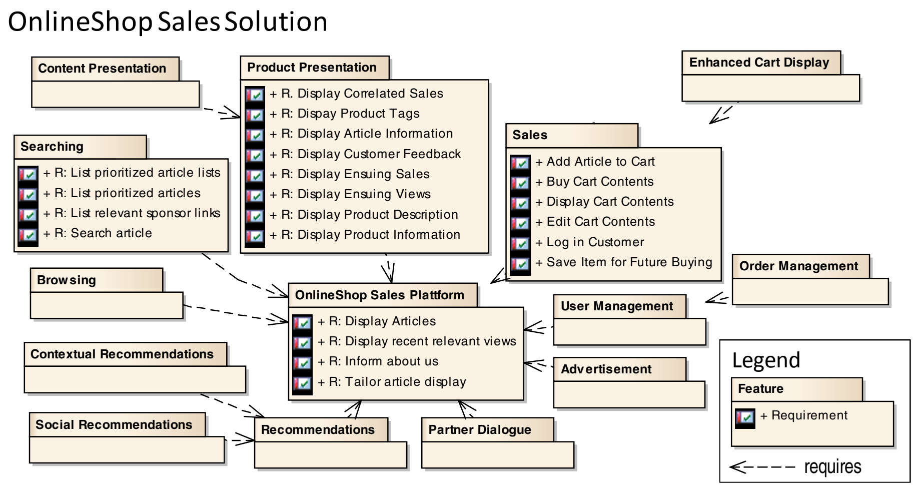
OnlineShop Sales Solution
Content Presentation
Product Presentation
+ R. Display Correlated Sales
+ R: Dispay Product Tags
+ R: Display Article Information
+ R: Display Customer Feedback
+ R: Display Ensuing Sales
+ R: Display Ensuing Views
+ R: Display Product Description
+ R: Display Product Information
Enhanced Cart Display
Searching
+ R: List prioritized article lists
+ R: List prioritized articles
+ R: List relevant sponsor links
+ R: Search article
Sales
+ Add Article to Cart
+ Buy Cart Contents
+ Display Cart Contents
+ Edit Cart Contents
+ Log in Customer
+ Save Item for Future Buying
Order Management
Browsing
OnlineShop Sales Plattform
+ R: Display Articles
+ R: Display recent relevant views
+ R: Inform about us
+ R: Tailor article display
User Management
Contextual Recommendations
Advertisement
Social Recommendations
Recommendations
Partner Dialogue
Legend
Feature
+ Requirement
requires

Fig. 1. Example of requirements structuring with a feature tree. The tree's root is *OnlineShop Sales Platform* in the middle of the diagram.

Figure 2 shows how we construct a feature tree, starting at the root. Initially, requirements and constraints related to architecture and infrastructure of the solution are allocated to the root. Then, feature vectors are built iteratively. For each feature, relevant requirements are identified and allocated to that feature. Feature-extending sub-features are identified and related to that feature. Requirements whose implementation can be postponed are extracted from the feature into these extending sub-features [28]. The requirements extraction process stops when no requirement can be extracted without making the concerned feature useless.

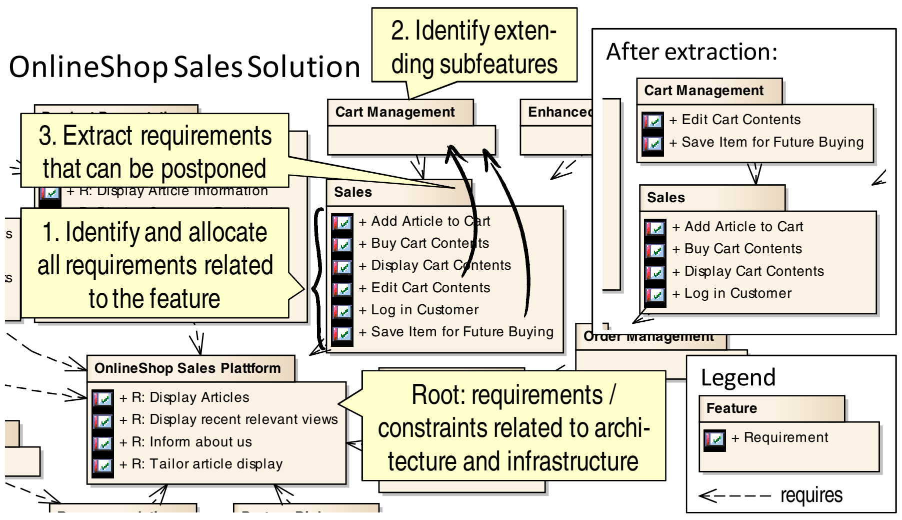
OnlineShop Sales Solution
2. Identify extending subfeatures
3. Extract requirements that can be postponed
1. Identify and allocate all requirements related to the feature
Root: requirements / constraints related to architecture and infrastructure
Cart Management
Enhanced
Sales
+ Add Article to Cart
+ Buy Cart Contents
+ Display Cart Contents
+ Edit Cart Contents
+ Log in Customer
+ Save Item for Future Buying
+ R: Display Article Information
OnlineShop Sales Plattform
+ R: Display Articles
+ R: Display recent relevant views
+ R: Inform about us
+ R: Tailor article display
Order Management
After extraction:
Cart Management
+ Edit Cart Contents
+ Save Item for Future Buying
Sales
+ Add Article to Cart
+ Buy Cart Contents
+ Display Cart Contents
+ Log in Customer
Legend
Feature
+ Requirement
requires

Fig. 2. Iterative feature tree construction process: repeat steps 1 to 3 for each feature until that feature contains just the minimal set of requirements to be useful. Progress from root to leafs

Figure 3 shows how we use the feature tree to document implementation progress and to visualize options for evolving the software solution. Initial development starts with the root. Features are implemented by following the REQUIRES dependencies. Implementation progress is documented by tagging features as being implemented, for example with a color code. Candidates for implementation are the features connected with already implemented or already planned features (connectivity rule).

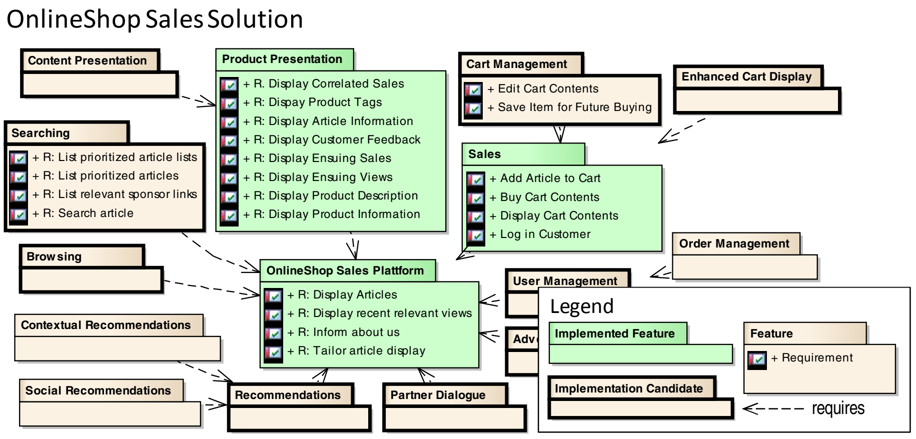
OnlineShop Sales Solution
Content Presentation
Product Presentation
+ R. Display Correlated Sales
+ R: Dispay Product Tags
+ R: Display Article Information
+ R: Display Customer Feedback
+ R: Display Ensuing Sales
+ R: Display Ensuing Views
+ R: Display Product Description
+ R: Display Product Information
Cart Management
+ Edit Cart Contents
+ Save Item for Future Buying
Enhanced Cart Display
Searching
+ R: List prioritized article lists
+ R: List prioritized articles
+ R: List relevant sponsor links
+ R: Search article
Sales
+ Add Article to Cart
+ Buy Cart Contents
+ Display Cart Contents
+ Log in Customer
Order Management
Browsing
OnlineShop Sales Plattform
+ R: Display Articles
+ R: Display recent relevant views
+ R: Inform about us
+ R: Tailor article display
User Management
Contextual Recommendations
Social Recommendations
Recommendations
Partner Dialogue
Legend
Implemented Feature
Implementation Candidate
Feature
+ Requirement
requires

Fig. 3. Progress tracking and visualization of options for software evolution

A feature tree simplifies the handling of a requirements specification in a release planning context. Features abstract from detail by grouping AND-related requirements. Allocating features instead of requirements to software releases reduces the number of release planning decisions. A feature tree hides incompleteness by handling non-specified features the same way as specified ones. Figure 1 shows ten features that can be used for feature-level release planning, even-though they do not contain requirements yet. Feature trees with information about development progress can be used to focus requirements analysis. Implementation candidates need to be of higher quality than other features.

A feature tree also captures requirements changes. Emerging requirements, e.g. discovered during elicitation or development, are added based on the product manager's judgment to existing non-implemented features or as new leaf features to the tree. Urgent changes are introduced as changes to active features according to a release project's change management process. Changes to already implemented features are introduced as part of the solution's maintenance process. The allocation of changes to features increases transparency for root-cause analysis and subsequent process and competence improvements.

4 Industrial Case Study

4.1 Study Definition, Planning, and Operation

Study Definition. Case study research was used to evaluate feature trees for release planning and to compare the approach with the backlog-oriented practice of using a flat list of requirements. The study aimed at understanding feasibility and impact of the approach in a real-world practical context from the perspective of the product manager responsible for release planning.

We asked the following research questions. RQ1: *How are feature trees used for planning software releases?* RQ2: *How do feature trees affect effort, decisions-making, and trust*? RQ1 focuses on the documentation of product features and the use of that documentation. It provides a rich picture of variability-based release planning and the context in which it is used. RQ2 describes the effects of the approach. It reports lessons-learned from the practitioner that has performed variability-based release planning. The answers help implementing the practice and deciding when to adopt the approach.

Case study research is adequate when *how* or *why* questions are asked and when the focus is on a contemporary phenomenon within a real-life context [15]. Case study research deals with many more variables of interest than data points. Hence, obtained results cannot be generalized statistically. However, they provide insights for building theories that are explored and evaluated with ensuing research.

Study Planning. The case study was performed in the organization described in section 2. This organization is characterized with a software product that is novel, but already has an initial user base. The product implemented the vision of a product manager who is an expert in the application domain. Corresponding to the product's development stage, the organization was small with many responsibilities bundled on a few professionals.

The organization desired to enhance its project-centered development approach by strengthening the product perspective. It decided to introduce short- and long-term planning to increase the impact that it could generate with the limited resources it had available. It decided to pilot feature-driven release planning and complemented it with roadmapping to cover timing and resource aspects [29].

The first author of this paper introduced the basic methodology to the organization and performed the case study research. The second author was the product manager who tailored and implemented the approach together with stakeholders. Over a period of a year, work results and experiences were reviewed repeatedly to collect lessons-learned and to fine-tune the implementation.

Study Operation. The authors obtained data by collecting work results created by the practitioners during release planning, by performing interviews with the project leader and steering committee members, and by reflecting on the release planning experiences. The use of multiple data sources enabled triangulation for reducing validity threats of the study results.

The collected work results included a description of product stakeholders, the feature tree, feature specifications, a detailed roadmap, and a project backlog. The collected data represented the state of the organization after the feature tree-based practice had been introduced and its use calibrated. Calibration balanced efficiency and effectiveness with the organization's needs. The data allows answering RQ1 with a multi-faceted view of how feature tree-based release planning was implemented.

The interviews surfaced the product manager's stance towards feature tree-based release planning and experiences from applying the practice. The interviews were performed on multiple occasions during and after implementing the approach. The interviews helped interpreting the work results and allowed answering RQ2.

4.2 Threats to Validity

Every empirical study has limitations. Typical threats to validity were addressed in this case study as follows.

Conclusion validity: is there a true relationship between the treatment and the outcome? Triangulation over multiple empirical data sources, accompaniment of the organization over a year, and review of the research results by the practitioners reduced threats to conclusion validity. The use of multiple views for describing how the approach was implemented provides transparency.

Internal validity: does the treatment and not something else cause the outcome? Particular threats are that second author's involvement in the release planning affects researcher bias and that already the awareness of being observed affects the behavior of practitioners [30]. The former threat was a conscious decision to increase the accuracy and completeness of the description as practiced in action research [31]. Researcher bias was controlled by triangulating data sources. The latter threat was reduced through the long-term collaboration and the repeated interviews about why the practitioner believed that the described effects were achieved.

Construct validity: do the treatment and outcome measurements adequately represent the theory? The study controlled proper feature tree use by analyzing how well the feature tree construction rules were adhered to and by letting the practitioner reflect on the technique's strengths and limitations. Effort, decision-making, and trust were evaluated by comparing the subjective practitioner views with the results of artifact analysis.

External validity: can the results of the study be generalized? The study was performed in a real-world industrial context. Such contexts differ, however, for example in terms of how innovative and how large the developed products are. It is likely that the same results can be achieved in organizations that develop new product features incrementally.

The obtained results should be further tested in follow up studies. Positive and negative replications in other contexts can corroborate or refute the results. Experiments that compare feature tree-based and backlog-oriented release planning can test whether the results generalize statistically.

4.3 Use of Feature Trees for Release Planning

Feature trees were a central element for planning software releases. They acted as pivotal point for integrating analyses of user groups and of design options, for planning product development in the form of detailed roadmaps, for steering development iterations with backlogs, and for capturing progress. This integration of the core idea, the feature trees, with related practices, the user group analysis and roadmapping, was not planned, but emerged naturally in the context of the company. The features and their traces to these other views became a basis for coordinating stakeholder involvement with product development.

User Groups. The organization desired to address the needs of important stakeholders groups with the software solution. The product manager refined these groups by defining personas [32] and by appointing representatives. The needs of these personas affected the scope of the solution and the supported use scenarios [33]. The availability of the personas' representatives for pilot projects affected the timing of corresponding feature development.

To support such analysis the product manager developed and maintained the stakeholder tree shown in Figure 4. The tree implemented the VORD viewpoint structuring concepts [34]. The needs of a given high-level group were valid for refined groups, but not vice-versa. For example the need *finding publishable media* of *ZHdK* was also valid for *Publicity* and of *Lecturer*. The need *understand frequency and sources of site visits* of *Publicity* was not applicable *ZHdK* in general.

The product manager felt too much uncertainty to draw sharp boundaries between user groups and their needs. As a consequence, the stakeholder tree was used to build a vocabulary of stakeholders and to guide analysis, but not for formally defining traceability to features. Concrete needs were elicited, and feature development re-planned if necessary, during pilot projects performed with the stakeholder representatives. The total support of a persona was documented with a bar chart.

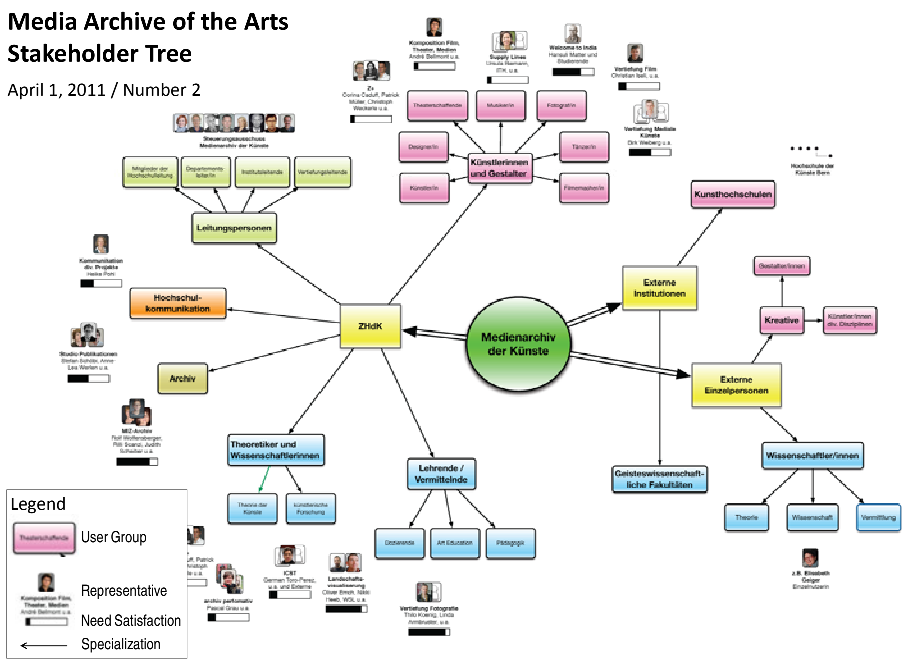

Fig. 4. Structure of the stakeholder tree. Geometric form: user groups. Photographs: user group representatives. Arrows: refinement of a generic user group to a special group. No need to read the feature names for understanding the case study.

Product Features. The feature tree provided an overview on the software solution by abstracting from requirements to features and by showing the fullest possible scope of the solution. It supported release planning by grouping requirements into cohesive units of implementation. The dependencies between these groups affected their order of implementation.

To support such analysis the product manager developed and maintained the feature tree shown in Figure 5. The tree captured the AND, OR, and REQUIRE requirements dependencies described in section 3. For example, the feature *Indexing* could not be developed before *Media Entry* and not after *Project-Oriented Indexing*. Not such dependency was defined between the features *Indexing* and *Basic Administration Interface*. The tree structure was not completely adhered to, however: some sub-features depended on more than one super-feature. The intention of these features was to combine these super-features. For example *Project-Oriented Filtering and Browsing* integrates *Filtering* and *Browsing*.

The feature tree captured the product manager's understanding of how the product should evolve. The initial tree was constructed by analyzing the originally available requirements specification based on the product manager's experience and gut feeling. The tree then was continuously evolved based on inputs from analyzing inputs elicited in stakeholder interviews and analysis of interfacing systems.

At the moment of analysis, the tree consisted of 91 features. It contained five branches with 57 functional features, one branch with 7 usability-related features, and one branch with 27 features that referred to supported media formats. The three types of branches interacted with each other. For example, adding a media format such as *Text* implied adjusting already implemented functional features. The necessary changes were planned before the implementation of the concerned media feature.

The product manager used the feature tree for reviewing progress and planned evolution with the steering committee, the reference team, and the pilot users. Color codes captured development progress, cooperation with company-external groups, and long-term scoping decisions. When planning the support of a pilot project, non-implemented but needed features were identified and integrated into the product's development sequence. The pilot projects were chosen so that the solution's key features could be implemented and validated as part of the public version 1.0 release.

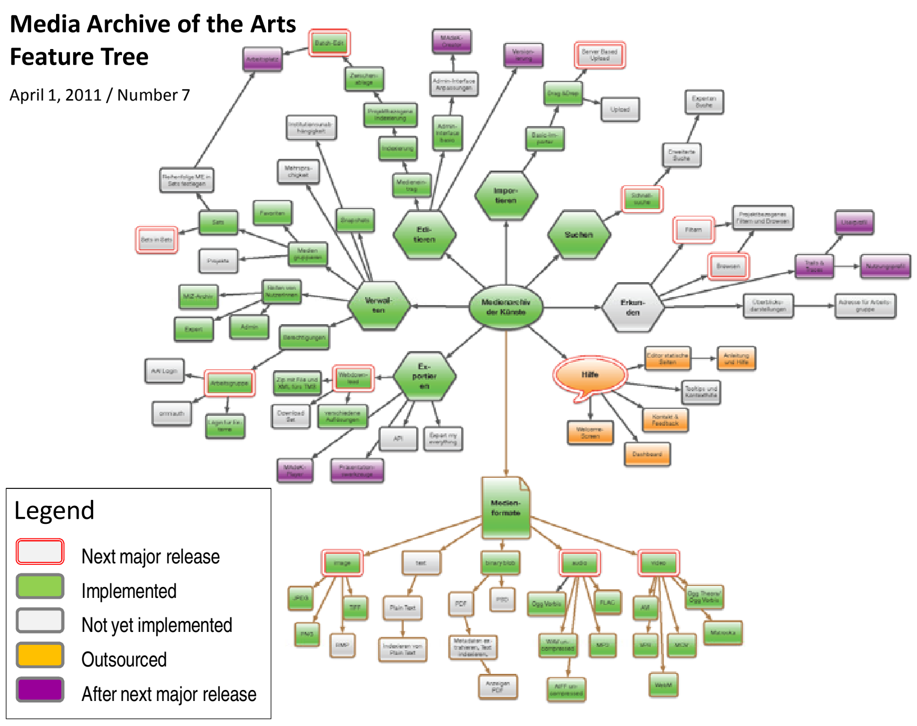

Fig. 5. Structure of the feature tree. Each geometric form represents a feature. Each arrows points from a base feature to enhancing features. No need to read the feature names for understanding the case study.

Feature Specification. The product manager used the features to align the developed solution with stakeholder needs. A feature was specified with 0 to 39 requirements. The progress of feature elaboration and development affected how far a feature was specified.

This practice allowed investing effort into those features that were implemented in near future.

No formal process was used to group known requirements into features, hence to define AND dependencies between the requirements. Instead, the product manager used her experience and gut feeling. Candidate features were then refined by removing requirements until they contained no optional requirements. The removed requirements were allocated to already known or ad-hoc defined sub-features, hence establishing REQUIRES dependencies. Alternatives, the OR dependencies, were captured by defining multiple sub-features.

Further refinement was done by considering each feature acted as a bridge between requirements and solution design [9]. The exploration of how a given feature would be implemented helped the product manager to set the right requirements and the development team to improve effort estimates. This dialogue also resolved situations where the requirements were fragmentary or specified at the wrong abstraction level.

To support the dialogue between the product manager and the development team the features were specified with the attributes shown in Table 1. The feature attributes were filled incrementally as specification and development progressed. Each feature was identified with its *name*. The product manager regularly discussed the features with the project leader and architect, leading to a *description* of the chosen of implementation alternative, early *effort* estimates, and initial *requirements*. The requirements were completed and important design aspects specified just before the feature was implemented. At the moment of feature implementation, the requirements were used to form the project backlog. A *comments* attribute provided a discussion forum for clarifications and coordinating implementation. *Bugs* and *future requirements* were placeholders for documenting maintenance and future enhancement needs.

Table 1. Feature specification attributes

Attribute	**Description**	**Example**
Name	Identifier	*Indexing*
Description	Feature's key ideas: concept describing the chosen implementation alternative	*Capture as much meta data as possible with input assistance, resp. an editor. Formalized metadata can be used for filtering and browsing.*
Effort	Estimated implementation effort	*35 points*
Requirements	Project backlog	*18 concluded requirements:* - *Keyword field* - *Standardized thesaurus* - *Visualize geo data with google maps widget…*
Attachments	Specification of important design aspects	*(examples of GUI elements)*
Comments	Discussions related to clarifications and open issues	*We can close Indexing if we close the ticket […].*
Bugs	Problems with the implemented solution	*20 resolved, 2 pending bugs such as* - *Auto complete does not work…*
Future Requirements	List of potential enhancements of the feature	*12 not implemented requirements:* - *New media files for already existing meta data Icons…*

Formal feature specification in the context of software product lines expects specification of requirements, domain assumptions, and solution [26]. This specification practice was calibrated to increase work efficiency and flexibility and to support depending activities, while accepting dependency on the involved practitioners for interpreting the documentation. Information used to steer and track development was specified: the explicit list of requirements, enhanced with effort estimates and lists of bugs and future requirements. Knowledge related to understanding the features was kept implicit. Domain assumptions that would relate the feature to its use scenarios and the users' personas were not documented. The solution that would describe how to implement the feature was only fragmentarily documented. Lack of such information was compensated with the discussion thread.

Roadmap. The product manager planned a hierarchy of development iterations. Full version releases, for example version 1.0, had to address all key needs of selected stakeholder groups, for example the *ZHdK* stakeholders. Such a version release was split into feature releases that supported the needs of selected pilot projects. The development project then had bi-weekly releases to provide transparency and feedback to the product manager.

The feature trees lacked timing information. To define the feature's development timing the product manager decided to use a detailed, layered product roadmap [35] with a time horizon of two years. Figure 6 shows an extract of the detailed first-year plan. The second year was more fragmentary. The layer *features* defined when given features would be implemented. A feature's spacing corresponded to its development duration that was computed based on estimated effort, available *resources*, and availability of *technologies*. For example, *Authorization* was dependent on AAI and required roughly one calendar month. The availability of a feature enabled *use scenarios* that were needed by the *pilot projects*. For example, *Authorization*, *Login for Externals*, *Work Groups*, and *Download of Different Resolutions* enabled the *Production* scenario that was first evaluated in the *Z+* and *Studio Publications* pilots. The top-most layer referred to milestones such as external events and own releases.

The roadmap provided the context for release planning. It allowed exploring planning options together with stakeholders to agree on the implementation sequence. Time-to-market of version 1.0 was expected to be minimized and piloting aligned with development activities. The critical path was represented by the sequence of double-edged key features. Availability of pilot projects was documented by defining their start and end points. Surprises that affected the planning were discussed with the steering committee. For example, development staff was increased to account for development delays. The roadmap simplified release planning to allocating imminent features, for example *Filter* and *Extended Search* to imminent development iterations.

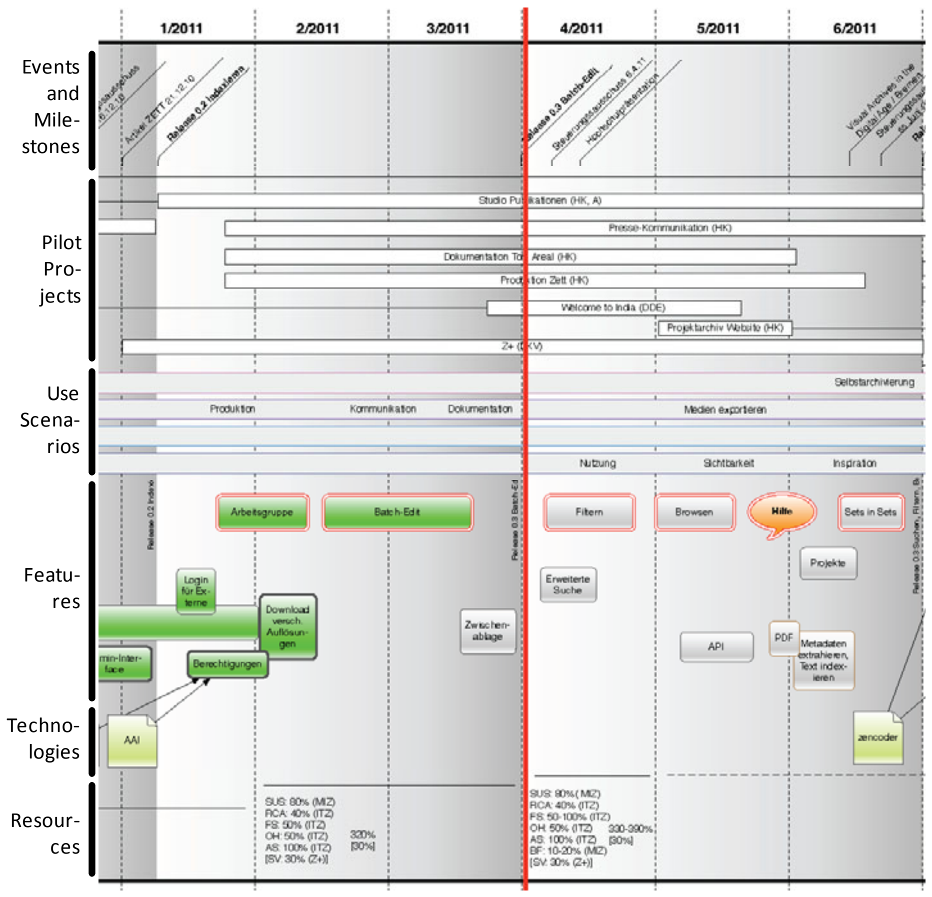

Fig. 6. Product roadmap (extract). Red bar: moment when the snapshot was taken. No need to read the detailed contents for understanding the case study.

Impact of Feature Trees

Effort. The feature tree, in comparison with a flat backlog of requirements, reduced complexity of release planning. The abstraction from requirements to features reduced the total number of elements to be considered by a factor 10.3. Table 2 evaluates the situation at April 2011. Row 1 describes the effect of the AND grouping. Row 2 describes the effect of adding the REQUIRES dependencies. Row 3 shows the complexity of prioritizing the implementation candidates, row 4 of the roadmap, and row 5 of the feature release project where the focus shifted from features to requirements.

Table 2. Comparison of list-based and feature tree-based approach

*: The feature-tree based requirements catalogue was intentionally incomplete. The estimate is extrapolated from the statistics of fully specified features.	**Flat Backlog: Requirements**	**Feature Tree: Features**
1 Total number of elements	937*	91
2 Number of implementation candidates	453*	23
3 Number of comparisons, efficient algorithm: $O(n \log_2 n)$	3997*	104
4 Number of elements in backlog of major release	206	20
5 Average number of elements in backlog of feature release	21	2

The product manager perceived planning of about twenty items fine-grained enough and feasible. Still discussions often centered on an even smaller set of features and did not need as much detail information about context as the tree provided.

Decision-Making. The feature tree and the roadmap were the key instruments used for deciding what to implement and when to implement. The feature tree provided a basis to discuss the scope of pilot projects with the stakeholders identified in the stakeholder tree. Stakeholder needs that could not directly be addressed led to discovering new potential features.

The roadmap was used for aligning the timing of feature implementation with the pilot project. The product manager had to ensure that needed features were available to the pilot users at the right moment in time and that no unnecessary feature was implemented. The roadmap was useful to check these rules together with the concerned stakeholders.

A number of criteria are known to evaluate product evolution options [4]. They include management concerns like development cost-benefit, business concerns like stakeholder priority and satisfaction, and system concerns like evolvability. Such information that is typically part of a business case [36] was not specified explicitly. Instead, the impact of these concerns was discussed in terms of product evolution scenarios. The agreement on which scenario to pursue was documented in the form of features in the feature tree and as timing information in the roadmap.

Traceability between features, use scenarios, and pilot projects was difficult to maintain, however. This difficulty now motivated the product manager to evaluate how specification of use scenarios, for example in terms of supported user groups and supporting features, could be used to bundle traceability. This approach could reduce the number of traces between stakeholders and features by a factor ten to hundred.

Development and use of the so far implemented solution led to massive learning about the real user needs and about what an effective media management solution is. Hence, even-though the product manager accepted a feature to be finished, new non-implemented requirements were added to the feature. These requirements are planned to be structured as features and enter development through enhancements of the feature tree shown in Figure 5.

Trust. In comparison to a flat list of requirements, the feature tree allowed building a mental model of the solution. The reduced number of features allowed building a shared vocabulary with stakeholders, the color coding visualizing growth of the solution, and AND-OR feature dependencies understanding design options. This focused discussions and communication with stakeholders on aspects that were essential for planning. Decisions could be taken together with these stakeholders, which led to trust in the plans and in the product organization.

Surprises and problems emerged despite the common decision making. For example, the feature tree only captured usability-related quality requirements. The pilot projects discovered that the solution's performance was too low. The resolution of that problem led to changes in technologies and architecture and required significant amount of unplanned time. The product manager now started to specify and plan quality with dedicated feature vectors [37].

5 Discussion and Conclusions

This paper has explained how feature trees [38] can be used to structure requirements and simplify release planning, hence to support release planning [20], i.e. the planning of variability over time [24]. AND relationships [13] can be exploited to group requirements into features. Feature vectors [27] can be built by exploiting REQUIRES dependencies. Features that have the same super-feature stand in an OR relationship. The resulting tree can be used for planning the development of the specified software and for controlling development progress.

The paper has shown a revelatory industrial case to evaluate feasibility and impact of the approach. The practitioners integrated the feature tree into stakeholder and need analysis, adapted the feature specification to communicate requirements and to manage the development project, and integrated the features into a roadmap that aligned the timing of pilot projects and development.

The approach reduced complexity of release planning that before would have been made with flat requirements lists [16]. The feature tree, combined with a roadmap, was a key instrument to plan development that allowed the product manager to make decision together with stakeholders. The visualization of the requirements as a feature tree allowed them building a mental model and a shared vocabulary. As a consequence, the stakeholders developed trust in the decision-making and in the product organization.

As any other approach, feature-tree based release planning had limitations, however. Documentation was based on office tools and traceability often kept implicit. Decisions, even though made together with the concerned stakeholders, turned out to be wrong because of omissions and rarely perfect estimates. These two issues made analysis of dependencies and coordination of stakeholders difficult.

The presented work has relations to other research beyond feature trees and release planning. The described feature trees are a new kind of AND/OR trees that differs from AND/OR goal trees [39]. The feature trees do not represent means-ends relationship, but dependencies in the implementation order. The documentation of a single feature, however, can be made with a goal tree. For example, the feature specification attributes *requirements* and *description* corresponded to two abstraction levels and were used to capture means-ends relationships [8]. Such feature-oriented goal trees specification is narrow in scope and can be developed incrementally. It hence has the potential to improve the scalability of goal modeling.

The case shows how feature trees can integrate roadmapping [35] and software specification. It has extended a the layered form of product roadmaps encountered in small companies [40] with explicit traceability to product feature. Such traceability allows understanding the impact of changes, for example changed effort estimates, to the other aspects of release planning, such as stakeholder support, and piloting.

Future research should replicate the study in different contexts to better understand when and how feature tree-based release planning should be used. Experimentation that compares the feature tree-based approach with the use of flat requirements backlogs provide statistical analysis of effort reduction and eliminate the potential presence of the Hawthorne effect.

Future research should enhance the presented approach with an understanding of how traceability, for example between features and stakeholders, can be structured to enhance understanding of these traces and effort for handling traceability. Also tool support can greatly simplify consistency management between the feature tree and related views and ease what-if analyses for exploring software development planning options.

References

1. Svahnberg, M., Gorschek, T., Feldt, R., Torkar, R., Bin Saleem, S., Usman Shafique, M.: A Systematic Review on Strategic Release Planning Models. Information and Software Technology 52, 237–248 (2009)
2. Cohn, M.: Agile Estimating and Planning. Prentice Hall (2006)
3. Amandeep, N.F.N.G., Ruhe, G., Stanford, M.: Intelligent Support for Software Release Planning. In: Bomarius, F., Iida, H. (eds.) PROFES 2004. LNCS, vol. 3009, pp. 248–262. Springer, Heidelberg (2004)
4. Wohlin, C., Aurum, A.: What is Important when Deciding to Include a Sotware Requirement into a Project or Release. In: International Symposium on Empiricial Software Engineering (2005)
5. Berander, P., Andrews, A.: Requirements Prioritization. In: Aurum, A., Wohlin, C. (eds.) Engineering and Managing Software Requirements. Springer, Heidelberg (2005)
6. Karlsson, L., Regnell, B., Karlsson, J., Olsson, S.: Post-Release Analysis of Requirements Selection Quality - An Industrial Case Study. In: 9th International Workshop on Requirements Engineering: Foundation for Software Quality, RefsQ 2003 (2003)
7. Karlsson, L., Dahlstedt, Å., Regnell, B., Natt och Dag, J., Persson, A.: Requirements Engineering Challenges in Market-Driven Software Development - An Interview Study with Practitioners. Information and Software Technology 49, 588–604 (2007)
8. Gorschek, T., Wohlin, C.: Requirements Abstraction Model. Requirements Engineering 11, 79–101 (2006)
9. Fricker, S., Gorschek, T., Byman, C., Schmidle, A.: Handshaking with Implementation Proposals: Negotiating Requirements Understanding. IEEE Software 27, 72–80 (2010)
10. Lehtola, L., Kauppinen, M.: Suitability of Requirements Prioritization Methods for Market-driven Software Product Development. Software Process Improvement and Practice 11, 7–19 (2006)
11. Fricker, S., Schumacher, S.: Variability-Based Release Planning. In: Regnell, B., van de Weerd, I., De Troyer, O. (eds.) ICSOB 2011. LNBIP, vol. 80, pp. 181–186. Springer, Heidelberg (2011)

12. Schobbens, P.-Y., Heymans, P., Trigaux, J.-C., Bontemps, Y.: Generic Semantics of Feature Diagrams. Computer Networks 51(207), 456–479
13. Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B., Natt och Dag, J.: An Industrial Survey of Requirements Interdependencies in Software Product Release Planning. In: 5th IEEE International Symposium on Requirements Engineering (2001)
14. Haberfellner, R., Nagel, P., Becker, M., Büchel, A., von Massow, H.: Systems Engineering: Methodik und Praxis. Verlag Industrielle Organisation (2002)
15. Yin, R.: Case Study Research: Design and Methods. SAGE Publications (2009)
16. Vlaanderen, K., Jansen, S., Brinkkemper, S., Jaspers, E.: The Agile Requirements Refinery: Applying Scrum Principles to Software Product Management. Information and Software Technology 53, 58–70 (2011)
17. Bekkers, W., van de Weed, I.: SPM Maturity Matrix. Utrecht University (2010)
18. Regnell, B., Beremark, P., Eklundh, O.: A Market-Driven Requirements Engineering Process: Results from an Industrial Process Improvement Programme. Requirements Engineering 3, 121–129 (1998)
19. Davis, A.: Just Enough Requirements Management. Dorset House Publishing (2005)
20. Carlshamre, P.: Release Planning in Market-Driven Software Product Development: Provoking an Understanding. Requirements Engineering 7, 139–151 (2002)
21. Ruhe, G., Saliu, M.O.: The Art and Science of Software Release Planning. IEEE Software 22, 47–53 (2005)
22. Li, C., van den Akker, M., Brinkkemper, S., Diepen, G.: An Integrated Approach for Requirements Selection and Scheduling in Software Release Planning. Requirements Engineering 15, 375–396 (2010)
23. Karlsson, J., Ryan, K.: A Cost-Value Approach for Prioritizing Requirements. IEEE Software 14, 67–74 (1997)
24. Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Heidelberg (2005)
25. Zowghi, D., Coulin, C.: Requirements Elicitation: A Survey of Techniques. In: Aurum, A., Wohlin, C. (eds.) Engineering and Managing Software Requirements. Springer, Heidelberg (2005)
26. Classen, A., Heymans, P., Schobbens, P.-Y.: What's in a Feature: A Requirements Engineering Perspective. In: 11th International Conference on Fundamental Approaches to Software Engineering, Budapest, Hungary (2008)
27. Nejmeh, B., Thomas, I.: Business-Driven Product Planning Using Feature Vectors and Increments. IEEE Software 19, 34–42 (2002)
28. Stoiber, R., Glinz, M.: Feature Unweaving: Efficient Variability Extraction and Specification for Emerging Software Product Lines. In: 4th International Workshop on Software Product Management (IWSPM 2010), Sydney, Australia (2010)
29. Phaal, R., Farrukh, C., Probert, D.: Strategic Roadmapping: A Workshop-Based Approach for Identifying and Exploring Strategic Issues and Opportunities. Engineering Management Journal 19, 3–12 (2007)
30. Draper, S.: The Hawthorne, Pygmalion, Placebo and Other Effects of Expectation: Some Notes, vol. 2011 (2010)
31. Davison, R., Martinsons, M., Kock, N.: Principles of Canonical Action Research. Information Systems Journal 14, 65–86 (2004)
32. Pruitt, J., Grudin, J.: Personas: Practice and Theory. In: 2003 Conference on Designing for User Experience (DUX 2003), New York, NY, USA (2003)

33. Carroll, J. (ed.): Scenario-Based Design: Envisioning Work and Technology in System Development: Envisioning Work and Technology in Systems Development. John Wiley & Sons (1995)
34. Kotonya, G., Sommerville, I.: Requirements Engineering with Viewpoints. Software Engineering Journal 11, 5–18 (1996)
35. Phaal, R., Farrukh, C., Probert, D.: Technology Roadmapping - A Planning Framework for Evolution and Revolution. Technological Forecasting and Social Change 71, 5–26 (2003)
36. Schmidt, M.: The Business Case Guide. Solution Matrix (2002)
37. Regnell, B., Berntsson Svensson, R., Olsson, S.: Supporting Roadmapping of Quality Requirements. IEEE Software 25, 42–47 (2008)
38. Schobbens, P.-Y., Heymans, P., Trigaux, J.-C., Bontemps, Y.: Generic Semantics of Feature Diagrams. Computer Networks 51, 456–479 (2007)
39. van Lamsweerde, A.: Goal-Oriented Requirements Engineering: A Guided Tour. In: 5th IEEE International Symposium on Requirements Engineering (RE 2001), Toronto, Canada (2001)
40. Vähäniitty, J., Lassenius, C., Rautiainen, K.: An Approach to Product Roadmapping in Small Software Product Businesses. In: 7th International Conference on Software Quality (ECSQ 2002), Helsinki, Finland (2002)