

Visualization of Release Planning

Korbinian Herrmann
Technische Universität München
Chair for Applied Software Engineering
herrmann@in.tum.de

Abstract

Software release planning is a complex task, as many different factors must be considered. Project managers can't cope with the complexity of release planning. Project managers need a tool support in release planning. Existing tools for release planning are not appropriate or are one tool in a set of specialized tools in a project. Using more tools leads to inconsistent data among this set of tools.

This paper proposes a decision support tool for release planning: Basis of the tool is the Sysiphus framework, which stores all the information related to a software project in a single repository. Knowledge nuggets are introduced as a concept to aggregate and develop knowledge that belongs to a specific release. The paper shows a prototypically implemented visualization for planning multiple releases.

1. Introduction

By shortening product cycles in software development the percentage of releases on a total project has become significant. Moreover, iterative and incremental software engineering has lead to multiple releases. A release is a version of the software system that has been made available externally [Bruegge 2004]. Release planning is the mapping of requirements to releases. There are two types of release planning: Road-mapping and operational planning. Road mapping is the strategic planning of releases for several months in advance. Operational planning is the plan for the next weeks. A release plan is influenced by several factors such as the requirements themselves, their implementation, urgency for the client, value for the developing company, risk management and personal decisions. [Ruhe 2005] In large software projects project managers can't cope with the complexity of these factors. There is a need to support project managers with a decision support tool. It has to provide and visualize the information, which is

necessary for planning multiple releases. [Carlsham 2002] constitutes that the tool must be interactive and responsive.

Existing tools for release management, such as the Release Planner Provotype [Carlsham 2002], do not consider the factors that influence the decision [Ruhe 2005] Existing tools that respect the factors of release planning, like the Release Planner [Ruhe 2005], are solutions for release planning only and do not support other tasks such as requirements elicitation or testing. As a result the current practice is to use a set of tools in a single software project, each of them supporting a single task. These tools are independent of each other and have their own repository to store the project data. As a consequence the data basis of the project is inconsistent and not up-to-date.

This paper proposes a solution for that problem with three things: First, a framework called Sysiphus that provides a single repository for the data in a project. Second, a concept called knowledge nuggets to aggregate the information necessary for planning releases and finally, a visualization of the data that are affected by the release planning.

2. Sysiphus

Sysiphus [Sysiphus 2006] is a framework that provides a unified approach to represent the knowledge, in particular of communicating, organizing and modeling, which exists for each software project. This knowledge is represented by elements that are interconnected by links. The elements and links can be of any type. This allows interconnecting knowledge of the three fields communicating, organizing and modeling. Examples for elements are emails, persons and classes of an object model; examples for links include associations and links, which connect classes with persons, who are therefore responsible. Elements and links form an interconnected graph, where the knowledge of a single project is stored. Figure 1 shows an example for a graph with elements of

communicating (C), organizing (O) and modeling (M) [Wolf 2004]

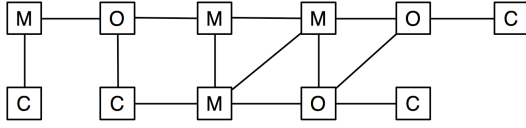


Figure 1: The Sysiphus framework allows connecting knowledge of any type using an interconnected graph.

Release planning is complex because it involves knowledge of different fields. This knowledge keeps on developing over time and must be continuously mapped to the releases it belongs to.

Sysiphus is a good basis as the knowledge of a software project is held together. Nevertheless, introducing releases into a framework such as Sysiphus leads to a problem: Parts of the elements in the graph need to be mapped to releases. This mapping is not static as the knowledge keeps on developing for each release. This problem can be solved by the concept of knowledge nuggets, which is explained in the next section.

3. Knowledge Nugget

A knowledge nugget is a subgraph of the Sysiphus graph. The subgraph contains the knowledge that a stakeholder needs to perform his task in the project, e.g. release planning. For each task an instance of a knowledge nugget can be used to manage the knowledge. Figure 2 shows three instances of nuggets for the Sysiphus graph of figure 1: Each of them is a subgraph of the underlying graph and assigned to a stakeholder. An ellipse with a dashed line contains all the nodes of the subgraph, which belongs to a nugget.

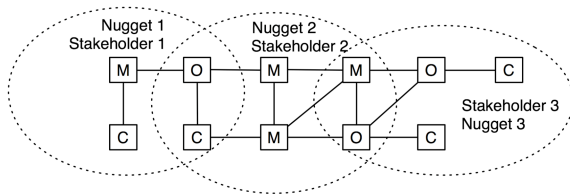


Figure 2: Knowledge nuggets for the Sysiphus graph in Figure 1

Instances of knowledge nuggets (N) can be ordered in a partial way:

$$N_0 < N_1 < \dots < N_n$$

Knowledge nuggets are used for both modifying and extracting the knowledge: After modifying an

element the corresponding knowledge nugget instance decides if the modification affects other knowledge nuggets instances. If there is an order defined on the knowledge nuggets instances, these modifications will be propagated automatically to other nuggets. Extraction of knowledge is the second use of knowledge nuggets. The knowledge nugget instance aggregates elements that belong together in a project.

An example for a knowledge nugget instance is a release. This instance aggregates the knowledge belonging to a specific release. It contains all elements of communicating, organizing and modeling, which are part of the release. To model a series of releases, we use the order defined on knowledge nuggets and introduce a temporal relationship $<_r$. For example

$$R_p <_r R_C <_r R_F$$

A past release R_p precedes the current release R_C that in turn precedes a future release R_F . This relationship allows project managers currently planning a release to focus only on elements of the current release.

The next section describes how knowledge nuggets support the development of different releases for a system.

4. Knowledge Nuggets for Releases

Knowledge nuggets support not only the planning of the releases but also the development of the system models of a release. As this paper concentrates on requirements, first the support for modeling the requirements is described. Second, the paper proposes on how a project manager is supported during assigning requirements to releases.

4.1 Modeling requirements

The requirements engineer models the requirements of the system in form of system models such as features, use cases, nonfunctional requirements and an analysis object model. Starting from an initial set of system models, this continues developing from release to release: Features, use cases, nonfunctional requirements and objects are modified, added or removed. The application of these operations on the system model is called refinement.

This refinement can be described as homomorphism: Reality R is transformed into a model M_1 . Performing a function on the transformed model M_1 results in the same model M_1 as performing the function in reality R and transforming the result to the model again. (see Figure 3)

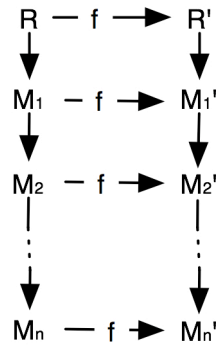


Figure 3: Homomorphism is the basis for developing requirements for multiple releases.

Sysiphus supports refinement. Every model of the homomorphism is mapped to a release. Looking at figure 3, M_1 is the system model for the requirements of the first release, M_2 the one of the second release and M_n the one of release n . These system models of the requirements are refined by the homomorphism: M_2 is a refinement of M_1 . As a release is an instance of a knowledge nugget, all elements, e.g. features, use cases, nonfunctional requirements and objects, belonging to a specific release are grouped together. The system traces changes made between the refinements. This allows tracing how single elements develop during the evolution of the releases for a system.

Applying the temporary order of releases, the knowledge nugget can be used to propagate changes in the model of an earlier release to the model of a later one, but not vice versa. For example, changes in a use case in the third release for a system will be propagated automatically to all releases after the third one, but not to the first and second release. In that way knowledge nuggets help the requirements engineer to maintain consistency between the models of different releases.

Consider the analysis object model of a multimedia management system as an example. The first release supports songs only: The analysis object model contains the classes Product, Audio Stream and Song (see Figure 4).



Figure 4: Analysis object model for the first release for a multimedia management system

In the second release new use cases supporting podcasts and TV shows are added. This results in a refined analysis object model containing a new class

Podcast as a subclass of Audio Stream and a class TV Show with the superclass Video Stream (see Figure 5).

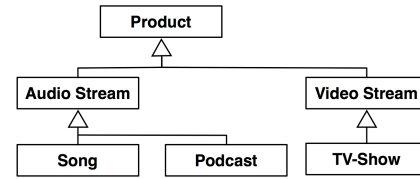


Figure 5: Analysis object model for the second release for a multimedia management system

The analysis object model of the third release contains additionally to the classes of the model of the second release the class Football Game as a subclass of Video Stream (see figure 6).

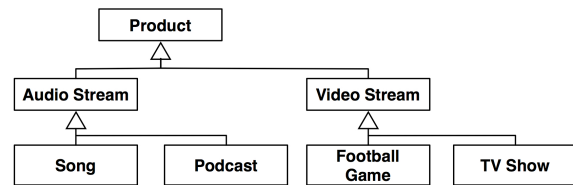


Figure 6: Analysis object model for the third release for a multimedia management system

During a review of the analysis object model of the second release one observes that songs and podcasts have authors. Adding a new class Author refines the model. This change of the model of the second release (without figure) is automatically propagated to the model of the third release. Figure 7 shows the model of the third release after the refinement. The model of the first release does not change and stays as shown in figure 4. The reason is the hierarchy of releases: The first release is earlier, e.g. past, than the second, e.g. current, and the second release is earlier than the third one, e.g. future.

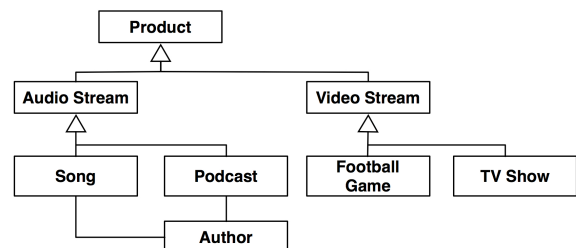


Figure 7: Analysis object model for the third release for the multimedia management system after a refinement of the object model in the second release. Compare to figure 6.

4.2 Managing releases

This section describes how knowledge nuggets can be used to plan releases: A project manager specifies the use cases that a release implements. Many factors influence his decision. Sysiphus supports a project manager during his selection by showing for every use case factors such as the priority for the client, risks, classes and assigned persons. This is realized by the connection of knowledge in the Sysiphus graph. To support the project manager even better, the system allows him to evaluate alternatives of release plans. He can define more alternative releases. Each of them is an instance of a knowledge nugget. Each nugget contains the knowledge that belongs to the corresponding alternative.

Modifying an alternative for a release or a release itself work identically: Modifications lead to an immediately change of the model: If a release plan changes and in consequence a use case moves to another release, the elements of the corresponding nuggets will be affected: Besides the use case itself, the classes related to it, action items and persons move to the other release.

For example, consider the multimedia management system described above: \$2.99 TV shows can't be realized in the second release and are moved to the third release: This leads to an immediate change of the class model of the second and third release: The video Stream and TV Show classes are removed from the model of the second release and added to the model of the third release. Figure 8 shows the class model of the second release after the modification.

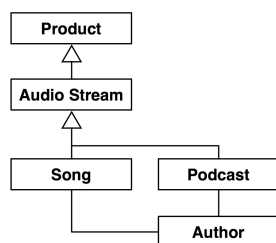


Figure 8: Analysis object model for the second release for the multimedia management system after moving the Use Case \$2.99 TV Shows to the third release. Compare to figure 5.

6. Visualization of Release Planning

The visualization of releases has been prototypically implemented by an application, called Rational-Based

Analysis Tool (RAT). The visualization consists of two main parts: On the left side there is a navigation frame to select elements. Selected elements are displayed and can be edited in an element panel, which is displayed in a frame on the right side of the application or in an own, specific window. Several elements can be opened and edited at the same time.

6.1. Navigation frame

The navigation frame provides views of the elements of a project and serves as entry point to view or edit elements. A view is a structured subset of elements. Three mechanisms are used to provide views of the elements: Documents, reports and an index.

Documents provide a structure for the elements in a project. The project administrator can define documents such as a Requirements Analysis Document to manage the requirements of a project. For release management a Roadmap document can be used that contains a section for every release. These sections consist of the use cases of the release (see figure 9).

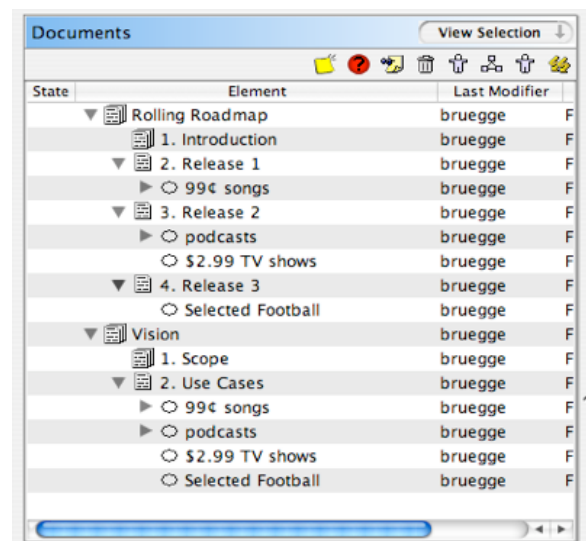


Figure 9: Navigation frame of RAT in the document view. The Rolling Roadmap is a document to manage releases.

Other views can be created using the report mechanism. It allows selecting an element according to defined constraints, for example all use cases of the first release. The third mechanism to select elements is the index: It allows searching elements by attributes such as name, type, release or description.

6.2. Element panel

The element panel is used to visualize elements. The visualization depends on the element itself: Some are textual other graphical. For example, the analysis object model can be displayed both as a list of classes and as an UML class diagram. The visualization serves for three things: Editing the elements by the user, showing the elements to the user, and navigating the model.

Editing the Use Case \$2.99 TV-shows of the multimedia management system, moving it from the second release to the third one (see Figure 10), leads to an immediate update of the user interface.



Figure 10: Element panel of RAT: The use case \$2.99 TV-shows is moved to the third release

Consequently, the navigation frame is updated immediately after assigning the use case to the third release. If there is another open element window this is updated immediately, too. Consider an element window with a class diagram of the second release. After assigning the use case \$2.99 TV shows to the third release, the classes related to that use case Video Stream and TV Show are removed from the class model (see Figure 11).

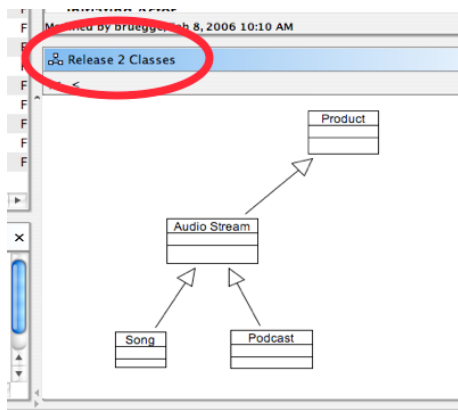


Figure 11: Element panel of RAT visualizing the class model of the second release after moving the Use Case \$2.99 TV shows to release 3.

5. Summary

This paper proposed a solution of planning and visualizing multiple releases. Basis of the solution is the Sysiphus framework, which connects the knowledge of a software project in a single repository. This avoids inconsistencies in the knowledge of a software project. Knowledge nuggets are used to aggregate the knowledge that belongs to a specific release. Additionally, they allow refining the knowledge concerning a specific release. This includes developing system models of the release. The consequences of changes both in the system model and in the release plan take effect immediately after performing the change. The visualization shows the knowledge that belongs to specific releases. A prototypical implementation visualizes the usage of knowledge nuggets with use cases and system models.

6. References

[Bruegge 2004] Bernd Brügge, Allen Dutoit: *Object Oriented Software Engineering - Using UML, Patterns and Java*. Prentice Hall, Upper Saddle River (NJ), 2nd edition, 2003.

[Carlshamre 2002] Carlshamre, Pär, "Release Planning in Market-driven Software Product Development - Provoking an Understanding, *Requirements Engineering Journal*, 7(3), Springer, London, 2002, pp. 139-151.

[Ruhe 2005] G. Ruhe: "Software Release Planning", *Handbook of Software Engineering and Knowledge Engineering*, Vol. 3 (S.K. Chang, Ed.), World Scientific 2005, pp 365-394.

[Wolf 2004] Timo Wolf, Allen Dutoit: "An Rationale-based Analysis Tool", *Proc. Of the 3th International Conference on Intelligent & Adaptive Systems and Software Engineering*, Nice, 2004. pp. 209-214.

[Sysiphus 2006] <http://sysiphus.in.tum.de> [02.03.2006]