Replan: A Release Planning Tool

David Ameller[1], Carles Farré[1], Xavier Franch[1], Antonino Cassarino[2], Danilo Valerio[2], Valentin Elvassore[1]

[1] Universitat Politècnica de Catalunya
Barcelona, Spain
{dameller, farre, franch}@essi.upc.edu,
valentin.elvassore@gmail.com

[2] Siemens AG Österreich
Vienna, Austria
{antonino.cassarino, danilo.valerio}@siemens.com

***Abstract*—Software release planning is the activity of deciding what is to be implemented, when and by who. It can be divided into two tasks: strategic planning (i.e., the *what*) and operational (i.e., the *when* and the *who*). *Replan*, the tool that we present in this demo, handles both tasks in an integrated and flexible way, allowing its users (typically software product managers and developer team leaders) to (re)plan the releases dynamically by assigning new features and/or modifying the available resources allocated at each release. A recorded video demo of Replan is available at https://youtu.be/PNK5EUTdqEg.**

***Index Terms*—Software Release Planning, Feature Scheduling, Resource Allocation**

I. Introduction

Software Release Planning (SRP) solves the problem of finding the best combination of features or requirements to implement in a sequence of releases. SRP seeks to maximize business value and stakeholder satisfaction without neglecting the constraints imposed by the availability of adequate resources and the existence of dependencies between features[1], among other constraints [1].

The SRP activity is composed of two phases: *Strategic planning*, the selection of features to be included in the next release(s); and *Operational planning*, the assignment of these features to a concrete team of developers [2]. Although this distinction between the two planning phases is neat from a conceptual perspective, it may provoke some practical problems. Consider for instance the case in which strategic planning takes into account the priority of features and operational planning considers the skills of the available developers. In this situation, it could happen that one feature selected during the strategic planning phase cannot be assigned to any developer during the operational planning phase because none of them has the required skills.

The tool presented in this demo, *Replan*, handles both phases together making it possible to consider all the release planning objectives in a single execution. Moreover, Replan supports flexible release re-planning by allowing the modification at any time of the features to be included in the releases, as well as of the resources available in each release. The intended users of our tool are software product managers that need a reliable and usable tool to (re)plan product releases as well as software team leaders that need a flexible tool to (re)plan the assignment of tasks to developers.

We must point out that the features that Replan deals with are already prioritized. Therefore, tasks like feature negotiation and prioritization, which can be considered part of the strategic planning activity, are not currently supported by our tool. However, a simple API is provided to allow any external tool to send its prioritized features to Replan.

The source code of Replan as well as an online demo version are available at http://www.essi.upc.edu/~gessi/replan. The rest of the paper is divided as follows: Section II, related work; Section III, the conceptual idea behind the tool; Section IV, the architectural and technical aspects of the tool; Section V, our preliminary evaluation results; and Section VI, the conclusions and future work.

II. Related Work

A literature review of the SRP models proposed in the academic literature can be found in [3]. In that survey, in which some of the authors of this paper participated, up to 17 SRP models published in 2009-2016 were identified and analysed. Among the conclusions of that survey, we highlight the following:

- Most of the SRP models examined focused only on strategic planning. In this way, no inputs such as required skills and resource availability are considered by those models.
- Poor industry validation of the SRP models due to scarce industry involvement.
- Poor tool support for the proposed SRP models. Most of these solutions only provided some proof-of-concept tool support.

The only exception to the last point is *ReleasePlanner* [4], a commercial SRP tool used by some of the SRP models discussed in [3]. However, ReleasePlanner only addresses strategic planning. Moreover, ReleasePlanner requires its users to provide a huge amount of complex input to produce a release plan, which impacts negatively on its perceived usability.

III. Conceptual Idea

Figure 1 summarizes the main concepts managed by Replan. At the core of our tool lies a Next Release Problem (NRP) solver whose mission is to produce a release plan consisting of a set of features that are not only scheduled but also assigned to

[1] For the sake of brevity, from now on we will refer only to features, not requirements.

This work is a result of the SUPERSEDE project, funded by the EU's H2020 Programme under the agreement number 644018. https://www.supersede.eu/

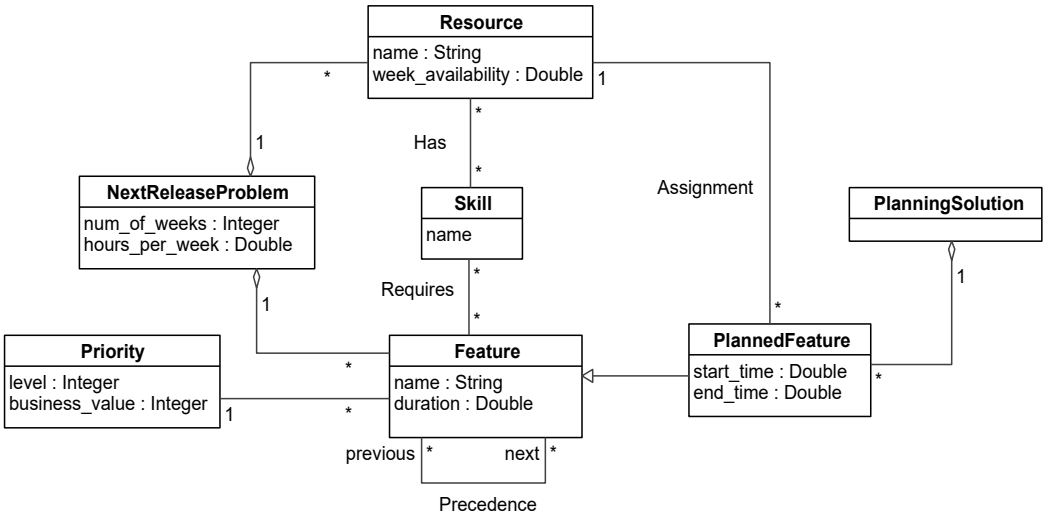

Fig. 1 Problem and solution domains

the resources that will implement them. Its execution is triggered whenever the user adds/changes/removes a release feature or resource. Each *NextReleaseProblem* instance must have defined the number of weeks that the release will last (*num_of_weeks*) and the amount of time (*hours_per_week*) that a full time employee may work in the release.

Obviously, the features that we propose to be considered in the release must be also specified. For each *Feature*, two key parameters must be defined: its *duration* (i.e. an estimation of the time effort that its implementations will require) and its *priority*. *Priority* is composed of two values: its *level* (on a 1-5 scale, from highest to lowest) and its *business_value.* The business value allows us to define a weighted value for the priority of a feature. For instance, a priority with level = 1 may have a business value = 16, whereas with a level = 2 we may have business value = 8 (the greater the value, the better).

Precedence dependencies among features are modelled by indicating, for each feature, which other ones need to be implemented *previous*ly. We can also model the different *Skills* that each feature *Requires* in order to be implemented.

Apart from features, a *NextReleaseProblem* should include the (human) resources that are available to implement the features assigned to a release. For each *Resource* (e.g. a developer) we need to know his/her *week_availability* (the percentage of his/her working time that s/he can devote to the release) and the skills that s/he *Has*.

An instance of a *PlanningSolution* for a given *NextReleaseProblem* will consist of several *PlannedFeatures*. Each PlannedFeature is a Feature assigned to a Resource, during a well-defined time slot [*start_time*, *end_time*] given in relative terms to the start of the release.

Due to the NP-hardness of NRP [5], we adopt the common approach of formulating it as an optimization problem that is solved in a reasonable amount of time by searching for good solutions that are not necessarily the best ones. In particular, our approach is that of a bi-objective optimization problem where we want to achieve the following objectives:

- Maximize the sum of the business value of the planned features.
- Minimize the total duration of the release implementation

Without violating any of the following constraints:

- Features must be implemented by employees that have the necessary skills.
- The week availability of employees.
- The precedence dependencies among features.
- The number of weeks of the release.

IV. Tool Description

The architecture of the Replan tool is depicted in Figure 2. It consists of two main parts: a dashboard for the web-based front-end, and a service-based backend. This latter is divided into two services: the controller that manages all the communications and persistence needs, and the optimizer that executes the main functionality of the tool. Having this separation has several benefits. First, since the optimizer may require more computational power, being an independent web service facilitates the possibility of moving it to a dedicated machine. Second, we facilitate the possibility of having several independent implementations of the optimizer service and the controller can use the one that is the most appropriate in each case. For instance, we can have an optimizer service that is more adequate for situations with many constraints and an optimizer service that is more adequate for situations with many resources. Moreover, the controller offers a separate API that allows external tools, such as decision support systems, to feed Replan with lists of the prioritized features to be enacted.

The normal workflow of the Replan tool is driven by the user who will interact with the dashboard. When the user decides that it is time to update a release plan, s/he will make the adaptations (e.g., include or remove a particular feature, redefine some dependency, etc.) and request a release plan. This will trigger the communication from the dashboard to the con-

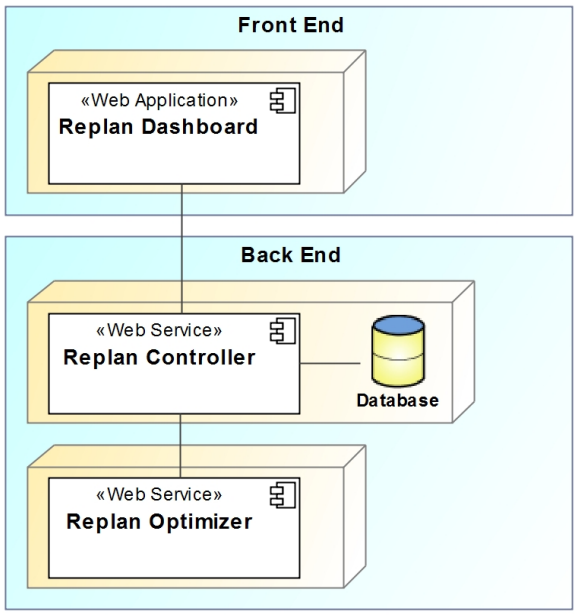

Fig. 2 Replan Architecture

troller, which in its turn, will ask the optimizer to produce the release plan.

Also, most of the CRUD operations available in the tool requires the dashboard to communicate with the controller. In addition, please note that all the persistence of the tool is managed by the controller.

In the remaining of this section, we present in more detail each one of the three components of Replan.

A. Replan Dashboard

The main purpose of this component is to provide a usable human interface in order to facilitate the access to the main functionality of the tool. This component also allows the user (e.g. product manager or developer team leader) to provide the additional information required by the tool (e.g., the resources available for each release). In order to minimize the amount and complexity of the input required to produce a release plan, we worked in several directions to produce a highly usable tool. Among the desired usability characteristics we remark the following:

- Gather features from different sources. Since the features can be obtained from sources such as feature prioritization tools, Replan users should not need to introduce them manually.
- Minimize the input required. Only the fundamental information required to execute the algorithm is requested before generating the release plan. Also, the user will be guided to introduce the missing information.
- Flexibility to make changes. We want to give freedom to the user to modify any information (e.g., ability to refine or adapt the features produced by external tools).

Figures 3-5 show some screenshots of the Replan Dashboard in order to illustrate the normal use of the tool when planning a software release.

Screenshot #1 (Fig. 3): the user sees the current candidate features for release (left) and the list of releases available for the project (right). From this screen, s/he can also create or modify new releases, and configure the resources of the project. To include a feature in a particular release, the user only needs to drag and drop the feature to the desired release. This action will trigger the release planning.

Once a feature is added into a release, the tool checks for completeness (e.g., by prompting the user to provide missing information) and compliance (e.g., by notifying that postponing a release is necessary to accommodate the newly added feature). For example, *Screenshot 2* (Fig. 4) shows the response of the tool when an added feature does not contain the effort required for implementation. These checks are fundamental to avoid or reduce user mistakes.

Screenshot 3 (Fig. 5) shows the representation of a release plan. It presents the resources and their allocation during the release timeline. The red dashed lines show the dependencies among tasks, and the red vertical line on the right shows the deadline of the release.

In order to create a new release, the user clicks the "+ Add release" button in the main screen (Screenshot #1, Fig. 3). This triggers a new screen, not shown here, where s/he can set a release name, description, and deadline. When creating a new release, the user should assign a set of resources (e.g. developers) to it.

Finally, there is a project resource management screen, not shown here, which can be accessed by clicking the "Edit project config." button in the main screen (Screenshot #1, Fig. 3). Here it is possible to add, edit, or remove project resources. For each resource, the user can specify the availability and a set of skills (which will be used for the assignment of tasks during the release planning)

The Replan Dashboard component has been implemented as a rich web application written in Java (server-side) and JavaScript (client-side). The server side relies on Spring Boot, while the client side uses the AngularJS framework. For the construction of the UI, considering the focus on usability and learnability, we used the popular JQwidgets and Bootstrap

Fig.3 Replan Dashboard Screenshot #1: Main page

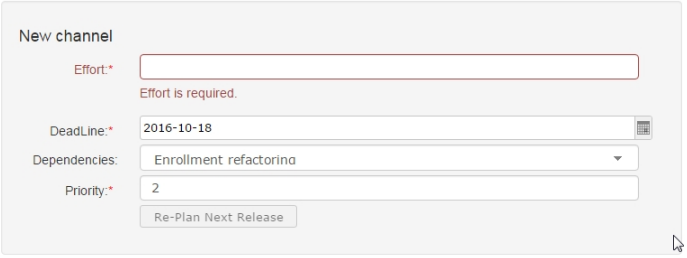

Fig.4 Replan Dashboard Screenshot #2: Feature Form

frameworks. This combination provides all the necessary elements for a modern UI design focused on quality of experience.

B. *Replan Controller*

This component is responsible for the management and storage of the internal representation of the domain knowledge, i.e. the information about features, releases, resources, release plans, etc. Therefore, this component provides all the information that the Replan Dashboard needs to show to its users and applies the requests that they make: feature assignments to releases, resource updates, etc. To do this, we implemented the Replan Controller component as a web service that exposes a well-defined REST API[2] to the Replan Dashboard.

As mentioned earlier, external tools can provide the Replan Controller with lists of prioritized features to be scheduled. For this reason, the Replan Controller exposes a dedicated and simple REST API[3].

When the Replan Controller receives a request to generate a release plan from a Dashboard user, it collects all the necessary information about the involved release, features, and resources (i.e. creates an instance of a *NextReleaseProblem*, according to Fig. 1) and sends it to the Replan Optimizer. The release plan that this latter returns (an instance of a *PlanningSolution* in Fig. 1) is then stored and sent back to the Replan Dashboard.

The Replan Controller component has been implemented with Ruby on Rails, a well-known and mature Model-View-Controller framework that provides a set of configuration defaults and built-in tools that allows API developers to setup and running quickly.

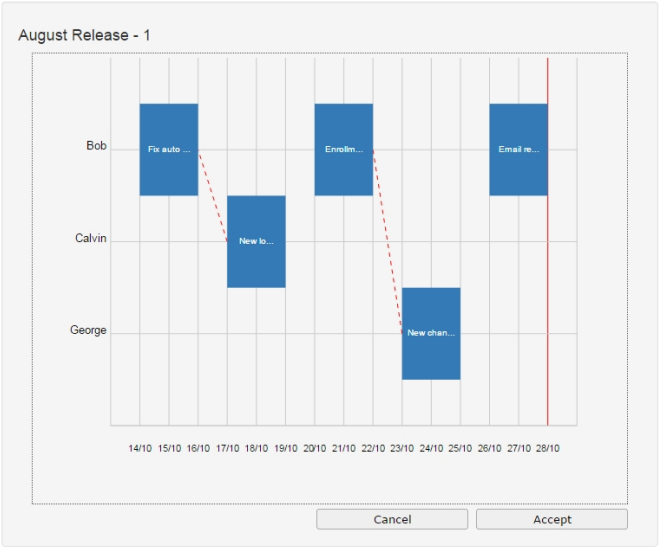

Fig.5 Replan Dashboard Screenshot #3: Release Plan Display

C. *Replan Optimizer*

This component has the sole purpose of generating a release plan. It has been developed as a stateless web service that given all the required information generates a release plan that, preserving the stated constraints, optimizes the use of the company resources to develop the next release. This web service is powered by our current implementation of the *NextReleaseProblem* solver, which relies on the application of state-of-the-art Genetic Algorithms. In particular, we use jMetal[4], a framework with an extensive portfolio of available multi-objective optimization algorithms: NSGA-II, MOCell, PESA-II, SPEA-II, etc. We refer to [6] for more details

In order to develop an optimizer service, the only requirement is to implement a web service that exposes the API[5] that the Replan Controller needs to consume. The input/output of the optimizer component is thus defined in the API specification.

The Replan Optimizer component has been implemented using Java-oriented technologies, Spring Boot in particular, because jMetal is only available in that language.

V. Preliminary Evaluation

We ran a preliminary assessment of the tool by submitting a mock-up version to three companies of different size and domain: two big enterprises and an SME (Small or Medium Enterprise), all very active in software development and offering very heterogeneous software products. The recipients (a set of fourteen product managers, project managers, developer team heads, and solution managers) were asked to inspect the mock-up in a one-hour session and fill out a questionnaire[6] measuring their perception of the tool.

The theme of the questionnaires was centered around the *perceived usefulness* and *perceived ease of use* of the tool, which have been demonstrated in the literature [7] to be largely correlated with one of the most important metrics for the success of a software product, i.e., the intention-to-use. In addition, the questionnaires were designed to capture a set of attributes defined in the ISO25010 quality in use model [8]. This allowed us to draw a conclusion on the perceived benefit of adopting Replan for release planning activities.

The results of the evaluation are reported in Table 1, aggregated for all participants from all three companies. The table reports the Mean Opinion Score (MOS), in a Likert scale from 1 (Strongly disagree) to 7 (Strongly agree), the Standard Deviation (SD), and the minimum and maximum rating given by all test users. For lack of space we cannot report the exact formu-

[2] https://supersede-project.github.io/replan/replan_controller/API-UI.html
[3] https://supersede-project.github.io/replan/replan_controller/API-WP3.html
[4] http://jmetal.sourceforge.net/
[5] https://supersede-project.github.io/replan/replan_optimizer/API-CTL.html
[6] https://supersede-project.github.io/replan/Replan_eval_questionnary.pdf

TABLE 1. AGGREGATED RESULTS OF THE TOOL EVALUATION

Statement	MOS	SD	Min	Max	Statement	MOS	SD	Min	Max
Usefulness ("Replan would…")									
…improve my effectiveness	4.9	1.2	3	6	…make my job easier	5.0	1.3	2	7
…increase my efficiency	5.1	1.1	3	7	…increase my productivity	5.1	1.0	4	7
…make me quicker	4.8	1.0	3	6	…be useful	5.7	1.0	4	7
Ease of use ("Replan is…")									
…easy to use	5.3	1.1	3	7	…controllable	5.5	1.2	4	7
…easy to learn	5.3	0.8	4	6	…flexible	4.4	1.0	2	6
Perceived relative benefits									
Financial benefits	4.7	0.8	4	6	Resource utilization	5.3	1.1	4	7
Reduced customer care	3.5	1.2	1	5	Transparency	5.6	1.2	3	7
Employee satisfaction	4.6	1.3	3	7	Organizational efficiency	5.1	1.1	3	7
Employee/manager ratio	4.9	1.2	2	7					
Others									
Satisfaction	4.6	1.5	1	7	Functional correctness	4.9	1.1	3	6
Trust	4.3	1.6	1	6	Functional completeness	4.3	1.2	2	6
Intention to adopt	4.7	1.7	1	7	Functional suitability	4.9	1.1	3	6

lation of the statements, but only a shortened version (for example, "Replan would improve my effectiveness" was formulated as "Replan would enable me to perform software evolution and maintenance more effectively").

The first and second groups of statements are related to the perceived usefulness and ease of use of Replan, respectively. We can see that all attributes scored above the mean of the rating scale. The tool is perceived to be useful to facilitate the release planning tasks and handle them with more effectiveness and efficiency. Replan is also perceived as easy to use, learn, and control.

The third group of statements lists a number of potential benefits produced by adopting the tool. The highest ratings are related to the capacity of Replan to improve transparency, resource utilization, and organizational efficiency. On the lower end, test users seem to slightly disagree with the fact that Replan reduces costs of customer care. This indicates that the tool was observed more from the perspective of ordinary software evolution, rather than maintenance.

The final group contains attributes that did not fit in the categories above. Also here, none of the ratings falls below the mean of the rating scale and the reception is generally positive. Test users feel satisfied with the tool. However, when asked whether they would trust the release plan suggested by the tool, the rating was close to neutral (MOS=4.3). This might be caused by the fact that some properties (like the speed and expertise of a developer) influence the release planning and task assignments but cannot be easily synthesized in a model/tool. This is confirmed by the functional completeness rating (again, MOS=4.3), which measures if the proposed approach covers all constraints to be considered in the release planning.

VI. CONCLUSIONS AND FUTURE WORK

Replan is a Software Release Planning tool that handles both strategic planning and operational planning in an integrated and flexible way. Replan has been architected in such a way that its components can be improved and scale up independently. At its core features a Next Realese Problem solver that is powered by state-of-the-art genetic algorithms.

Preliminary evaluation shows that Replan is perceived as convenient tool to perform release planning in an effective an efficient way, and, at the same time, as a tool that it is easy to use, learn, and control. Users also appreciate the ability of Replan to improve transparency, resource utilization, and organizational efficiency.

As an immediate future work, we will use and evaluate Replan in three real industrial use cases, in the context of the SUPERSEDE Project. Moreover, we will adapt and evolve Replan to support *Continuous* Software Release Planning in the terms described in [9].

REFERENCES

[1] G. Ruhe and M.O. Saliu, "The art and science of software release planning," *IEEE Software* 22(6), pp. 47-53, 2005.

[2] G. Ruhe, "Software release planning", in *Handbook of Software Engineering and Knowledge Engineering, Vol 3*, pp. 365-394, 2005.

[3] D. Ameller, C. Farré, X. Franch, and G. Rufian, "A Survey on Software Release Planning Models," in *17th International Conference on Product-Focused Software Process Improvement (PROFES)*, 2016, pp. 48–65.

[4] G. Ruhe, Product release planning: methods, tools and applications. CRC Press, 2010.

[5] R. Karp, "Reducibility among combinatorial problems," In *Complexity of computer computations*, pp. 85-103, 1972

[6] V. Elvassore, Experimenting with generic algorithms to resolve the next release problem. Master Thesis, Universitat Politècnica de Catalunya, 2016, http://hdl.handle.net/2117/89935.

[7] F.D. Davis, "Perceived usefulness, perceived ease of use, and user acceptance of information technology," *MIS Quarterly*, 13, 3, pp. 319-340, 1989.

[8] ISO/IEC 25010:2011, Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models.

[9] D. Ameller, C. Farré, X. Franch, D. Valerio, and A. Cassarino, "Towards Continuous Software Release Planning," in *24th IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2017