Towards Continuous Software Release Planning

David Ameller, Carles Farré and Xavier Franch
Universitat Politècnica de Catalunya
Barcelona, Spain
{dameller, farre, franch}@essi.upc.edu

Danilo Valerio and Antonino Cassarino
Siemens AG Österreich
Vienna, Austria
{danilo.valerio, antonino.cassarino}@siemens.com

***Abstract*—Continuous software engineering is a new trend that is gaining increasing attention of the research community in the last years. The main idea behind this trend is to tighten the connection between the software engineering lifecycle activities (e.g., development, planning, integration, testing, etc.). While the connection between development and integration (i.e., continuous integration) has been subject of research and is applied in industrial settings, the connection between other activities is still in a very early stage. We are contributing to this research topic by proposing our ideas towards connecting the software development and software release planning activities (i.e., continuous software release planning). In this paper we present our initial findings on this topic, how we envision to address the continuous software release planning, and a research agenda to fulfil our objectives.**

***Index Terms*—Software Release Planning, Continuous Software Engineering, Project Management.**

I. Introduction

Software engineering has always aimed to identify separate engineering activities and assign responsibilities to each of them as a way to have a clear path to produce or maintain software products. An extreme example is the seminal waterfall development process that required the current activity to be completely finished in order to start the following one. However, this process has been proved inadequate in most cases due to the instability of the software requirements [1]. To tackle this problem, several iterative software development processes emerged in the last fifteen years (e.g., Agile, XP, etc.), which still keep the activities apart but are organized into smaller iterations that facilitate the incorporation of changes in the initial activities.

Continuous software engineering is going one step further by establishing strong connections between software engineering activities. The objective of these connections is to accelerate and increase the efficiency of the software engineering process; for example, continuous integration is one of the best established continuous approaches in which the development and deployment activities are kept linked and synchronized automatically. In this sense, continuous software engineering can be also related to DevOps, *"a conceptual framework which aims at benefiting information systems development by reintegrating development and operations in various ways"* [2]. Keeping the connection between activities is not easy, therefore it is common to find tool support to provide some level of automation for continuous software engineering approaches. For example, Jenkins[1] is a very popular tool that supports continuous integration.

In this paper, our focus is on the connection between the software development and Software Release Planning (SRP) activities. Software development is the activity of producing the implementation of the software (e.g., writing the source code) while SRP is the activity of creating a release plan that indicates what is to be implemented, when and by whom during the time of a release [3].

The main idea of our approach is to re-plan the current release every time a developer activity triggers the need of an updated release plan (e.g., when some requirement implementation is finalized earlier or later than planned, when a developer takes a sick leave, etc.). To facilitate this re-planning we will analyse developers' activity in the ongoing release, either by information explicitly provided by them (e.g., through notifications) or indirectly (e.g., by analysing the data from the project management tools and software repositories).

With continuous SRP the release plan will become more adaptable and flexible in real-time, e.g., when some requirement is implemented very quickly and there is time to include yet another one in the current release, or once there is new information available about the development team that suggests the need of replanning the software release.

The rest of the paper is divided as follows: Section II introduces the necessary background concepts related to SRP; Section III presents our approach for continuous SRP and describes the envisioned process; Section IV reports the most relevant related work and how it links with our ideas; and finally, in Section V and VI we provide our research agenda and highlight the main conclusions.

II. Background

In the context of this paper, we refer to software development as the activity of implementing one or more requirements in some form of code. This includes creating tests and verifying the correctness of the implemented code as well as management tasks (e.g., the use of code version control systems, bug reporting tools, build tools, etc.).

On the other hand, the SRP activity is composed of two phases: *strategic planning*, the prioritization and/or selection of requirements to be included in the next release; and *operational planning*, the assignment of the requirements to a

[1] https://github.com/jenkinsci

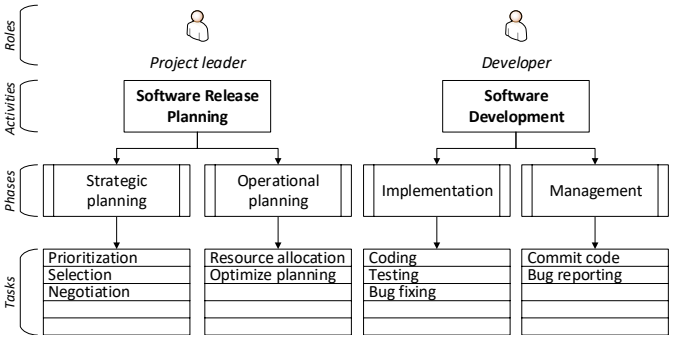

Fig. 1. Project leader and developer tasks.

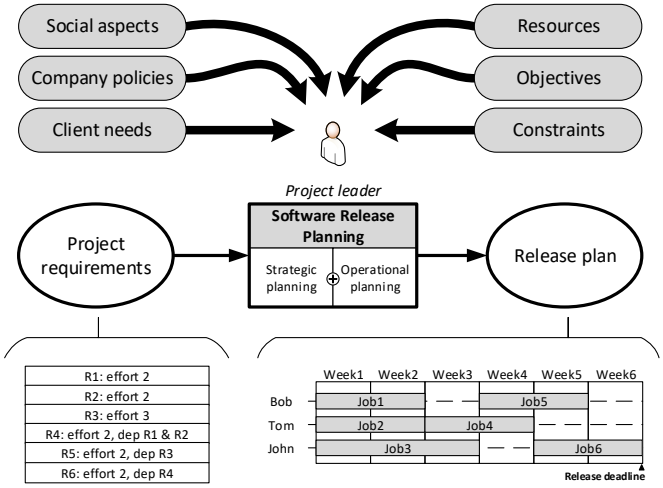

Fig. 2. Traditional SRP.

concrete team of developers [3]. Strategic planning is normally conceived for long-term planning (i.e., multi-release) while operational planning is more adequate for short-term planning (i.e., current release).

Figure 1 shows the conceptualization of the roles, activities, phases, and tasks related to SRP and software development.

In the traditional SRP, the project leader produces a release plan from a set of requirements. To carry out this activity, the project leader takes into account several aspects such as social aspects (e.g., who works well with whom), company policies (e.g., movement of the developers from project to project), client needs (e.g., change of priorities, budget, etc.), resources (e.g., the development team), objectives (e.g., priorities), and constraints (e.g., the deadline). Figure 2 shows this conceptualization of the traditional SRP.

In a previous work, we conducted a literature review about the available models related to SRP [4]. We found out that SRP has been formalized in the literature through the Next Release Problem (NRP) [5], but commonly it only covers the selection of requirements to be included in the next release or releases (i.e., strategic planning), while deciding when and by whom these requirements will be implemented (i.e., operational planning) is more similar to the Job Shop Problem (JSP) [6] (in essence, a scheduling problem). Both problems, NRP and JSP, are classified as NP-hard. The models found in our literature review are oriented to fulfil some objectives and constraints (see Table I and Table II) taking into account the available resources, but none of them handle the other aspects mentioned above. The objectives and constraints in Table I and Table II can be seen either from the strategic or the operational point of view (e.g., in both phases it is common to consider first high priority requirements and take into account the dependencies among the requirements).

TABLE I
EXAMPLES OF OPTIMIZATION OBJECTIVES

Name	Description
Maximize priority	Try to include the highest priority requirements
Minimize Cost	Try to reduce the use of expensive developers (e.g., specialists)
Maximize Revenue	Try to include the requirements that provide most revenue (independently of their priority)
Minimize Risks	Try to adapt the release plan in a way that minimizes a particular risks (e.g., reduce the possibility of a delay)

TABLE II
EXAMPLES OF CONSTRAINTS

Name	Description
Deadlines	There may be deadlines for specific requirements, and also for the whole release
Skills	Each requirement may need to be implemented by a developer with particular skills (e.g., expertise in a particular technology)
Hard Dependencies	The implementation of a particular requirement cannot be started before the dependent requirement is implemented
Budget	There may be a maximum budget that can be spent in the whole release

III. CONTINUOUS SOFTWARE RELEASE PLANNING

A. Definition and Scope

Continuous SRP is a specialization of the traditional SRP with the added capability of adapting the release plan in response to events that occur in the daily development activity.

Our approach tackles both strategic and operational planning. However, it is important to remark that operational planning is the most important phase in the continuous context, because development activities usually have a short-term impact in the release planning. For this reason, in this paper, we do not consider some of the long-term strategic planning tasks such as prioritization and negotiation as part of the continuous SRP approach.

Fig. 3. Continuous SRP process.

B. Description of the Continuous SRP process

The continuous SRP process (see Figure 3), as it happens in continuous integration, is an event driven process. For example, in continuous integration, the main trigger is when a developer pushes some commit to the version control system repository. In continuous SRP, the main trigger would be the finalization of the implementation of a requirement, however we can also consider other kind of events (see our catalogue of events in Section III-C). In continuous SRP these events will be detected and, in consequence, will trigger a re-planning that will produce an updated release plan.

A common factor for continuous approaches is the use of some kind of automation. To show this automation in continuous SRP we have refined the process depicted in Figure 3 to separate the automatic part from and the part controlled by the project leader. Figure 4 shows this refined process.

In the first part (Figure 4-left), the process starts, as before, with the detection of events that trigger the re-planning, but now instead of generating the updated release plan, we propose one or more release plans. Also, we added the possibility to warn about foreseen risks or issues (see Section III-D). In the second part (Figure 4-right), the project leader will have the opportunity to select the most appropriate release plan (from the proposed ones) and analyse the risks and issues (if any), and eventually, perform a manual adaptation of the release plan before approving the updated release plan.

We consider that the detection of events, the production of a tentative set of release plans, and the detection of issues and risks could be done automatically by a tool (or a set of tools) while the risk analysis, release plan review, and the adaptation still needs to be performed by the project leader. However, for the tasks in the second part, the ideal solution would be to have a support tool that, for example, highlights the changes between the previous and the proposed release plans and indicates which requirements are originating the risks or issues with a reasoned description.

There are important reasons for not having full automatic continuous SRP process: on the one hand, in general, full automatic processes are not trusted by the key stakeholders, e.g., project leaders normally want to keep the control on the decisions made; on the other hand, it would rise some ethical concerns to include in a tool knowledge about some of the aspects that need to be considered, for instance social aspects of the employees of a company. It is worth mentioning that other authors in the field of SRP also opted to have hybrid approaches (e.g., *"A hybrid release planning approach integrates the strength of computational intelligence and the knowledge and experience of human experts"* [7]).

C. Catalogue of Development Activity Events

We have identified some events related to the aspects mentioned in Section II, however we do not intend to provide an exhaustive list of possible triggers for a re-plan. In particular, we are considering the following development-time events in our approach:

Events related to the requirements:

- *Changes in the dependencies (to existing or new requirements).* The set of requirements included in a release may have unforeseen dependencies that forces the inclusion of a new requirement in the release (e.g., a developer starts the implementation of a requirement and realizes that a new requirement to support a particular technology is needed).
- *Changes in the effort.* The effort required to implement a requirement is an estimation and is subject to variations. The effort can be reduced (e.g., the developer finds that the requirement can be implemented very easily thanks to a library he just found) or extended (e.g., the estimation was too optimistic or there have been complications).

Events related to the resources:

- *Changes in the availability.* The planning is done based on a given availability of the resources, normally specified as the dedication ratio to the project for each resource. Lower granularity, e.g., a day by day availability, as far as we know is not being used by the algorithmic solutions. This availability may change for many reasons (e.g., sick leaves).
- *Changes in the skills.* Software developers are normally specialized in particular technologies or software parts (e.g., front-end developer and back-end developers).

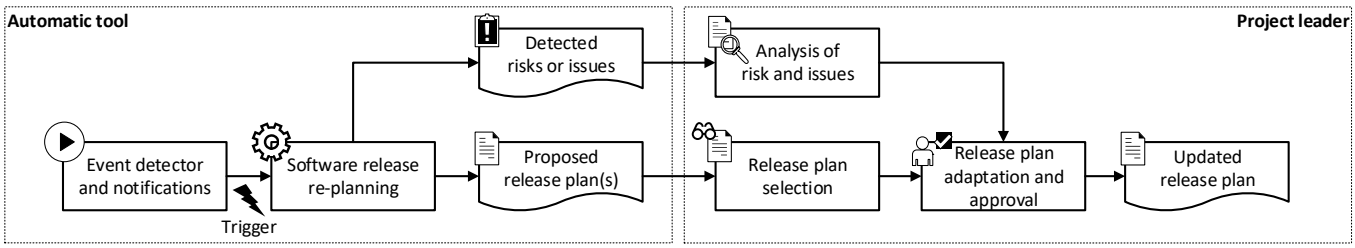

Fig. 4. Refined continuous SRP process.

However, developers are constantly renewing their knowledge by learning new technologies and languages, these new skills may be used to produce a better plan of the software release.

- *Changes in the cost.* The use of a particular resource may become more expensive, e.g., a developer is promoted in the company.

D. Risks and Issues

One of the characteristic of the proposed approach for continuous SRP is the possibility to warn about possible risks and issues much earlier in time compared to the traditional SRP (which only happens once in a while). We are considering the following risks and issues in our approach:

- *Risk of overrun.* This risk will emerge when, as consequence of the events occurred, the release plan may not finish on time (this can also be seen from the cost perspective). This risk can be either a high priority risk, i.e., the release plan already surpasses the release deadline as consequence of the changes; or as a risk indicator, i.e., taking into account the previous delays there is a certain probability of overrun.
- *Unfeasible solution issue.* The algorithm that generated the release plan may be unable to generate a solution that satisfies all the constraints. For example, a requirement needs a particular skill that none of the developers have, therefore the requirement cannot be scheduled and neither all its dependent requirements.

E. Example

To exemplify the continuous SRP approach presented in this paper, we use the same example shown in Figure 2 with the same dependencies. In Figure 5 we can see, at the top, the initial status; in the middle, the automatic part; and at the bottom, the project leader part. In the automatic part, we show some events that can occur during a continuous SRP and the consequences that they could imply as result of an automatic re-plan. Please note that in the Figure 5 we are exemplifying more than one event at the same time, but in continuous SRP each event would have had its impact at the moment it occurred.

Figure 5 shows that during the Week 3 two events happened: first, Job 1 is delayed, therefore the depending Job 4 (and Job 6 because it depends on Job 4) is moved forward; second, Job 3 finished earlier, therefore the depending Job 5 is moved backward. Then during the Week 5, another event happened: when John started to work on Job 6 (which was already delayed) he found out that a new job was necessary before Job 6. As result, the project leader has approved the proposed release plan, and by analysing the risk of release delay, s/he opted to change the release deadline to Week 7.

In the example we have not shown alternative release plans, but for instance, the new job (Job 7) could have been done by Tom or by John (see Figure 5), then the project leader would have selected the better release plan considering all the other aspects that are not included in the automatic part.

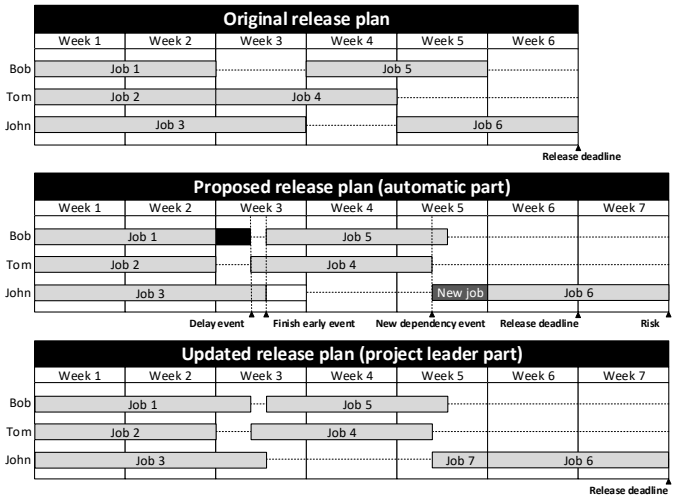

Fig. 5. Example of continuous SRP process.

Finally, we did not include the skills to keep the example simple and clear but, for example, Job 7 could have required a skill that only Bob has, therefore it would had added an additional delay to the release plan.

F. Open Issues for Automating the Continuous SRP Process

There are several open issues involved in the automatic part of the continuous SRP process:

- *Detection of events.* Clearly, the easiest way to manage these events would be to have manual notifications (a solution that may be suitable for a proof of concept) but at least some of these events should be detected in an automatic way, our idea is to integrate this approach with the tools that developers already use (e.g., Git, Trello, Slack, etc.) in order to reduce the overload during the daily work.
- *Re-planning.* From our literature review [4], only few approaches deal with re-planning (i.e., take into account the previous plan while generating a new plan). Re-planning is different from single planning because: a) when a re-plan occurs some parts of the planning may be already finished or being implemented, therefore they cannot be changed in the new plan; b) from the developer's point of view it would be chaotic if every day there are unnecessary changes in the release plan, therefore a specific re-plan optimization objective should be to maximize the correspondence with the previous release plan; and c) since the re-plan may occur several times a day, the execution time of a re-plan should be among seconds or minutes.
- *Detection of risks and issues.* While some risks and issues are quite easy to detect (e.g., breaching the deadline of the release), risk indicators (e.g., determine the probability of a delay) are more complicated and will require further research and the use of some technique such as machine learning.

IV. Related Work

Fitzgerald and Stol (available online in 2015) [8] published a roadmap and agenda for continuous software engineering. In this road map continuous planning is defined as an *"holistic endeavor involving multiple stakeholders from business and software functions whereby plans are dynamic open-ended artifacts that evolve in response to changes in the business environment, and thus involve a tighter integration between planning and execution"*, we mostly agree with this definition, except that our focus is put on the changes that occur in the development activity rather than in business environment. We think that our definition of continuous SRP (see Section III-A) is more aligned with the ideas behind other continuous approaches such as continuous integration.

Suomalainen et al. (2015) [9] did a preliminary study of how software companies perform (to some extent) continuous SRP, their work is based on a literature review and three case studies. The major difference is that their work is more oriented to strategic planning rather than operational planning.

Al-Emran et al. (2007) [10] presented a simulation-based approach to planning, re-planning and risk analysis of software releases on operational level. This work also uses an event-based approach for re-planning, the main difference is that this approach is not in the context of continuous software engineering.

V. Research Agenda

This work is based on the hypothesis that continuous SRP will improve the current situation. Some promising results were presented in [11] and [12] related to the use of tool support in SRP. However, the common approach in IT companies is still to decide the release plan based on the experience of the project leader. More empirical studies are needed to determine the real needs of the industry for continuous SRP.

The presented approach for continuous SRP, still has plenty of space for improvement: a) identify additional development-time events that can have an impact in the release planning; b) explore additional risks and issues that may emerge as result of the re-planning; c) provide feedback to developers (e.g., notify them when a change is made in the release plan).

Creating a proof of concept is a very important step in the research agenda. This includes resolving the open issues mentioned in Section III-F (i.e., detection mechanisms, adapt release planning for re-planning, and machine learning techniques). We already have a tool for release planning [13] and our plan is to adapt it to support continuous SRP.

Beyond the proof of concept, this tool can be also extended with other useful functionalities to improve its suitability and usability in the context of continuous SRP. For example, provide a graphical visualization of the evolution of the release plan would help the project leader to identify the roots of the issues. Also, the tool needs to be informative and easy to use to facilitate its adoption in real settings.

Finally, we need to validate our approach with experiments in an industrial environment. For this we are exploring the possibility of testing the presented ideas in some selected Siemens projects. Furthermore, we are in contact with several other companies that have expressed interest in our continuous SRP approach (e.g., SEnerCon[2], Agilogy[3], Getupcode[4]).

VI. Conclusion and Future Work

In this paper we have presented our approach for continuous SRP. We provided a definition of continuous SRP and described a process that implements this approach accompanied with an example. We also identified the open issues and compared our approach with some of the most relevant related works.

We also drafted a research agenda that points out our next steps for these ideas, we will start by creating a proof of concept and improving the proposed approach, and then validate it in an industrial setting.

Acknowledgment

This work is a result of the SUPERSEDE project, funded by the EU's H2020 Programme under the agreement number 644018.

References

[1] I. Sommerville, *Software Engineering*, 9th ed. Addison-Wesley, 2010.
[2] F. Erich, C. Amrit, and M. Daneva, "A mapping study on cooperation between information system development and operations," in *15th International Conference on Product-Focused Software Process Improvement (PROFES)*, 2014, pp. 277–280.
[3] G. Ruhe, *Handbook of Software Engineering and Knowledge Engineering, Vol 3: Recent Advances*. World Scientific Publishing, 2005, ch. Software release planning, pp. 365–394.
[4] 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.
[5] A. Bagnall, V. Rayward-Smith, and I. Whittley, "The next release problem," *Information and Software Technology*, vol. 43, no. 14, pp. 883–890, 2001.
[6] M. R. Garey, D. S. Johnson, and R. Sethi, "The Complexity of Flowshop and Jobshop Scheduling," *Mathematics of operations research*, vol. 1, no. 2, pp. 117–129, May 1976.
[7] G. Ruhe and M. O. Saliu, "The art and science of software release planning," *IEEE Software*, vol. 22, no. 6, pp. 47–53, Nov. 2005.
[8] B. Fitzgerald and K.-J. Stol, "Continuous software engineering: A roadmap and agenda," *Journal of Systems and Software*, vol. 123, pp. 176–189, 2017.
[9] T. Suomalainen, R. Kuusela, and M. Tihinen, "Continuous planning: an important aspect of agile and lean development," *International Journal of Agile Systems and Management*, vol. 8, no. 2, pp. 132–162, 2015.
[10] A. Al-Emran, D. Pfahl, and G. Ruhe, "DynaReP: A Discrete Event Simulation Model for Re-planning of Software Releases," in *1st International Conference on Software Process (ICSP)*, 2007, pp. 246–258.
[11] G. Du, J. McElroy, and G. Ruhe, "A Family of Empirical Studies to Compare Informal and Optimization-based Planning of Software Releases," in *5th ACM/IEEE International Symposium on Empirical Software Engineering (ISESE)*, 2006, pp. 212–221.
[12] M. Felderer, A. Beer, J. Ho, and G. Ruhe, "Industrial Evaluation of the Impact of Quality-driven Release Planning," in *8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2014, pp. 621–628.
[13] D. Ameller, C. Farré, X. Franch, D. Valerio, A. Cassarino, and V. Elvassore, "Replan: a Release Planning Tool," in *24th IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2017.

[2] http://www.senercon.de
[3] http://www.agilogy.com
[4] https://www.getupcode.com