Semi-Automatic Generation of Audience-Specific Release Notes

Sebastian Klepper
Technische Universität München
Munich, Germany
klepper@in.tum.de

Stephan Krusche
Technische Universität München
Munich, Germany
krusche@in.tum.de

Bernd Bruegge
Technische Universität München
Munich, Germany
bruegge@in.tum.de

ABSTRACT

Agile development methodologies encourage frequent releases. However, many releases can overwhelm clients, testers and users if they do not understand what the actual difference is between two versions. Every release raises questions like whether they need to update right away, whether functionality has been added or problems have been fixed and finally whether it is worth their time to try out a new version.

Release notes can alleviate this problem by informing their audience about the contents of a particular release, but the creation of high quality release notes takes time and effort. A release manager needs to have the release‘s target audience in mind, access information from project management, issue tracker, and build system and might even need input from designers and developers.

We describe a semi-automated approach for generating targeted, informative release notes. Our solution is designed from the point of view of a release manager who acts as an editor of auto-generated content based on information gathered from both build server and issue tracker. Furthermore, it allows release notes to be tailored to a specific audience depending on their specific information needs.

CCS Concepts

•**Software and its engineering** → **Agile software development; Software post-development issues;** *Software development process management; Software evolution;* Software configuration management and version control systems; Acceptance testing; Software version control; Programming teams;

Keywords

Agile Development, Continuous Delivery, Release Management, Automated Workflows

To appear in the International Workshop on Continuous Software Evolution and Delivery - CSED, May 2016, Austin, TX, USA.

1. INTRODUCTION

Continuous delivery promises improved product quality and high customer satisfaction through regular releases [4, 2]. However, both issuing and receiving releases in high volume can be a communication challenge for the development team and users, respectively. One solution to this problem are release notes that summarize changes contained in a particular release. Such release notes can serve a multitude of purposes, depending on the domain they are used in. For example, they can improve user experience and serve as a marketing channel in end user software, educate professional users of specialized or B2B applications about new functionality, provide status updates in client-contractor projects, and elicit feedback from recipients.

We therefore consider different target groups for release notes that might exist in a given software project as shown in Figure 1. Everyone in a software project can be a stakeholder and assume one or even multiple roles, each with different needs to be addressed with release notes, e.g.:

- An internal tester receives unfinished versions and needs to know which features to test and which to ignore.
- Both project manager and customer want to know how the project is going, but with different levels of detail.
- A lead user is waiting for a critical issue to be fixed.
- Beta testers give feedback on pre-release features.
- New features are advertised to existing users.

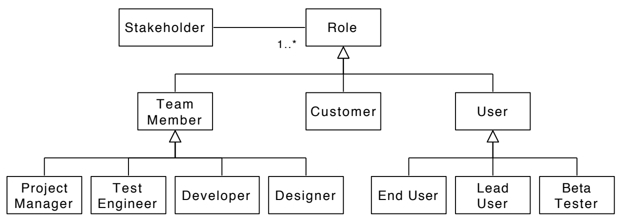


Figure 1: Taxonomy of stakeholder roles in software projects

A recent trend with mobile applications delivered through a vendor's app store is to update them every couple of weeks, but without any release notes[1]. While automatic updates in the background might lead to release notes not being read before the update, missing information about updates leaves users in the dark.

[1]http://techcrunch.com/2015/09/04/app-release-notes-are-getting-stupid

A major problem standing of proper release notes being attached to every update is the amount of time it requires. Creating understandable release notes that exactly match the target group's needs takes effort and might require information from multiple sources [8]. In previous publications [5, 6], we introduced a continuous delivery approach where event-based releases are created from different branches in the version control system as shown in Figure 2.

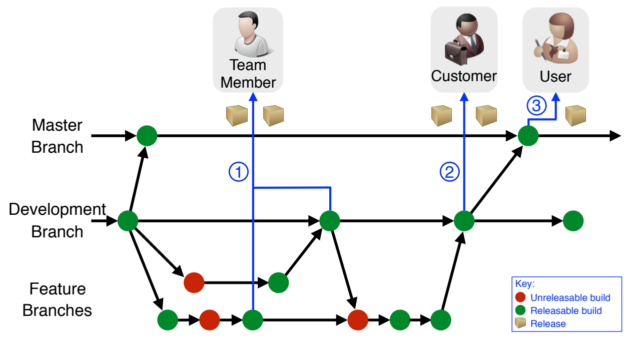


Figure 2: Event based delivery for different stakeholders from different branches (adapted from [6])

Features are developed on feature branches and integrated into the development branch once completed. External releases are merged into the master branch and usually sent to all stakeholders including users. Releases from development branches can be sent to the customer to validate if features were implemented correctly. Team members can receive releases from all branches, e.g. to discuss status and impediments in team meetings.

In this paper, we describe a semi-automated approach for the generation of release notes that is integrated into the event based release mechanism. Our solution is designed from the point of view of a release manager who should not act as a copywriter but rather an editor of pre-generated release notes. In a continuous delivery environment, we can use automated infrastructure to save time collecting and generating release notes. Each release can be treated individually, providing an opportunity for tailoring release notes content. We define release notes and describe related work in Section 2. We present our approach in Section 3 and provide an overview of future work in Section 4.

2. BACKGROUND AND RELATED WORK

Release notes are part of software configuration management [7] and can be defined as documents produced by the development team to summarize the main changes between two releases [8]. They usually contain the release date and version of the product as well as a summary of the release and a list of additions, removals, changes, or fixes included.

Release notes generally inform the recipient of an application update about its contents. This can take different forms, depending on the audience of a particular release: A team member is instructed how to test functionality that's currently in development; a customer is informed about the state of changes and improvements they have requested; an end user is prompted to try out a new feature and assured that known problems have been fixed. Release notes provide a concise overview of the changes a new release introduces and how to take advantage of them. In doing so they improve both, the recipient's user experience and serve as an important communications channel for the development team.

Moreno and his colleagues propose an approach that identifies changes in the commits between two releases of a software project, such as structural changes, upgrades of external libraries, license changes [8]. They summarize the code changes and link it to information from commit notes and issue trackers. Then they organize the release notes into categories and present them in an HTML document. Our approach differs because it is based on a continuous delivery workflow, where releases are created on a regular basis for different target groups, and takes the target audience into account for the creation of release notes. The release manager is responsible for triggering a release and has the authority about what makes it into the release [3]. Based on this role, we enable automatic generation of release notes *content*, but keep the option to manually review and curate them.

The reasoning is that manual curation allows the release manager to tailor the release notes to the audience of a particular release, which is another important requirement for our approach. We postulate the hypothesis that **audience-specific release notes provide exactly the information their recipients are looking for**. As a consequence, we restrict release notes to only relevant content in the context of their target audience. Information is filtered, grouped and sorted so that the recipients' information needs are optimally fulfilled. Data sources are selected to provide exactly the information needed to realize this requirement – in contrast to other approaches where release notes might simply contain all information available, regardless of whether that information is insufficient or even unnecessary.

3. APPROACH

We use a semi-automatic approach to release notes generation to reduce workload for the development team and release manager while still allowing them to provide properly targeted content depending on the context and recipients of a release. We enable this by integrating with tools along the continuous delivery model [4]. The information that is already produced during the development process is used to prepare release notes content. Subsequently, we allow release managers to individualize this content either via configuration options or as part of the release process. Release notes generated this way will then be delivered alongside the actual application to intended recipients.

3.1 Requirements

We identify the following **functional requirements** for a potential solution:

- Automatically gather all relevant information about a release, e.g. by integrating with the build server, issue tracker, or version control system.
- Automatically compile a list of changes to be published in the release notes.
- Allow a release manager to review and edit this list of changes before publishing.
- Allow a release manager to add custom notes such as a summary of the release.
- Allow to filter the auto-generated information based on the target audience of the release.
- Allow to restrict the information density so that it's appropriate to the target audience.

Content	Audience			Data source
	User	Customer	Team member	
Release description	✓	✓	✓	Input by release manager
New functionality	✓	✓	✓	Issue tracker
Fixed issues	✓	✓	✓	Issue tracker
Other changes	✓	✓	✓	Issue tracker
Known issues		✓	✓	Issue tracker
Technical information			✓	Version control system, build server
Testing instructions			✓	Input by release manager

Table 1: Sample mapping of release notes contents to audiences and data sources.

Additionally, we identify **non-functional requirements**: Generating release notes must integrate seamlessly into an existing release workflow. The generator must be easy to install and configure. Using the generator during the release process must be easy and intuitive.

3.2 Targeting

As potential recipients of release notes can vary from project to project, a generalized role model helps to differentiate their expectations of a release notes document: *team members* such as developers or test engineers, regular *users* of the application, and the project's *customers*. The latter group may also include other internal or external stakeholders such as business units or management.

These roles might have very different information needs when reading release notes for a particular version. Regular users might only be interested in new functionality and fixed problems, whereas a client also wants to know about existing but known issues and other changes. Testers within the team are interested in all of that but also expect instructions for testing as well as technical information such as the current branch and commit of the app in their hands.

A release manager therefore defines a mapping of available information to the audience of a particular type of release. A good way of targeting these different types of releases is to use a suitable branching model. In the approach of our choice, users would only receive releases from the *master* branch whereas customers might also receive pre-release versions from the *development* branch. Team members can test versions from everywhere including *feature* branches.

By integrating release notes generation as build step, we enable different settings for each type of branch and include the branch name a release is based on. Based on a sample mapping fulfilling the requirements of our iOS project course [1], we identify each audience's information needs and suitable data sources. Table 1 shows such a mapping of potential release content to audiences as well as utilized data sources.

3.3 Workflow

Figure 3 depicts the basic workflow of our solution as part of the delivery model of a continuous delivery project:

1. Release notes generator is executed as build step.
2. Generator fetches relevant information from data sources.
3. Generator generates release notes as a build artifact.
4. Release manager reviews and revises release notes.
5. Release notes are delivered along with the app.

A central element to this workflow is that the build server knows which issues are associated with a particular build. This information can either be provided manually when starting a build or developers can use so-called "smart commits" with messages that contain issue keys to allow automatic detection of relevant issues. To support automated issue linking and make things easier for developers, we provide a script that integrates with Git's *commit-msg* hook: Issue numbers contained in a branch name automatically appended to a commit message. Whenever a build runs, release notes generation is triggered as a build step and release notes are attached as a build artifact:

1. Configuration is retrieved, options can either be set statically (e.g. settings file) or dynamically for each build (e.g. environment variables or build server UI).
2. Generator fetches data from respective source systems, e.g. build server, issue tracker, version control.
3. Data is filtered, sorted and grouped based on the configuration, during either API calls or post-processing.
4. Release notes are generated in a desired target format, e.g. plain text, Markdown, or HTML.

Using static or dynamic configuration, a release manager has precise control over release notes content, level of detail, and format. Carefully selecting information for each target audience is particularly important to avoid a "garbage in, garbage out" effect. Tickets can be selected or ignored, based on type, status, or other available attributes. Similarly, tickets can be sorted and grouped, even differently for each ticket type. Data to display can then be selected based on the audience's needs, e.g. with technical or non-technical focus. For example, useful configuration options could be: Group user stories by epic, ignore user-generated bug reports as well as those flagged "wont't fix", include general improvement tickets but exclude technical tasks, use a custom field instead of ticket title if provided, etc.

Figure 4 shows how different audiences can be addressed using the same data, ranging from technical details for team members (4a) over a styled list for a status report (4b) to a casual description for the app store (4c). Our approach provides release managers with flexibility and can turn the same data set into different forms of release notes, further reducing the need for manual revision. It is important to note that content does not have to be restricted to a single build. The same generator can as well produce a change log across multiple builds, versions, or sprints.

4. CONCLUSION AND FUTURE WORK

In this paper, we proposed a semi-automatic approach for the generation of audience specific release notes for event based releases. We want to evaluate the approach in a case study in a large capstone course in university, refine it upon the evaluation results and then introduce it into industry.

As a proof of concept, we have implemented a working albeit not full-fledged solution: Our generator script can be

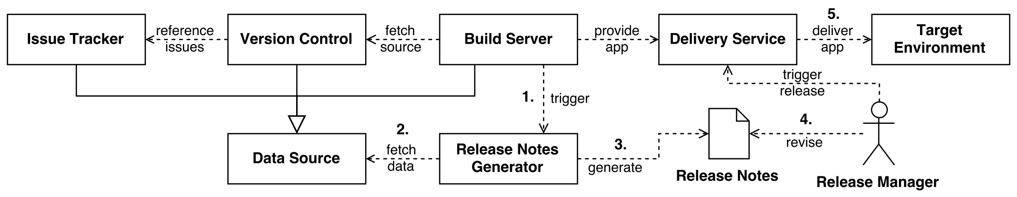
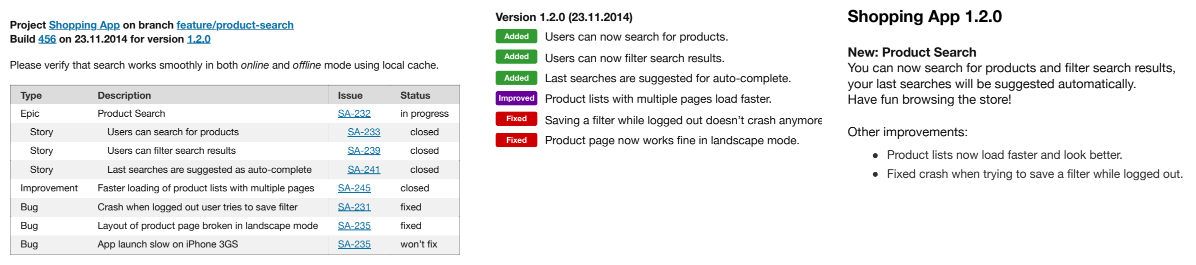


Figure 3: Semi-automatic release notes generation as part of a continuous delivery workflow.



Project Shopping App on branch feature/product-search
Build 456 on 23.11.2014 for version 1.2.0

Please verify that search works smoothly in both *online* and *offline* mode using local cache.

Type	Description	Issue	Status
Epic	Product Search	SA-232	in progress
Story	Users can search for products	SA-233	closed
Story	Users can filter search results	SA-239	closed
Story	Last searches are suggested as auto-complete	SA-241	closed
Improvement	Faster loading of product lists with multiple pages	SA-245	closed
Bug	Crash when logged out user tries to save filter	SA-231	fixed
Bug	Layout of product page broken in landscape mode	SA-235	fixed
Bug	App launch slow on iPhone 3GS	SA-235	won’t fix

Version 1.2.0 (23.11.2014)

Added	Users can now search for products.
Added	Users can now filter search results.
Added	Last searches are suggested for auto-complete.
Improved	Product lists with multiple pages load faster.
Fixed	Saving a filter while logged out doesn’t crash anymore
Fixed	Product page now works fine in landscape mode.

Shopping App 1.2.0

New: Product Search
You can now search for products and filter search results,
your last searches will be suggested automatically.
Have fun browsing the store!

Other improvements:

- Product lists now load faster and look better.
- Fixed crash when trying to save a filter while logged out.

(a) Technical details for team members. (b) Styled list for status report. (c) Casual description for app store.

Figure 4: Examples for release notes based on the same data but tailored to different target audiences.

configured via settings files or build variables. It is run as a build step and uses Bamboo’s and JIRA’s API to generate a markdown file containing all changes in the current sprint. This file is then uploaded to HockeyApp, our delivery service, along with the app and rendered to HTML before being displayed to users. Potential future extension of our solution hinges on several interesting evaluation questions.

Information needs: Which specific questions does each target audience have in mind when examining a release? Are these questions answered sufficiently by our release notes? What other requirements or restrictions apply when publishing release notes? An appropriate solution might also require involving other stakeholders such as marketing, editorial or legal staff at the right time.

Workflow integration: How much effort is caused by this way of preparing release notes? If it affects the team’s productivity, would better integration with existing tooling improve efficiency? Possible improvements are integrating with other CI servers and issue tracking systems. Proper UI integration, e.g. as a CI server plugin, might bring additional benefits with regards to ease of use and productivity.

Content generation: Is it sufficient to use information that is already produced during the development process or should release notes content be prepared separately? How does time and place of delivery affect requirements to release notes format? We could further improve our method of customizing content and add other data sources. Utilizing custom fields of the issue tracker or knowledge management systems, e.g. Confluence, would allow release managers to capture information ahead of time.

Feedback cycle: Do release notes enable users to give feedback more quickly? Do release notes improve the quality of feedback the team receives? This is an important research field as stated by Rodríguez and her colleagues: “a clear research gap exists for mechanisms to use customer feedback in the most appropriate way so that information can be quickly interpreted” [9]. To track this methodically, we would utilize feedback management as e.g. offered by HockeyApp.

5. REFERENCES

[1] B. Bruegge, S. Krusche, and L. Alperowitz. Software engineering project courses with industrial clients. *ACM Transactions on Computing Education*, 15(4):17:1–17:31, 2015.

[2] L. Chen. Continuous delivery: Huge benefits, but challenges too. *Software, IEEE*, 32(2):50–54, 2015.

[3] J. Erenkrantz. Release management within open source projects. *Proceedings of the 3rd Open Source Software Development Workshop*, pages 51–55, 2003.

[4] J. Humble and D. Farley. *Continuous delivery: reliable software releases through build, test, and deployment automation.* Pearson, 2010.

[5] S. Krusche and L. Alperowitz. Introduction of Continuous Delivery in Multi-Customer Project Courses. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 335–343. IEEE, 2014.

[6] S. Krusche, L. Alperowitz, B. Bruegge, and M. Wagner. Rugby: An agile process model based on continuous delivery. In *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*, pages 42–50. ACM, 2014.

[7] M. Moreira. *Software configuration management implementation roadmap*, volume 1. John Wiley & Sons, 2004.

[8] L. Moreno, G. Bavota, M. Di Penta, R. Oliveto, A. Marcus, and G. Canfora. Automatic generation of release notes. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 484–495. ACM, 2014.

[9] P. Rodríguez, A. Haghighatkhah, L. E. Lwakatare, S. Teppola, T. Suomalainen, J. Eskeli, T. Karvonen, P. Kuvaja, J. M. Verner, and M. Oivo. Continuous deployment of software intensive products and services: A systematic mapping study. *Journal of Systems and Software*, 2016.