Release Engineering Practices and Pitfalls

Hyrum K. Wright and Dewayne E. Perry
Dept. of Electrical and Computer Engineering
The University of Texas at Austin
Austin, Texas
hyrum_wright@mail.utexas.edu, perry@ece.utexas.edu

***Abstract*—The release and deployment phase of the software development process is often overlooked as part of broader software engineering research. In this paper, we discuss early results from a set of multiple semi-structured interviews with practicing release engineers. Subjects for the interviews are drawn from a number of different commercial software development organizations, and our interviews focus on why release process faults and failures occur, how organizations recover from them, and how they can be predicted, avoided or prevented in the future. Along the way, the interviews provide insight into the state of release engineering today, and interesting relationships between software architecture and release processes.**

***Keywords*-release engineering; software process**

I. Introduction

Release engineering is one of the least studied areas in software engineering, yet to be useful all software projects must create releases. As software systems and their development processes grow more complex, so do release processes. Release processes are as diverse as the software projects they support, from small collections of related tools, to large complex software systems. Fundamentally, though, the experience of releasing software is common to all software development organizations.

Release processes are not perfect, however, and understanding release engineering processes, their faults and failures, can be especially important for a quality software engineering experience. Release teams often operate under great external pressure, and failure during the release process can have a large impact on project success. Understanding how these teams operate, what causes their activities to break down, and how they recover from them will help future practitioners avoid the same problems.

This paper describes our early results after performing a series of semi-structured interviews with several practicing release engineers. The software these individuals work on, and the release processes they use vary extensively, but the interviews have given us insight into what influences release process design. Additionally, the interviews have looked at failures in release processes, and how those failures can be avoided in the future.

II. Background

Release engineering has not historically received very much attention in the software engineering research community. There have been piecemeal efforts to describe various aspects of release, but nothing holistic. Even the definition of "release engineering" is not very well articulated. This section outlines our working definition for release engineering and processes, as well as selected prior work in the area.

This work is not the first effort to examine project release processes, but does have a much wider scope than previous studies. Both Erenkrantz and Wright examined the release processes of specific software projects [1], [2]. Other work attempted to mine the version control system and bug tracker for a single project to develop a release history for that project [3].

There have been several quantitative studies on optimal software release timing [4], [5], [6], but these usually use various models of fault density and development rates which in practice can exhibit a high amount of variance. Additionally, these techniques require as input development and fault models which are difficult to develop and rarely readily available. Process discovery techniques also require inputs, such as event streams [7], but unfortunately, such streams are largely unpublished—or simply nonexistent—for release processes.

In contrast, our work takes a more qualitative approach by interviewing the actual people involved with software releases in practice. We feel this lends the necessary "mundane realism" to our results which may be lacking in more generalized quantitative approaches.

In addition to practical experience managing releases for widely-used software projects, some of our own previous work in this area involved the attempt to generalize a solution to automatically collect release history information from software repositories [8]. That method proved to require a large degree of individual and specialized attention for each project, leading us to believe that the more qualitative approach discussed in this paper is a more practical next step.

Other empirical work has looked at feature-driven development and the impact technical and organizational factors have on integration failures [9]. In such an environment,

978-1-4673-1067-3/12/$31.00 © 2012 IEEE

ICSE 2012, Zurich, Switzerland
New Ideas and Emerging Results

Table I
INTERVIEW SUBJECTS

Label	Software Type
A	Hosted web application
B	Network router control software
C	Behind-the-firewall web application
D	Open source tools distribution

integration is an important part of the release engineering process, and this work supports our own.

In addition, many of these projects only record a single aspect of the release process, such as its history or fault levels. The work described in this paper indicates that release processes are much richer than can be captured via automatic means or quantitative analysis methods. Such techniques are useful, but process discovery often requires interaction with participants, such as we describe here. As such, our work paints a more holistic picture of the release process and its potential pitfalls.

III. METHODOLOGY

Our initial work focuses on a set of semi-structured interviews with practicing release engineers at a variety of software development companies. These initial results come from four such interviews, though we have performed more and are in the midst of analyzing them. Due to the open-ended nature of our work, we chose to use a grounded theory approach, and will continue to interview until we feel we have reached an appropriate level of saturation [10]. The subjects were recruited by soliciting contacts from the professional networks of the researchers, as well as sending inquiries to release engineering-based industry groups.

From this pool of people, we scheduled one-hour interviews, which were conducted in person or via phone and recorded. Prior to the interviews, we provided each subject with a brief questionnaire to seed the interview process. A list of selected interview subjects, as well as the type of software they are responsible for releasing are shown in Table I. When possible we attempted to interview multiple individuals from the same software development organization. In many cases, however, this proved impractical, often because there was only one individual in an organization who managed the release process.

These interviews are currently ongoing, but our initial analysis highlights several themes that are common to release processes and their failures. These include relationships between release processes and software architecture, social causes of release problems, and the relationship that tools and software domain have on release process design.

A. Data sources

Software engineering researchers tend to over-emphasize open source projects and data when doing empirical research [11]. In order to avoid this problem, we attempted to ensure a large set of subjects on proprietary software development organizations were included in our pool. While this technique presents some logistical challenges, we feel it better portrays the state of software development today.

To reach as many potential release engineers as possible, we sent requests to various software development mailing lists, and used our own networks of professional connections to find individuals to interview. This method obviously suffers from various kinds of biases, such as self-selection, but we feel it gives sufficient variety to lend validity to the results.

Our initial group of interview subjects spans a range of software domains and development methods, from small "agile" teams which release frequently to large organizations which only create occasional release artifacts. Similarly, the artifact distribution models ranged from shipping to internal corporate customers in a controlled hosted environment, to sending a hard disk with 80GB of software updates with a technician to a customer site.

Each of our subjects fell into one of two self-identifying categories: a dedicated release engineer whose primary responsibilities were on release and deployment; or a member of a development team who was also responsible for that team's release activities. Insights from both groups were useful, with many common themes present.

IV. DISCUSSION

Our analysis of the interviews performed presents several ideas that we feel are interesting and ripe for future research. These include the relationship between software architecture and release processes, the risks and rewards of various release strategies, the social problems associated with producing an effective release process and the relationship between the type of software being shipped and the release process.

A. Software Architecture and Release

Many of the organizations and individuals we spoke with described their software release as being heavily tied to the architecture of the system. This relationship is analogous to the long held belief that communications structures often mirror the organizations which generate them, also known as Conway's Law [12].

Systems that exhibited a large, tightly-coupled design and code base often require large and complex release processes, since many modules of the code are interdependent. This interdependence practically dictates that release processes are highly coordinated affairs, requiring large amounts of effort and involving many individuals, and only occurring with low frequency. They can also be high-risk efforts, as a failed release results in the sunk cost of a large amount of resources.

In contrast, a smaller, more modular software system with fewer interdependencies often requires less effort to release,

leading to a more frequent releases, and individual releases of various subcomponents of the entire system. This often resembles the agile software development methodology, which encourages modular features and incremental releases [13]. Anecdotal evidence of this phenomenon is observable in open source ecosystems where the mantra "release early, release often" historically correlates with small, modular software components and rapid release cycles [14].

Loosely coupled systems are not without their own risks during release, however, as compatibility interfaces must be maintained and supported for use by older dependent components. The design of this interfaces, as well as their continued maintenance can introduce significant overhead in the development process, even though it may make releases easier to perform.

B. Social Causes of Release Failure

While the release process can be highly dependent upon the architecture of the software being released, failures in the release process are rarely due to the software itself. Instead, failures can usually be attributed to configuration issues, lack of communication, poor infrastructure or social problems within the releasing organization.

Many of the individuals interviewed cited social issues as the root of their release engineering failures. Release teams can vary from being tightly integrated with developers, to operating as completely separate organizational units and on disparate schedules. For those who are disjoint from the developers, it becomes difficult for them to easily coordinate releases with the developers, which adds additional friction and potential for failure in the release process.

Often, release managers trying to improve this problem are rebuffed by their superiors who see little business sense in improving the project's ability to properly manage releases. One subject noted the lack of support and funding to improve the release infrastructure as a serious detriment to her ability to deliver proper releases.

These social problems may also exist in open source communities, where participants self-select tasks, and may stop trying to improve release management beyond some local optimum [15]. Each member of the team is satisfied with the state of the release process, and has little incentive to improve it, even though such improvements may be generally useful to the project.

C. Software Domain and Tools

Other aspects of a software project which affects its release process are the domain of the software, including its usage model, and the development tools used to build and track the release environments and artifacts.

Several of the subjects we talked to produced software for in-house customers or deployment to servers controlled by their organization. Shipping to an in-house organization often means developers can release multiple times in a short period, even several times a day. This requires a low-overhead release process, but also allows for occasional process failures, as restarting the process is not a high-cost activity.

In contrast, software with a high amount of friction in the distribution mechanism tends to be more carefully tested before it is shipped. One subject we interviewed worked for a company whose software was over 80 GB in size, and whose distribution model involved copying the software to a hard disk and sending that disk to a customer site with a technician to assist in the installation. In this scenario, shipping faulty software can require expensive measures to correct, so the release process is much more controlled.

The tools release managers use also impact their process and ability to recover from process problems. Issue trackers and version control systems, continuous integration and deployment systems all play a role in helping to create a release. The ease of use of these tools, and the extent to which they are put to use, can impact the quality and timeliness of the release product.

One release engineer had recently switched version control systems and in doing so indicated that the switch allowed the team to better control feature development and release. By improving the tools the release engineer used, she was able to deliver more targeted releases in a more effective manner, ultimately improving the quality of the software produced.

The foregoing themes all present an interesting discussion of some of the variables that affect release processes, but we have yet to discover methods of preventing release process failures. We anticipate that as our studies proceed with transcription and coding, additional insights will emerge as to the ways that organizations avoid process problems.

V. Future Research Directions

Based up on the preliminary work described above, we envision a future research direction including a number of related questions. These include the use of formal process analysis to better understand release process structure and efforts to assist in process standardization.

A. Formal Process Analysis

Our work has primarily focused on *qualitative* measures of release processes, and our early results demonstrate that release processes vary widely across organizations. Our interviews have indicated a need for more formal *quantitative* methods for reviewing, comparing, and analyzing release processes. Process modeling languages, such as Little-JIL [16] or Interact [17], may prove useful to aid in reasoning about process interactions and properties [18]. However, due to release process complexity, capturing a complete release process, with its many exceptions, may require significant resources.

Release process modeling and analysis would also benefit from an understanding of where release engineering fits in the comprehensive software development cycle [19]. Such process unification would be beneficial to both release engineers, as well as those who look to integrate their efforts into a wider development process.

B. Process Standardization

As release processes continue to evolve and change, some degree of standardization among them will occur. Future research into release processes can assist such standardization before bad practices become entrenched and difficult to dislodge.

Process standardization could also help to encourage a set of best practices for the release engineering industry. Such knowledge is currently buried within organizations, with very little ability for discussion and learning across release groups. Developing a repository for release process information could help these groups better communicate and standardize their processes.

VI. Conclusion

This paper presents some of our early research ideas surrounding release processes, their faults and failures. We discuss an set of semi-structured interviews with practicing software release engineerings from a number of difference commercial software development organizations, and some of the early themes detected in these interviews. Some of these themes include the relationship between software architecture and release process, the impact of tools on release processes, and social issues surrounding release failures.

While specific common process failure modes are not yet clearly evident, the early results from these interviews provide some interesting direction as to future research areas. These include more thorough process analysis and efforts at process standardization. We hope our continued research can address these issues, and ultimately improve the state of the practice of release engineering.

References

[1] J. R. Erenkrantz, "Release Management Within Open Source Projects," in *Proceedings of the ICSE 3rd Workshop on Open Source Software Engineering*, May 2003.

[2] H. K. Wright and D. E. Perry, "Subversion 1.5: A Case Study in Open Source Release Mismanagement," in *Proceedings of the ICSE 2nd Emerging Trends in FLOSS Research and Development Workshop*, May 2009.

[3] M. Fischer, M. Pinzger, and H. Gall, "Populating a release history database from version control and bug tracking systems," in *ICSM '03: Proceedings of the International Conference on Software Maintenance*. Washington, DC, USA: IEEE Computer Society, 2003, p. 23.

[4] H. Koch and P. Kubat, "Optimal release time of computer software," *Software Engineering, IEEE Transactions on*, no. 3, pp. 323–327, 2006.

[5] Z. Jiang and S. Sarkar, "Optimal Software Release Time with Patching Considered," in *Proceedings of 13th Annual Workshop on Information Technologies and Systems December*, 2003, pp. 13–14.

[6] K. D. Levin and O. Yadid, "Optimal release time of improved versions of software packages," *Information and Software Technology*, vol. 32, no. 1, pp. 65–70, 1990.

[7] J. E. Cook and A. L. Wolf, "Discovering models of software processes from event-based data," *ACM Trans. Softw. Eng. Methodol.*, vol. 7, pp. 215–249, July 1998. [Online]. Available: http://doi.acm.org/10.1145/287000.287001

[8] J. Tsay, H. Wright, and D. Perry, "Experiences mining open source release histories," in *International Conference on Software and Systems Process (ICSSP 2011)*, 05/2011 2011.

[9] M. Cataldo and J. Herbsleb, "Factors leading to integration failures in global feature-oriented development: an empirical analysis," in *Proceeding of the 33rd international conference on Software engineering*. ACM, 2011, pp. 161–170.

[10] B. Glaser, *Doing grounded theory: Issues and discussions*. Sociology Press Mill Valley, CA, 1998, vol. 254.

[11] H. K. Wright, M. Kim, and D. E. Perry, "Validity Concerns in Software Engineering Research," in *Proceedings of the Workshop on the Future of Software Engineering Research*, November 2010.

[12] M. Conway, "How do committees invent?" *Datamation*, vol. 14, no. 4, pp. 28–31, 1968.

[13] J. Highsmith, *Agile software development ecosystems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[14] E. Raymond, "The Cathedral and the Bazaar," *Knowledge, Technology, and Policy*, vol. 12, no. 3, pp. 23–49, 1999.

[15] M. Michlmayr, "Quality Improvement in Volunteer Free Software Projects: Exploring the Impact of Release Management," Ph.D. dissertation, University of Cambridge, 2007.

[16] A. Wise, A. Cass, B. Lerner, E. McCall, L. Osterweil, and S. Sutton Jr, "Using little-jil to coordinate agents in software engineering," in *Proceedings of the Fifteenth IEEE International Conference on Automated Software Engineering*. IEEE, 2000, pp. 155–163.

[17] D. Perry, "Enactment control in interact/intermediate," *Software Process Technology*, pp. 107–113, 1994.

[18] L. Osterweil, "Modeling processes to effectively reason about their properties," in *Proceedings of the ProSim '03 Workshop*, 2003.

[19] L. Osterweil, "Unifying microprocess and macroprocess research," in *Unifying the Software Process Spectrum*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2006, pp. 68–74.