BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

AUTOMATION OF RELEASE ENGINEERING TASKS IN FEDORA LINUX

AUTOMATIZACE ÚKOLŮ PRO VYDÁVÁNÍ OPERAČNÍHO SYSTÉMU FEDORA LINUX

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR ANTON MEDVEDEV
AUTOR PRÁCE

SUPERVISOR Ing. ONDŘEJ LENGÁL, Ph.D.
VEDOUCÍ PRÁCE

BRNO 2023

Bachelor's Thesis Assignment

156945

Institut:	Department of Intelligent Systems (DITS)
Student:	**Medvedev Anton**
Programme:	Information Technology
Title:	**Automation of Release Engineering Tasks in Fedora Linux**
Category:	Software Engineering
Academic year:	2023/24

Assignment:

1. Get familiar with Fedora Release Engineering infrastructure and services. Focus on Standard Operating Procedures, particularly their part related to packaging and composing.
2. Investigate the current state of automation of Standard Operating Procedures.
3. Propose a set of methods to extend the automation of Standard Operating Procedures.
4. Implement the proposed methods as new tools or as a part of the existing tooling.
5. Execute the implemented tools in the production environment. Evaluate the amount of human time saved on Fedora release operations.
6. Summarize and describe the achieved results and discuss their possible future improvements.

Literature:

- Red Hat, Inc. and others: "Fedora Release Engineering Standard Operating Procedures". Dostupné online: https://docs.pagure.org/releng/sop.html.
- The Fedora Toddler tool: https://pagure.io/fedora-infra/toddlers/tree/main.
- Fedora Linux Release Life Cycle. Dostupné online: https://docs.fedoraproject.org/en-US/releases/lifecycle/.

Requirements for the semestral defence:
The first two points of the assignment.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Lengál Ondřej, Ing., Ph.D.**
Consultant:	Tomáš Hrčka
Head of Department:	Hanáček Petr, doc. Dr. Ing.
Beginning of work:	1.11.2023
Submission deadline:	16.5.2024
Approval date:	6.11.2023

Abstract

This paper aims to familiarize the reader with automating the Fedora Release Engineering team's task. It includes an introduction to Fedora Infrastructure and the processes involved in releasing new versions of Fedora Linux. To achieve this goal, I analyzed the possibilities for automation and chose to automate the Unretirement process by adding new functionality to the existing Fedpkg and Toddlers tools. The result was the addition of a new command for Fedpkg that creates a request and a plugin for Toddlers that processes the request. This result frees up engineers' resources for more important things.

Abstrakt

Cílem tohoto článku je seznámit čtenáře s automatizací úkolu týmu Fedora Release Engineering. Zahrnuje úvod do infrastruktury Fedora a procesů spojených s vydáváním nových verzí systému Fedora Linux. Pro dosažení tohoto cíle jsem analyzoval možnosti automatizace a rozhodl jsem se automatizovat proces uvolňování vydání přidáním nové funkce do stávajících nástrojů Fedpkg a Toddlers. Výsledkem bylo přidání nového příkazu pro Fedpkg, který vytvoří požadavek, a zásuvného modulu pro Toddlers, který tento požadavek zpracuje. Tento výsledek uvolní prostředky inženýrů pro důležitější věci.

Keywords

Operation system Fedora Linux, SOP(Standard Operation Procedures), Toddlers, Package Unretirement, fedpkg, fedora-scm-requests, AMQP message, RelEng (Release Engineering), Fedora Infrastructure

Klíčová slova

Operační systém Fedora Linux, SOP(Standardní pracovní postupy), Toddlers, odchod balíčku z důchodu, fedpkg, fedora-scm-requests, AMQP zprava, RelEng (Release Engineering), Fedora Infrastructure

Reference

MEDVEDEV, Anton. *Automation of Release Engineering Tasks in Fedora Linux.* Brno, 2023. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Ondřej Lengál, Ph.D.

Rozšířený abstrakt

Účelem tohoto článku je seznámit čtenáře s automatizací úkolů týmu pro vydávání verzí systému Fedora. Obsahuje úvod do infrastruktury Fedory a procesů spojených s vydáváním nových verzí systému Fedora Linux. Vzhledem k tomu, že projekt Fedora je rozsáhlý projekt s mnoha různými součástmi, které se vzájemně ovlivňují, aby fungoval. Bylo nutné čtenáře seznámit také s typickým životním cyklem balíčků a podobnými věcmi, aby pochopil infrastrukturu Fedory jako ucelený projekt. Vzhledem k omezeným lidským zdrojům se s růstem projektu zvyšuje počet procesů, které je třeba automatizovat. Najít proces, který to vyžaduje, a ukázat způsob jeho automatizace, který může motivovat lidi, aby se podíleli na vývoji open-source distribuce, bylo cílem mé práce.

K dosažení tohoto cíle jsem analyzoval možnosti automatizace. Můj konzultant mi doporučil, abych se podíval na procesy popsané v SOP jako na způsob jejich nalezení. Protože takové procesy jsou popularní mezi každodenní prací inženýrů RelEng. Protože mnoho z těchto procesů je již automatizováno na dobré úrovni, padl můj zrak na Unretirement SOP. Tento proces zahrnuje propojení s různými službami infrastruktury Fedora, což zvyšuje užitečnost popsaných informací pro podobné a další procesy.

Výsledkem analýzy tohoto procesu byl návrh na vytvoření doplňku pro stávající nástroje a služby. Jednou z nich je Toddlers. Tato služba má architekturu založenou na zásuvných modulech a lze ji poměrně snadno rozšířit. Funguje na principu, že se jednotlivé zásuvné moduly přihlašují k odběru konkrétních témat zpráv, v nichž vznikají požadavky, které by měly být automaticky zpracovány. V mém zásuvném modulu se odehrává hlavní část zpracování požadavků na odchod do důchodu. Včetně nastavení potřebných modulů a kontroly správnosti a platnosti dat požadavků.

Tato automatizace vyžadovala vytvoření dobře strukturovaného požadavku, ale lidé to ne vždy dělají, takže bylo rozhodnuto jim pomoci poskytnutím nástroje, který umožňuje vytvořit požadavek pomocí jediného příkazu v nástroji fedpkg. Účelem tohoto příkazu je zkontrolovat údaje o argumentech dodané žadatelem a vytvořit požadavek na základě těchto údajů.

Důležitým výsledkem této automatizace je také uvolnění lidských zdrojů. Protože tento proces vyžadoval ruční kontrolu requesteru a ruční zpracování. Nyní mohou inženýři věnovat více času zajímavým úkolům, které vyžadují kreativitu, spíše než ručnímu sledování, což zvyšuje kvalitu projektu Fedora a snižuje pravděpodobnost lidské chyby při podpoře životního cyklu balíčků.

Automation of Release Engineering Tasks in Fedora Linux

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Ing. Ondřej Lengál Ph.D. The supplementary information was provided by Ing. Tomáš Hrčka my external consultant from Red Hat. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

. . . .
Anton Medvedev
May 16, 2024

Acknowledgements

I would like to thank Ing. Ondřej Lengál, Ph.D., for his assistance. I would also like to thank Ing. Tomáš Hrčka, who provided me with meaningful advice and encouragement to complete the project. Last but not least I would like to express my gratitude to my mother, Ing. Marina Medvedeva, for her support during my whole life.

Contents

List of Figures

Chapter 1

Introduction

Fedora Linux is a popular open-source Linux distribution. In fact, even the distribution of choice by Linus Torvalds (2020)[34]. People who work for this distribution are called the Fedora community. They include maintainers, packagers, stakeholders, as well as developers and *DevOps (development and operations)* engineers. The *CPE[18] (community platform engineering)* team is also a part of the community structured into a team that is helping Fedora Linux to maintain quality, security, robustness, and reliability. One of the sub-teams of CPE is *RelEng[9] (Release Enineering)* team. It is responsible for providing smooth Fedora release development on different stages and containing regulation of packages life cycle[7], which are part of the Fedora Active Packages repository. This team is processing different tasks. Most popular of them, the team writes down in the form of *SOP (Standard Operation Procedures)* to share knowledge between team members and the community on how to accomplish individual actions in the form of step-by-step instruction guidelines. Most of these SOPs still require manual processing, which can become a problem.

Unfortunately, CPE and releng don't have an unlimited budget and unlimited people resources, so it is necessary to keep on increasing responsibilities by automating as many SOPs as possible. In this thesis, I explained and described the process of automation works based on the automation of the Unretirement SOP. Trying to motivate community members or people who are interested in Fedora to help their favorite distribution grow and become better by participating in development.

There is a huge amount of SOPs that cover release engineering processes, but this thesis will concentrate on package-related ones. Mainly on the Unretirement SOP because it contains steps that are related to Fedora infrastructure and tools that are common within it so new people can better understand the overall picture. The People (mostly maintainers) who would like to unretire a branch (usually „rawhide" as the latest branch) now should leave a request in one of the issue topics, and on a daily basis, Fedora Release engineering goes through open tickets and process such requests manually.

The goal is to create or extend existing tools and provide an easy structure for the community to unretire branches of the package they need by automating the ticket processing and actions required to unretire individual branches of the package and reintroduce them to the Fedora Active Packages.

This thesis is organized as follows. Chapter 2 presents basic information about the releng team and the package-related SOP we are using. Chapter 3 is dedicated to describing the current state of the unretirement process in detail. Chapter 4 is devoted to explaining my idea on the automation of this process. Chapter 5 describes the implementation, including testing the environment and deploying it into production using Openshift and Ansible. In

Chapter 6, I sum up the results that I got, with estimations of time saved by people and reviews from the community.

Chapter 2

Fedora Release Engineering package related SOP's

In this chapter I will define the necessary terminology that will be encountered throughout the rest of this thesis. Next, I will describe the structure of the CPE team[18] and its sub-team RelEng[9], which works on release engineering practices, to better understand the context and the problem I am trying to solve in this thesis. Lastly, we will take a look at a few SOPs related to the package and describe the overall picture of the package lifecycle[7]. We will continue by choosing the suitable SOP for automation and further consideration.

2.1 Community Platform Engineering team responsibilities

The Community Platform Engineering is a part of Fedora and Centos communities. This team is contributing to the infrastructure and release engineering. It defines the mission statement 2.1, on which the team should balance.

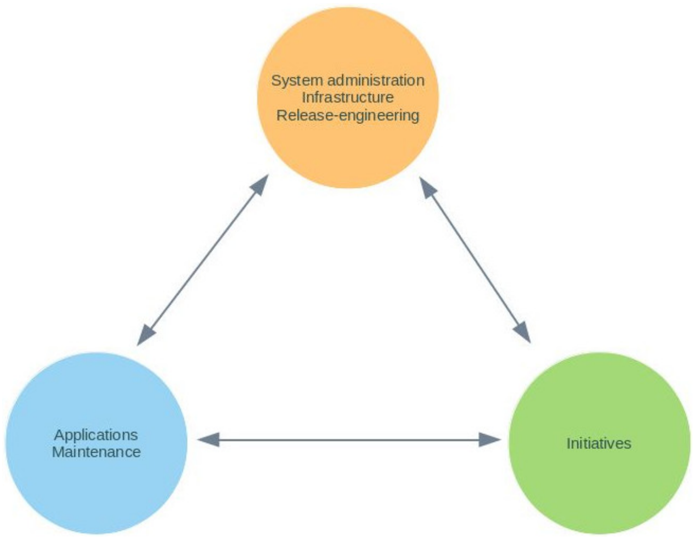


Figure 2.1: **Three axes of CPE mission**[18]

The CPE is responsible for infrastructure and services, including:

- Hosting, automating, monitoring, and maintaining infrastructure components: CPE manages and monitors networks, storage, and other infrastructure to support the Fedora and Centos projects.

- Service monitoring and lifecycle management of services hosted within our infrastructure: CPE monitors performance and any kind of issues to provide reliable access to the services within Fedora and Censos. The team also manages a lifecycle of services.

- Feature development for infrastructure-related initiatives: CPE actively develops tools and expands existing ones to process community initiatives. It includes automation of tasks.

- Tooling to enable all of the above: CPE dedicates resources to developing new tools and integrating them with infrastructure to cover the mission.

Those components are essential to the success of Fedora and Centos[18]. It's always handled through the ticket creation process because it is necessary to have logged every change to share it within the team and track the progress on individual tasks and initiatives[21].

2.2 Fedora Release Engineering activities

One of the crucial parts of CPE is the Release Engineering team. This team is ensuring timely and successful release of Fedora Linux versions. This team works on different tasks throughout the release cycle, from planning the new release and managing the building system to pushing updating packages to a production server. Here is the list of essential jobs to maintain Fedora as a cutting-edge and reliable Linux[9]:

- Release planning and preparation: In collaboration with Fedora Project Steering Committee (FESCo) is developing release plans, setting timelines, and preparing build infrastructure to ensure it is ready to handle the number of packages involved in Fedora build[7].

- Freeze management: Releng enforces freeze periods during the release cycle to ensure that the final version will have stability. It is the time that need to handle any issue.[15]

- Build System Administration: Releng monitors the system performance, troubleshoots build failures, and makes adjustments to ensure efficient builds.

- Package Updates and Pushes: Releng is responsible for the pushing latest updates of packages into production to ensure users have access to the latest software versions. The team manages any package conflicts and monitors the deployment process.

- Write and maintain tools to compose and push Fedora: Releng composes the final version of Fedora, combining packages and configurations into an operation system image.

- Documentation and Communication: Maintaining comprehensive documentation for the processes and tools. Also, the releng team communicates with the community using different channels, including mailing lists and IRC channels.

The following image 2.2 helps better understand the life cycle of Fedora releases:

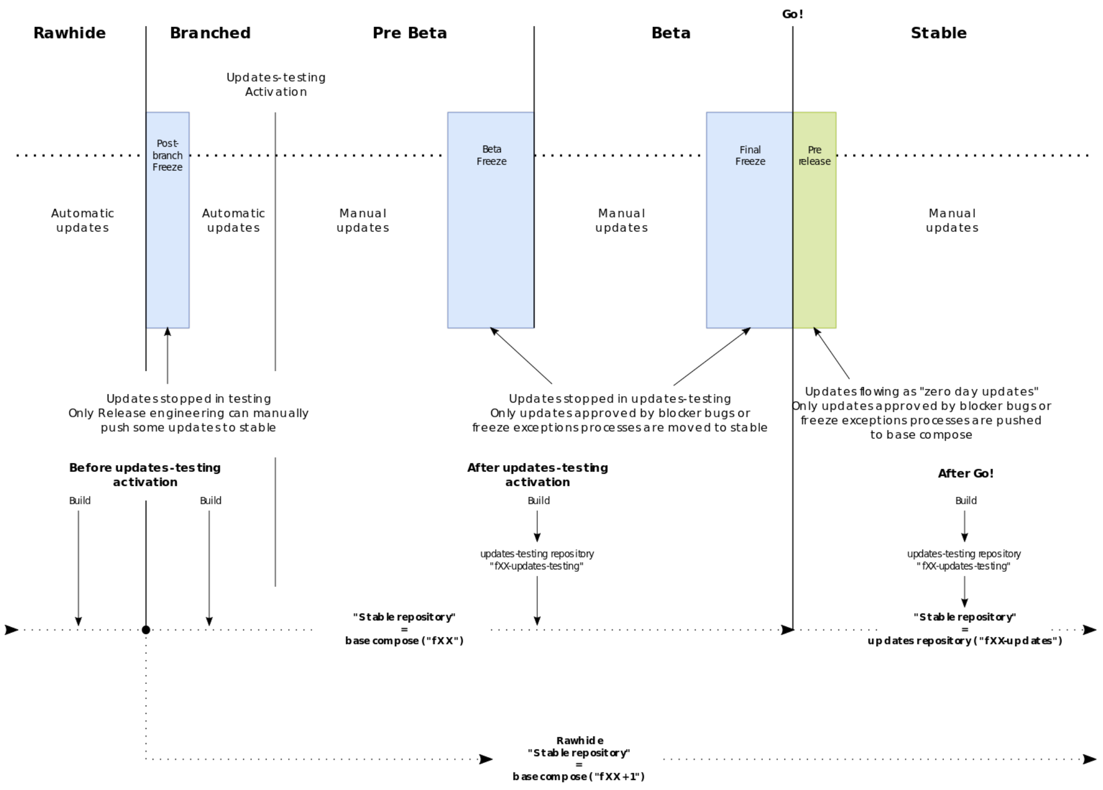

Go!
Rawhide
Branched
Pre Beta
Beta
Stable
Updates-testing Activation
Post-branch Freeze
Beta Freeze
Final Freeze
Pre release
Automatic updates
Automatic updates
Manual updates
Manual updates
Manual updates
Updates stopped in testing Only Release engineering can manually push some updates to stable
Updates stopped in updates-testing Only updates approved by blocker bugs or freeze exceptions processes are moved to stable
Updates flowing as "zero day updates" Only updates approved by blocker bugs or freeze exceptions processes are pushed to base compose
Before updates-testing activation
Build
Build
After updates-testing activation
Build
updates-testing repository "fXX-updates-testing"
"Stable repository" = base compose ("fXX")
After Go!
Build
updates-testing repository "fXX-updates-testing"
"Stable repository" = updates repository ("fXX-updates")
Rawhide "Stable repository" = base compose ("fXX+1")

Figure 2.2: **Creation new stable release of Fedora Linux**[35]

The activities of Fedora Release Engineering are really important for maintaining Fedora's position as a leading Linux distribution. I want to dedicate this thesis to solving a problem with the automation of package-related activities. So, it is necessary to understand a standard Package lifecycle.

2.3 Fedora Package Lifecycle

The casual package lifecycle in Fedora has several stages. Here is an overview of the most important ones.

- Development and Review: The lifecycle starts with developing and reviewing the package by Fedora's community and maintainers. This step ensures that packages are following Fedora Packaging guidelines[25]. Maintainers will be responsible for the package updates and following maintenance[7].

- Branching and Freezing: The main branch of development is called *Rawhide*. Releases are *Branched* from Rawhide before they are sent out as a stable release.[10] Fedora maintains a stable branch for every release, including creating a freeze for every milestone (Alpha, Beta, Final)[10] during this period, the build won't be marked as stable and pushed from "updates-testing" status[30].

- Composing Releases: Packages are then composed into the Fedora release. Using tools like Pungi and Livecd-creator. This stage ensures the appearance latest versions of packages in the new Fedora distribution[10].

- Testing and Validation: Before being released, a package goes through tests and validation. It involves the QA team that maintains the process and setting up test events[19]. The goal is to ensure that the release candidate composition meets Fedora Release Criteria[14].

- Release and Update: After the release, the package can get updates and bug fixes. This process is handled through the tool *Bodhi*, which oversees updates and ensures they are aligned with Fedora policy[31].

- End of Life (EOL): Each Fedora release and its packages have defined a schedule of End of Life when it no longer receives updates and support. Maintainers should be aware of it to move the package to the newer versions[22].

- Retiring Package: When a package reaches the end of its useful life or being replaced or the maintainer no longer supports it (orphaned package) the package should be retired. It involves removing packages from the Fedora repositories. This process exists to ensure that Fedora distribution remains current and maintainable[28].

Those stages of a package's life are crucial to supporting a healthy lifestyle for Fedora Linux. This is why CPE involves SOPs. They describe strict guidelines on how to process individual steps to succeed in one action. In this thesis, I would like to focus on automating the SOP process to make the life of Fedora Release engineers easier. I would like to focus on a few interesting package-related SOPs. They will be described in the next section.

2.4 Package-related SOPs

Fedora Release Engineering (releng) employs various Service Operations Procedures to cover a list of tasks that are common. It is open because we want the community to engage with it and make new SOP documentation. This way helps share knowledge between team members and the community and ensures that not just individuals know how to fix a problem. Basically, SOP is a set of steps required to accomplish a particular task. These SOPs cover many aspects of package management, including the creation and branching of new packages, managing updates, and blocking and unblocking packages. For instance, the *Branching SOP* describes the way how to make *git* and *pkgdb* branches for the package either for new packages that passed review or for existing ones[8]. Releng created a script to automate it. Those procedures are crucial for maintaining the quality and integrity of Fedora Linux, ensuring all the packages are up-to-date and compatible with the entire system. SOP is a clear guideline for maintainers and contributors to follow to keep an efficient workflow within Fedora Project[11]. There are a few SOPs related to *EOL (End Of Life)*, and I would like to focus on them.

Retire SOP

Retirement is a really important process that ensures that the package will be removed if it is no longer needed or lost its maintainer. This procedure usually involves a maintainer who will decide to retire a package for various reasons such as obsolescence, lack of upstream, or personal disability to maintain it. In case the package is still useful for Fedora, it's getting orphaned status. Then other maintainers or contributors who are interested in maintaining can get ownership of this package. *Orphan package* still remains in stable versions but will become the responsibility of the community. If the package remains orphaned in the rawhide branch longer than 6 weeks or if it is no longer needed, then the package will be retired[20]. The maintainer should open a ticket to create a request. After that, Releng can retire an orphan package using a script that creates and verifies successful retirement[12].

Unretire SOP

Service Operations Procedure for package branch unretirement was designed to enable return packages back to life. It usually happens with the rawhide branch, as the latest one, but it can be applicable to other branches as well. This process was typically initiated when the package that was previously retired was needed again or when the maintainer took over the responsibility for looking after the package that was orphan. The unretirement process begins with a request from the packager[13]. This request should be filled in a specific ticket tracker[1]. After that, a member of the Release engineering team will ensure that the package complies with Fedora packaging guidelines and policies[20]. Upon approval, the package branch will be recreated, and the package will be returned to the Package Database. This SOP is important for the flexibility and responsiveness of the Fedora ecosystem.

In conclusion, standard operations Procedures play a significant role in the work of the Release Engineering team and community. These SOPs cover everything from the creation, updating, and retirement to the unretirement of packages to ensure a well-structured approach to package management. The problem is that a lot of SOPs still require manual execution, and this makes the life of the team much harder, given the ever-increasing number of packages. Notably, the unretirement process still stands on a manual nature, it's not automated like the retirement process or some of the other SOPs. This makes it an interesting topic for further investigation in this thesis. Focusing on the unretirement process provides the opportunities to investigate nuances of package management in open-source projects and help the community and the releng team, in particular, to automate an important part of the work routine.

[1] https://pagure.io/releng/issues

Chapter 3

Current state of package unretirement process

Chapter 3 of this thesis describes the current state of the package unretirement process in the Fedora Project. This process is crucial in Fedora package lifecycle management. It allows the reintegrating of packages that were previously retired, ensuring the distribution can adapt to the changes. The chapter will explore the steps that are required to process this task. Started with the creation of a ticket in a specific ticket tracker to show a desire to unretire a package. Then, process the Validation and Verification of the package to ensure it is ready and appropriate for unretirement. After that, there are technical steps, such as reverting the retirement commit and unblocking the package on *Koji (Fedora's building server)*. The process also includes verifying the package is not orphaned and updated Product Definition Center (PDC). Each of these steps is important to ensure the smooth reintegration of packages into the Fedora active packages repository[13].

3.1 Ticket Creation for unretirement process

The ticket creation process is a formalized step to initiate an unretirement process. The maintainer, a packager, or a community member who would like to reintroduce the package in the Fedora Active Packages should fill a ticket in a specific releng[1] ticket tracker on pagure[5]. Releng prepared a few ticket templates, including a package unretirement template 3.1, to make ticket creation easier.

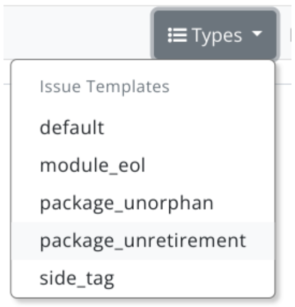
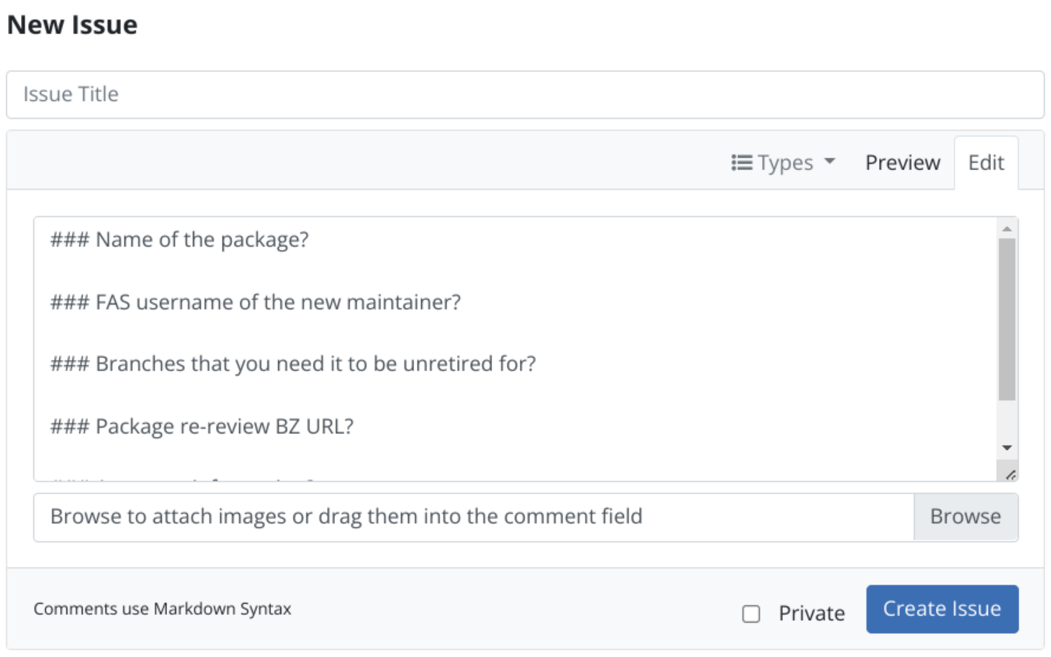


Figure 3.1: **Selecting a suitable template.**

[1] https://pagure.io/releng/issues

After choosing the right template, the requester should provide comprehensive information that is needed to continue reviewing the request. By answering particular questions in a template. An example of a template is in the figure 3.2 below:



New Issue

Issue Title

```
### Name of the package?

### FAS username of the new maintainer?

### Branches that you need it to be unretired for?

### Package re-review BZ URL?
```

Browse to attach images or drag them into the comment field
Browse
Comments use Markdown Syntax
Private
Create Issue

Figure 3.2: **Unretirement template.**

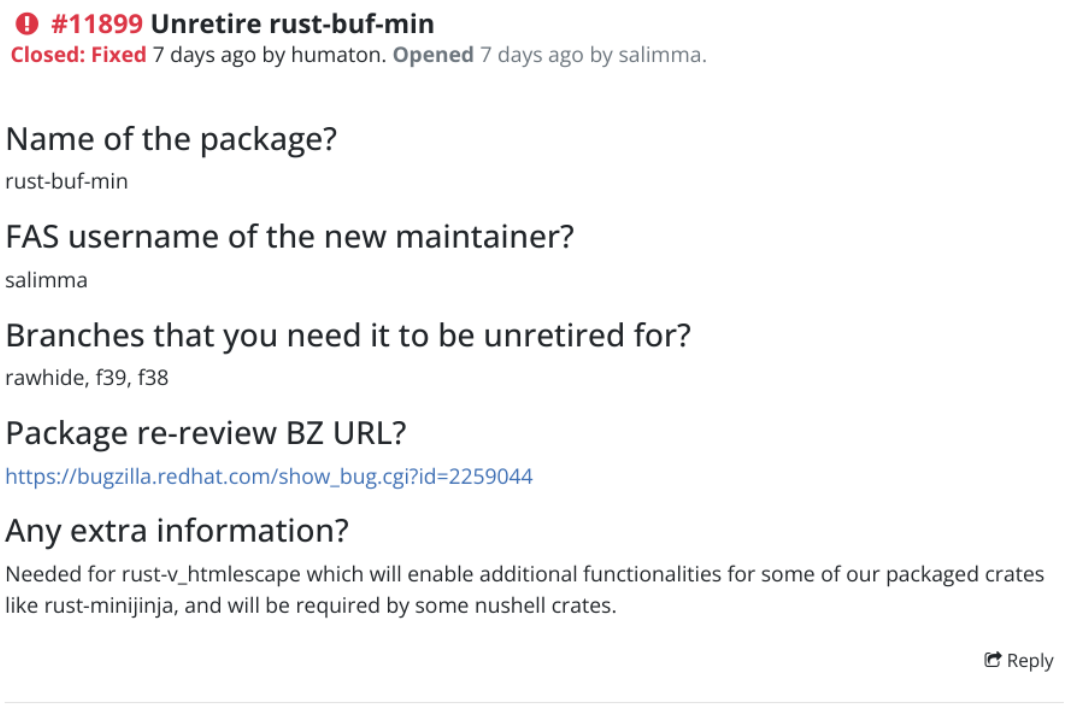
Here's an explanation of the fields that are requested to fill:

- Issue Title: This field usually looks like „Unretire“ + the name of the package. It is necessary to have a more robust structure of tickets in the ticket tracker.
- Name of the Package: The name of the package itself. It should be the name of the package that the package would like to unretire.
- FAS username of the new maintainer: This field should contain the Fedora Account System (FAS) username of the user who will be the maintainer of the unretired package. If this field is empty, then the requester himself will be a new maintainer of the package.
- Branches that you need to be unretired: The Unretirement SOP requires branches to process. It's necessary to provide a list of branches that you need to unretire a package on. Those branches usually contain the rawhide branch as the latest one and might have few of the stables branches. If this field is empty, then the package will be unretired just on the rawhide branch.
- Package re-review BZ URL: This field should contain a *Red Hat Bugzilla* URL with a package review. Red Hat Bugzilla is a Red Hat bug-tracking system that supports submitting and reviewing packages and bugs that are found in Red Hat distribution[4]. Before opening the ticket with an unretirement request, the requester should submit

a review request on the package in RedHat Bugzilla and get a fedora-review+ flag. BZ URL is not necessary if the package was retired less than 8 weeks ago[28].

- Any extra information: In this field, the requester can provide any additional information that might be important or helpful or keep this field empty.

After the requester sends a ticket, one of the members of the releng team will look into it. If everything is clear, then the releng engineer will process an SOP. Otherwise, if some important part is missing, the engineer will notify the requester in the comments section under the ticket to correct missing or wrong parts and reopen the ticket. Here is an example of a well-structured Unretirement ticket:


#11899 Unretire rust-buf-min
Closed: Fixed 7 days ago by humaton. Opened 7 days ago by salimma.

Name of the package?
rust-buf-min

FAS username of the new maintainer?
salimma

Branches that you need it to be unretired for?
rawhide, f39, f38

Package re-review BZ URL?
https://bugzilla.redhat.com/show_bug.cgi?id=2259044

Any extra information?
Needed for rust-v_htmlescape which will enable additional functionalities for some of our packaged crates like rust-minijinja, and will be required by some nushell crates.

Figure 3.3: **Example of well-structured unretire ticket.**

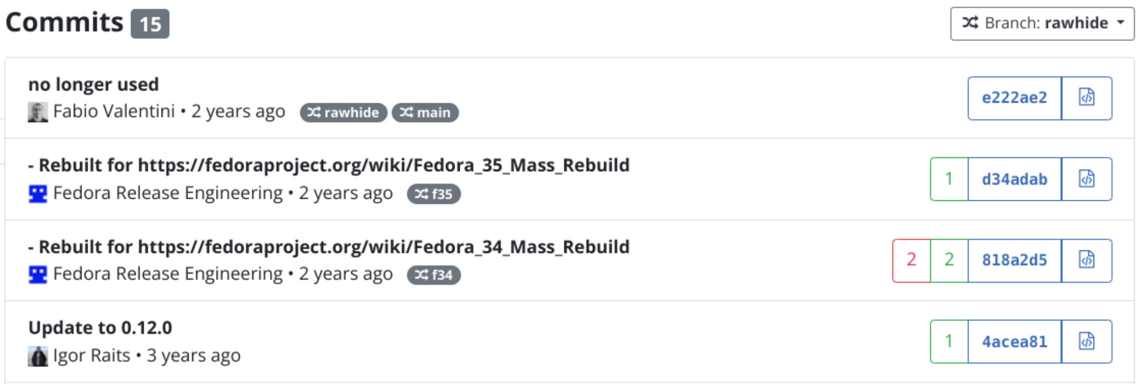
The ticket creation procedure is crucial for maintaining transparency and collaboration within Fedora, ensuring high-quality package management standards. It reflects the community-driven approach.

3.2 Validation Package is ready for Unretirement

Validating a package for unretirement is an important step to ensure that the package is safe. It needs to be done to keep Fedora Linux warranted and feasible. The validation process includes the following steps[13]:

Legal and Licence Verification

It should be confirmed that the package wasn't retired because of legal or license issues that would prevent it from being reinstated. This involves checking retired packages in the packages repository. The retired package will have only a specific file on the branch, which indicates its status: `dead.package`. This file is used to indicate that the branch is obsolete and no longer in use. To ensure that there were not any Legal or Licence issues behind retirement, releng, members should look at the last commit message (the commit that removed all files and left `dead.package` file there). If this message has a clear reason, such as "no longer need" 3.4, it's okay to unretire the package. The last commit should always be a retirement commit when the requester asks for unretirement.


Commits 15 | Branch: rawhide

no longer used
Fabio Valentini • 2 years ago rawhide main — e222ae2

- Rebuilt for https://fedoraproject.org/wiki/Fedora_35_Mass_Rebuild
Fedora Release Engineering • 2 years ago f35 — 1 d34adab

- Rebuilt for https://fedoraproject.org/wiki/Fedora_34_Mass_Rebuild
Fedora Release Engineering • 2 years ago f34 — 2 2 818a2d5

Update to 0.12.0
Igor Raits • 3 years ago — 1 4acea81

Figure 3.4: **Example of commit history of retired package.**

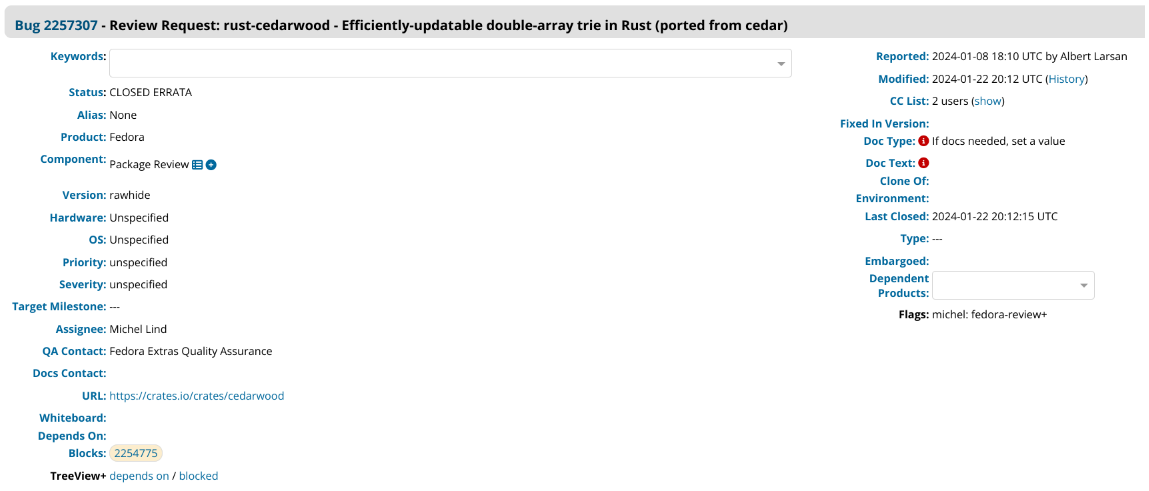
Bugzilla review

As mentioned before, the Bugzilla review URL should be provided just in case the package was retired more than 8 weeks ago and requires re-reviewing on Bugzilla[2]. Review process using `fedora-review` flag on Bugzilla. This flag can have the following statuses[29]:

- fedora-review "(BLANK)": Package needs review.
- fedora-review "?": The package is under review.
- fedora-review "-": The package failed review, dropped for legal or other issues.
- fedora-review "+": The package approved

Releng engineer must ensure that a package passes review on Bugzilla to continue the unretirement process. The flag is shown after all data about review in the **Flags** section 3.5.

[2] https://bugzilla.redhat.com/



Bug 2257307 - **Review Request: rust-cedarwood - Efficiently-updatable double-array trie in Rust (ported from cedar)**

Keywords:	
Status:	CLOSED ERRATA
Alias:	None
Product:	Fedora
Component:	Package Review
Version:	rawhide
Hardware:	Unspecified
OS:	Unspecified
Priority:	unspecified
Severity:	unspecified
Target Milestone:	---
Assignee:	Michel Lind
QA Contact:	Fedora Extras Quality Assurance
Docs Contact:	
URL:	https://crates.io/crates/cedarwood
Whiteboard:	
Depends On:	
Blocks:	2254775
TreeView+	depends on / blocked

Reported:	2024-01-08 18:10 UTC by Albert Larsan
Modified:	2024-01-22 20:12 UTC (History)
CC List:	2 users (show)
Fixed In Version:	
Doc Type:	If docs needed, set a value
Doc Text:	
Clone Of:	
Environment:	
Last Closed:	2024-01-22 20:12:15 UTC
Type:	---
Embargoed:	
Dependent Products:	
Flags:	michel: fedora-review+

Figure 3.5: **Example of successful package review on Red Hat Bugzilla (fedora-review flag has „+“ status).**

This review process allows the community or stakeholders to comment on potential issues or provide additional insights that may affect unretirement decisions.

Verification tags, that required to be unretired

The requester should specify which *tags (branches)* should be unblocked. Tags are helping to manage different versions of Fedora releases, so accuracy in this aspect ensures that the package will be unblocked on the right versions. It's important to keep in mind that tags should be the tags of current or pending releases. An example of tags is shown in this figure 3.3.

The Validation process is developed to maintain the integrity of the Fedora distribution. If these steps are carefully followed, Fedora can continue to provide robust, secure, and legal packages to the end users.

3.3 Reverting the Retirement Commit

Reverting a retirement commit is a necessary step for the unretirement process. This action undoes the changes that were made to mark a package as retired. Here is how usually this process unfolds[13]:

1. Accessing compose system: The Fedora Release engineer will start by connecting one of the compose matching though `ssh`. This provides the necessary infrastructure to perform package management tasks[26].

```
ssh compose-x86-02.phx2.fedoraproject.org
```

2. Clonning the Package Repository: This step involves cloning a package's git repository using releng credentials. This is done through the *CLI (command line interface)*, that fedora infrastructure provides, such as `relengpush` or `fedpkg`. *Fedpkg* is more popular so further investigation will consider it as a main option.

```
GIT_SSH=/usr/local/bin/relengpush fedpkg --user \
releng clone PACKAGENAME
```

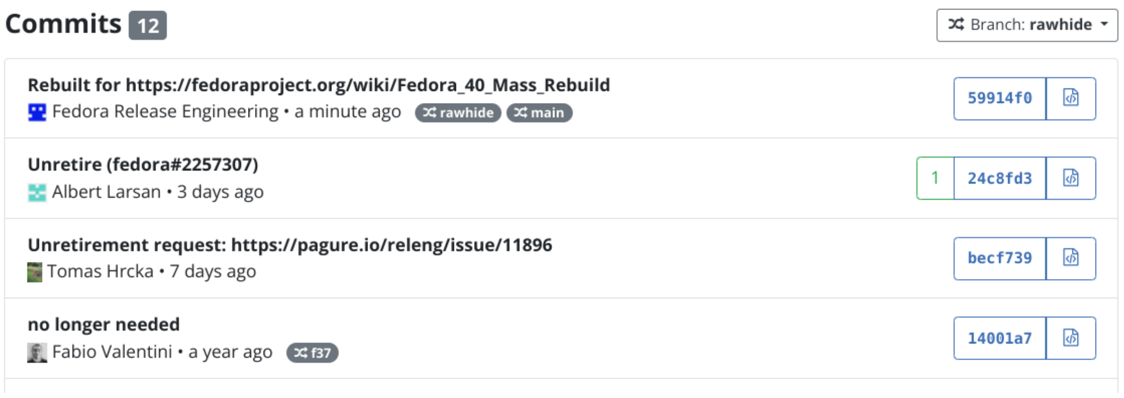
3. Setting Git Configuration: After cloning the package repository, the RelEng engineer needs to enter the directory and configure git user information to ensure that the action taken will be properly attributed.

```
cd PACKAGENAME
git config --local user.name "Fedora Release Engineering"
git config --local user.email "releng@fedoraproject.org"
```

4. Reverting the retire commit: The core step is to revert a commit that contains `dead.package` file on the particular branch using its commit hash_id. Ensure the commit message contains a URL to the request in pagure[13].

```
git revert -s COMMIT_HASH_ID
GIT_SSH=/usr/loca/bin/relengpush fedpkg --user releng push
```

An example of commit structure with revert commit is shown in the following figure 3.6:


Commits 12 — Branch: rawhide

Rebuilt for https://fedoraproject.org/wiki/Fedora_40_Mass_Rebuild Fedora Release Engineering • a minute ago rawhide main	59914f0
Unretire (fedora#2257307) Albert Larsan • 3 days ago	1 24c8fd3
Unretirement request: https://pagure.io/releng/issue/11896 Tomas Hrcka • 7 days ago	becf739
no longer needed Fabio Valentini • a year ago f37	14001a7

Figure 3.6: **Example of commit structure with reverted retire commit.**

Completing these steps makes the package active in the Fedora repository, and it is ready to proceed with the rest of the unretirement actions.

3.4 Unblocking the Package on Koji

For unretirement, unblocking the package on *Koji* [3] is an important step. Koji is Fedora's packages building system, where packages are built before they can be part of the distribution[37]. So when the package is retired, it's also blocked on Koji to prevent it from being built or updated. To restore a package, the following steps are taken[13].

[3]https://koji.fedoraproject.org/koji/

1. Check current state: The first step is to ensure that the package is actually blocked on Koji. This is done by running the following command.

   ```
   koji list-pkgs --show-blocked --package=PACKAGENAME
   ```

2. Unblock requested tags on Koji: Once confirmed that the package is blocked, it's time to unblock every tag that the requester specified. Every tag is responsible for the build of the package on different Fedora releases or release candidates. This is done by running the following command.

   ```
   koji unblock-pkg TAGNAME PACKAGENAME
   ```

This process reenables the package to be built and updated through the Koji system, reversing its retirement status and reinstating it in the Fedora build process. Unblocking on Koji is a crucial part of processing that allows a package to move through the build pipeline.

3.5 Verifying that package is not Orphaned

Verifying the package is not Orphaned is a step to ensure that Fedora keeps a healthy ecosystem. If the package is orphaned (that means that the package doesn't have an active maintainer and owner), it can't get updates and security patches. Here is how verification typically works[13]:

1. Check package ownership: The first step is to find out the maintainer of this package. This involves navigating to the package sources webpage [4]. And checking details to see if the package has maintainers. If there are some *Fedora Accounts* usernames, then the package is not orphaned.

2. Transfer the responsibility over the package to the requester: If a package is orphaned, that means it doesn't have a current maintainer. In this case, to prevent removing the package from distribution, a requester will become a new maintainer for this package. The Release Engineering team has a script that allows them to assign a package to the new maintainers. To achieve this the engineer needs to run the following command:

   ```
   ./scripts/distgit/give-package --package=PACKAGENAME \
   --custodian=REQUESTOR
   ```

   For running this script, the engineer who is processing unretirement requests must be part of `cvsadmin` group. Groups like this are the main way how we add and restrict rights for individual persons within *Fedora Infrastructure*. So, if the person is not a part of this group, then this person should create a ticket and ask to be added to this group.

The verification and reassignment process is vital to ensuring that all packages in the Fedora distribution have maintainers who can provide updates and security patches, which is crucial for Fedora's stability and reliability.

[4]https://src.fedoraproject.org/

3.6 Update Product Definition Center (PDC)

The *PCD (Product Definition Center)* is a repository and sets of APIs to collect and store metadata related to the packages, releases, and artifacts that are required to support Release Engineering workflow[3]. Fedora's implementations of PDC allow Fedora to maintain a database of composes and their components, which helps with automation and making decisions. The following figure shows the layout of PDC3.7 and the information it stores

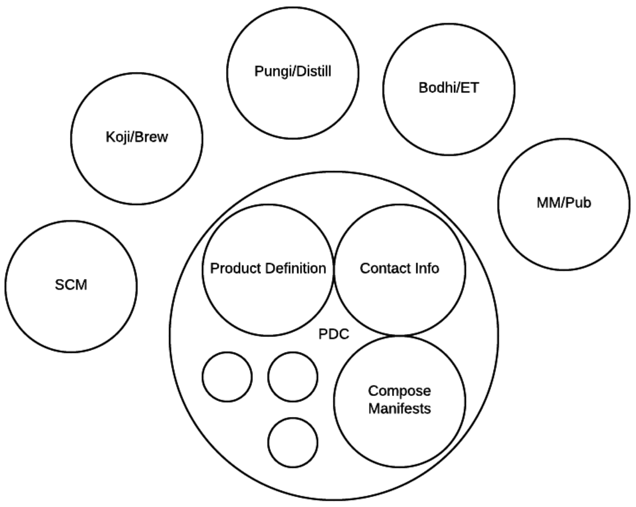


Figure 3.7: **PDC overview**[16]

The PDC update is a process that requires several steps. Here is how the PDC updates information for the Unretirement process[13] and what should be done to get rights for it:

1. Log into PDC: The first step is to log into PDC using your *FAS (Fedora Account System)* account. This account is used to authenticate maintainers and contributors and provides necessary permission to make changes in PDC.

2. Check PDC entry: After login, you need to check every branch that was unblocked in previous steps on a package by querying PDC's REST API. The example of the link is: `https://pdc.fedoraproject.org/rest_api/v1/component-branch-slas/?branch=TAG&global_component=PACKAGENAME` If no information is returned, the package or its branch is probably missing in PDC, so the person who is processing SOP should additionally create a request for a new branch using `fedpkg request-branch` command.

3. Obtain a Token: If a package and its branches exist within PDC, the RelEng engineer must obtain a token from the PDC site. It often involves navigation to the `https://pdc.fedoraproject.org/rest_api/v1/auth/token/obtain/` section with *Firefox web browser*. Then `F12` should be pressed and selected a tab labeled *Network*. After refreshing a page, the engineer should find a line whose string matches with `/rest_api/v1/auth/token/obtain/` column.

4. Extract a token as cURL: After the previous step, right-click and select: `Copy>Copy as cURL`. Now the engineer should add this `cURL` into a terminal and add `-H "Accept: application/json"` it should look like similar to command below:

   ```
   curl 'https://pdc.fedoraproject.org/rest_api/v1/auth/token \
   /obtain/' \
   -H 'Host: pdc.fedoraproject.org' \
   -H .0) Gecko/20100101 Firefox/57.0' \
   -H 'Accept: text/html,application/xhtml+xml, \
   application/xml;q=0.9,*/*;q=0.8' \
   -H 'Accept-Language: en-US,en;q=0.5' \
   --compressed \
   -H 'Cookie: csrftoken=CSRF_TOKEN_HASH; SERVERID=pdc-web01; \
   mellon-saml-sesion-cookie=SAML_SESSION_HASH; \
   sessionid=SESSION_ID_HASH' \
   -H 'Connection: keep-alive' \
   -H 'Upgrade-Insecure-Requests: 1' \
   -H 'Cache-Control: max-age=0' \
   -H "Accept: application/json"
   ```

   By running this command, the engineer will obtain a token needed for further processing.

5. Use a token to run PDC update script: Finally when the token is obtained, the releng engineer can run a script from the Release Engineering Repository [5]. This script will update PDC entries with new information related to the unretirement requests. Script will require `TOKEN`, `PACKAGE_NAME` and `TAG`. A person who is running this script can also add additional flags to perform the update correctly. The command looks like the following:

   ```
   PYTHONPATH=scripts/pdc/ scripts/pdc/adjust-eol.py fedora \
   MYTOKEN PACKAGENAME rpm TAG default -y
   ```

This process ensures that PDC is reflected in the current state of a package in Fedora, and any tools that use PDC will get up-to-date and correct information about a package. It is crucial to maintain the integrity of Fedora's packages data and metadata throughout Fedora infrastructure.

The Unretirement Package Branch SOP in Fedora is a multi-step procedure that evaluates that package is carefully reviewed and reintegrated into the Fedora Packages ecosystem. This involves a set of steps from the ticket initiation process to the updating of metadata in the Product Definition Center. Each step is designed to maintain the security, integrity, and quality of Fedora Linux. After this whole process, the initial ticket can be closed, and the releng member who was responsible for it can move to the next work unit. This SOP highlights Fedora's commitment to a robust, reliable, and transparent package management system. It required careful coordination between team members, maintainers,

[5] https://pagure.io/releng

and stakeholders, which emphasizes open-source principles on which Fedora is staying. By enabling unretirement, Fedora allows the continuous evolution of its repository, adapting to the changing needs of its user base and embracing the dynamic nature of open-source software development. This and the complexity of this process make me want to help the open-source community and automate this SOP.

Chapter 4

Plan of Unretirement Process Automation

This chapter covers the plan of automation. The first idea was that it could be achieved by writing a Python script that would automatically process all checks that the unretirement process required, followed by actions that return the package to the list of Active Fedora Packages. But this solution won't be ideal. It will still require members of the RelEng team to define specific requests from the issue topic and get all the authentications needed to run the script, followed by running the script with specific parameters where errors can occur. The next idea is to make this process fully automated. To solve this problem, a few tools and services need to work together. Automation will require:

- Fedora Messaging - This package provides tools and APIs to make using Fedora's messaging infrastructure easier[17].
- Toddlers - Toddlers is a simple application to run tasks upon fedora-messaging notifications[6].
- Fedpkg - Tooling for working with Fedora's dist-git and artifact build, including RPMs, containers, and modules[2].

This chapter will explain how each tool works, followed by a proposal of extensions for those tools. In conclusion, I will show the final scheme of tool collaboration.

4.1 Fedora messaging

The best place to start is by explaining the concept of Fedora Messaging. Fedora Messaging plays a key role in the Fedora Project's infrastructure. It is designed as a way for communication between different components within the ecosystem. Since Fedora was continuously growing in complexity and scale, Fedora Infra required a solution that would allow communication between different services. This problem was addressed with a message bus architecture[23]. Fedora Messaging is built to handle asynchronous message sending, which enables different services to interact and exchange information effectively.

Fedora Messaging provides a secure, scalable, and reliable messaging framework[1]. It serves for efficient message passing, automating workflows, distributing information, and

[1]https://github.com/fedora-infra/fedora-messaging

managing event-driven actions. The valuable factor is that it works with different tools such as Koji, Bodhi, FAS, and more. So, for Fedora Infrastructure, it's something similar to arteries for humans. Here is a scheme of collaboration through Fedora Messaging:

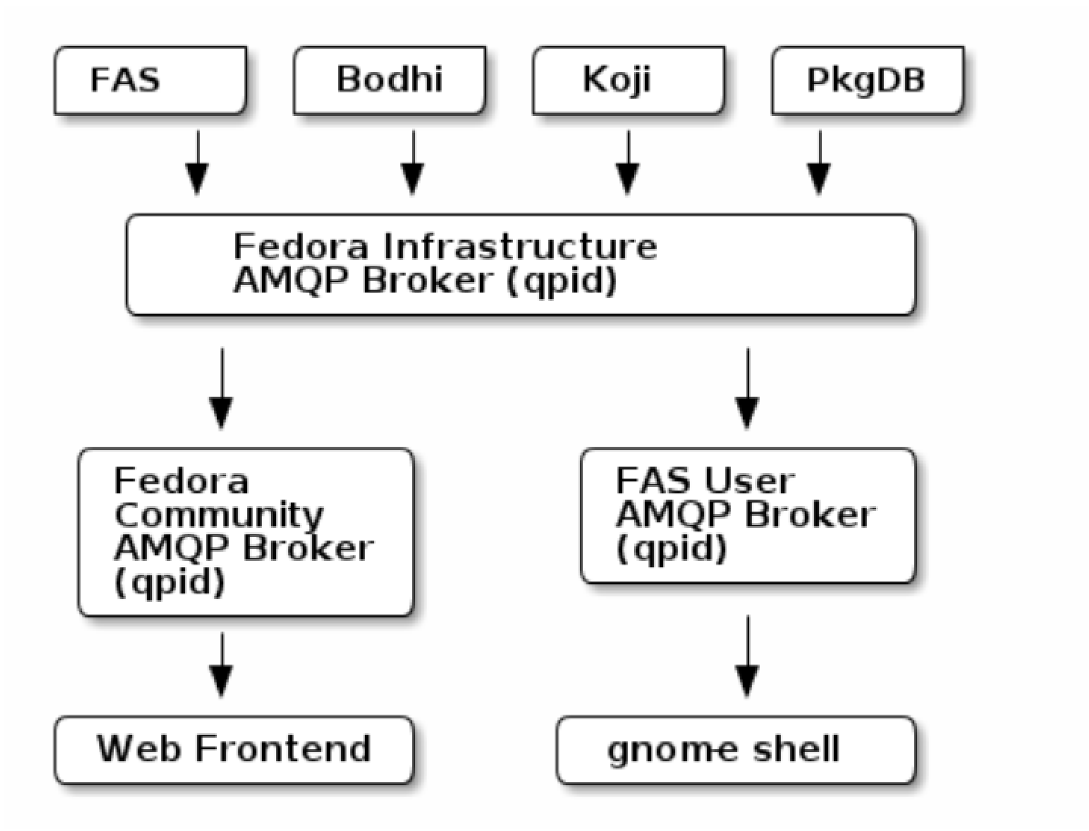

FAS
Bodhi
Koji
PkgDB
Fedora Infrastructure AMQP Broker (qpid)
Fedora Community AMQP Broker (qpid)
FAS User AMQP Broker (qpid)
Web Frontend
gnome shell

Figure 4.1: **Colaboration of differents tools trough Fedora Messaging**[23]

Fedora Messaging is built on top of the Advanced Message Queuing Protocol (AMQP)[24]. This choice ensures, important for Fedora Ecosystem, qualities such as:

- Scalability: Capability to handle thousands of messages, making it perfect for Fedora's high-volume infrastructure
- Reliability: Build-in mechanisms that ensure message delivery
- Flexibility: The publish/subscribe model allows the different tools to subscribe to specific topics required for their operations and not to be overwhelmed by other information.

AMQP

AMQP 0-9-1 (Advanced Message Queuing Protocol) is a messaging protocol that enables conforming client applications to communicate with conforming messaging middleware brokers.[32]. The basic model of this protocol includes the following steps:

1. Messages are published to **exchanges**, which are something like mailboxes.

2. **Exchanges** distribute copies of messages to **Queues** using rules called bindings.
3. The **broker** delivers a message to the **consumer** that subscribes to a queue or demand from the consumer (fetch/pull).

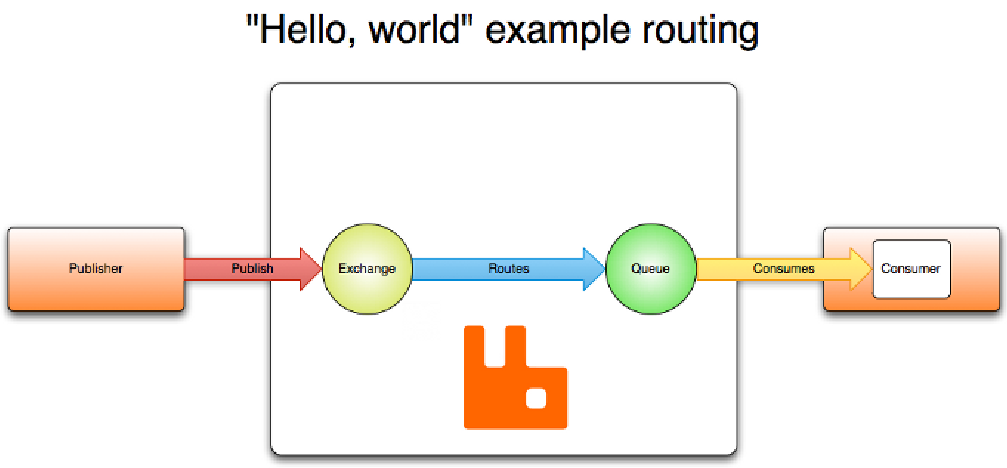


Figure 4.2: **AMQP example routing**[32]

Fedora messaging is deeply integrated into Fedora Infrastructure. It can send notifications from building tasks, automated testing alerts, and more. However, the functionality extends beyond notifications. That enables integration across different platforms such as Koji(building system) and Bodhi(updates system). For example, when a package build is completed on Koji, Fedora Messaging ensures that Bodhi will be notified to begin following the steps of the process.

To summarize, it plays an essential role in maintaining the efficiency of the integration of the Fedora Project's infrastructure. Its scalable and robust architecture easily adapts to the needs of Fedora. As Fedora continues to grow, Fedora Messaging will remain a key element that enables smooth collaboration between many project components.

4.2 Datagrepper

Fedora Datagrepper is a web-based tool essential for querying and analyzing vast amounts of data that are transferred by Fedora's messaging system. As part of Fedora infrastructure, Datagrepper interacts with Fedora Messaging, which carries event-driven data all across the ecosystem. It is very useful for users or processes that need to retrieve historical message data[2]. That makes it important for developers, administrators, and maintainers who are interested in system events and reactions to it[1].

Datagrepper provides users the ability to perform a detailed search on message histories with filters such as date ranges, topics, categories, packages, and more. This functionality is essential for tasks such as following progress, troubleshooting issues, and performing audits for Fedora Infrastructure activities. It is achieved by providing two modes of interaction: web interface and JSON API, which is used in a wide range of tools within Fedora Infrastructure. The API allows the creation of event-driven scripts based on message history.

[2]https://apps.fedoraproject.org/datagrepper/v2/search?

4.3 Toddlers

Fedora *Toddlers* is an essential tool due to the increasing complexity of Fedora's Infrastructure. It allows developers to automate everyday tasks and improve system efficiency. As the Fedora Project grows in scale, such a tool as Toddlers is becoming increasingly important to maintain the stability of Infrastructure. Automating repetitive tasks and providing specific insights into possible system issues allows Fedora contributors to significantly smooth their workflow and spend more time on complex tasks that require human interaction, such as developing new features, instead of spending on repetitive jobs that machines can achieve with better efficiency.

Fedora Toddles is a plugin-based system[3]. Contributors can customize its functionality and add new plugins to cover new repetitive tasks. It allows interaction with different Fedora Services, such as Koji, Pagure, and Bodhi, to automate activities like removing retired branches, creating new package repositories, providing different checks, and reporting anomalies. Toddlers are a bunch of small programs that keep running around. It is aimed at running tasks upon fedora-messaging notifications[6]. The modularity allows users to modify the functionality of specific scripts easily, and its extensibility makes it a reliable tool for the long term because it is likely to adapt to new challenges that may arise.

The tool represents a step forward in automating system tasks inside the Fedora Project. It reduces the manual effort required to do the work of CPE engineers and the number of errors that can occur when tasks are processed manually. An important aspect is that Toddlers is adaptive to the scaling of the Fedora Project, making it an effective long-term tool.

4.4 Fedpkg

Fedpkg is one of the most essential command-line tools on the Fedora Project. It makes it easier to perform package management and maintenance tasks by running commands. It provides *CLI (command-line interface)*[4], allowing users to build, create, and maintain packages. Fedpkg is "Front-end to the Fedora Infrastructure for package maintainers"[36]. Since the number of packages and contributors is constantly increasing, it is crucial to have such a tool that can work with the vast majority of repositories through a single interface.

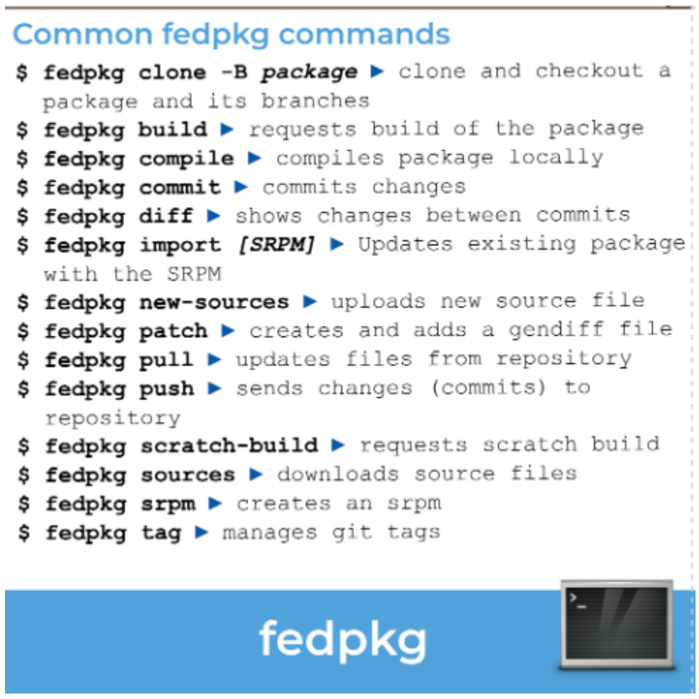
Fedpkg is designed to make a workflow for Fedora package maintainers easier because it automates and simplifies a lot of tasks for package lifecycle management. The most important features include:

- New package creation: It makes it easy to create a package within Fedora Infrastructure, as it has automatic scripts that perform repetitive tasks.
- Building and testing: Fedpkg is integrated with Koji, which allows to build and test packages either locally or remotely.
- Repository management: It has all popular git commands such as clone, push, pull, etc. So, the engineer can handle different actions with the git repository through an interface.

[3]https://pagure.io/fedora-infra/toddlers
[4]https://pagure.io/fedpkg

- Release management: Fedpkg is also integrated with metadata tools, which can be useful when the maintainer needs to tag packages for a particular release, upload updated packages to Bodhi, and so on.

The most popular *Fedpkg* commands are shown in the following image:



Common fedpkg commands

```
$ fedpkg clone -B package ▶ clone and checkout a
  package and its branches
$ fedpkg build ▶ requests build of the package
$ fedpkg compile ▶ compiles package locally
$ fedpkg commit ▶ commits changes
$ fedpkg diff ▶ shows changes between commits
$ fedpkg import [SRPM] ▶ Updates existing package
  with the SRPM
$ fedpkg new-sources ▶ uploads new source file
$ fedpkg patch ▶ creates and adds a gendiff file
$ fedpkg pull ▶ updates files from repository
$ fedpkg push ▶ sends changes (commits) to
  repository
$ fedpkg scratch-build ▶ requests scratch build
$ fedpkg sources ▶ downloads source files
$ fedpkg srpm ▶ creates an srpm
$ fedpkg tag ▶ manages git tags
```

Figure 4.3: **Fedpkg commands for Packaging process**[33]

Fedpkg is closely integrated with Fedora Infrastructure[2]. It uses Koji, Pagure, Bodhi. It offers an easy way for the building process, allowing maintainers to launch builds from the terminal and upload results to the relevant services. It lowers the barrier to entry for new contributors, as it simplifies package maintenance. It helps ensure important qualities such as security by reducing human factors and saves a huge amount of maintainers' time, allowing them not to spend time on repetitive tasks.

In conclusion, this tool became indispensable for the Fedora Project ecosystem, as it enables maintainers to be more effective and efficient in everyday work. With the integration into Fedora's infrastructure and automation of repetitive tasks, it became an important part of Fedora's development workflow. As Fedora evolves, such tools are becoming more crucial for maintaining the scalability of the Fedora ecosystem.

4.5 Proposed solution

This is the most important part of this chapter. Here, previous knowledge is grouped into a proposal. This section will explain how the idea changed from the beginning. It is necessary for understanding the next chapter and why those tools collaborate in this way.

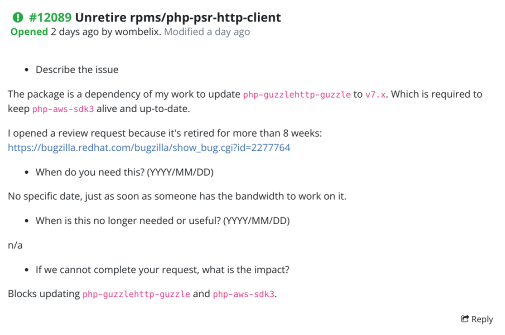
The best way to start is to consider whether actions really need to be taken or if RelEng should keep it as it is now. As mentioned earlier, in an ever-growing project, process automation becomes not just optional but mandatory 2.4. I have identified a few key reasons and benefits that the RelEng team will get from having the Unretirement process automated.

- Consistency and Accuracy: It will reduce the possibility of human-made mistakes.
- Faster Response Time: The requester doesn't need to wait for the RelEng engineer to review his ticket and process it. The same works the opposite; the engineer doesn't need to wait for the requester's reaction to mistakes or errors.
- Cost savings: Releng engineers will have more time to dedicate to initiatives and more important tasks that require human effort.

Ticket creation and problems associated with it

After understanding the reasons behind automation, it is important to start with the request itself because the way a person makes the request will impact the robustness of automation and the time it takes to process the request.

Every request is created in the form of a ticket. The appearance of a new ticket in a system will automatically send a Fedora Message with a specific topic. This topic, in most cases, is a place where the ticket was created. The first idea for automation was to implement a plugin for Toddlers 4.3 that would subscribe to a specific topic and follow messages from it. After receiving a message with the Unretirement request, it would consider doing all needed checks and controls followed by processing it. This solution can work, but it won't be ideal. The problem is that *Fedora Contributors* who are interested in such requests are different humans, and they are always happy to show their personalities. The following figure shows two similar requests for Unretirement, but they were filled by different persons and in different ways :



#12089 Unretire rpms/php-psr-http-client
Opened 2 days ago by wombelix. Modified a day ago

- Describe the issue

The package is a dependency of my work to update `php-guzzlehttp-guzzle` to `v7.x`. Which is required to keep `php-aws-sdk3` alive and up-to-date.

I opened a review request because it's retired for more than 8 weeks:
https://bugzilla.redhat.com/bugzilla/show_bug.cgi?id=2277764

- When do you need this? (YYYY/MM/DD)

No specific date, just as soon as someone has the bandwidth to work on it.

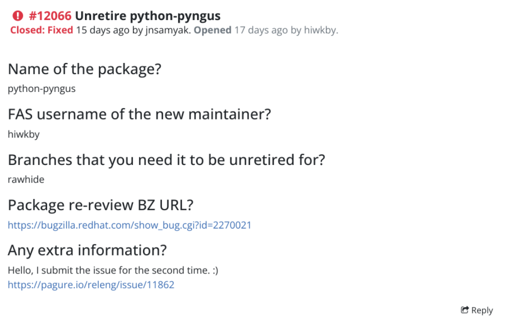
- When is this no longer needed or useful? (YYYY/MM/DD)

n/a

- If we cannot complete your request, what is the impact?

Blocks updating `php-guzzlehttp-guzzle` and `php-aws-sdk3`.

(a) Ticket for unretirement without using template



#12066 Unretire python-pyngus
Closed: Fixed 15 days ago by jnsamyak. Opened 17 days ago by hiwkby.

Name of the package?
python-pyngus

FAS username of the new maintainer?
hiwkby

Branches that you need it to be unretired for?
rawhide

Package re-review BZ URL?
https://bugzilla.redhat.com/show_bug.cgi?id=2270021

Any extra information?
Hello, I submit the issue for the second time. :)
https://pagure.io/releng/issue/11862

(b) Ticket for unretirement with using template

Figure 4.4: Different versions of people created tickets.

It can be difficult to automatically process requests, as people use various templates and write information in different ways. This means that a lot of code is needed to cover all the different possibilities, and even then, new types of requests may require further adaptation. While this solution can handle many requests, the release engineering team is looking for an ultimate solution that will remain effective for years to come.

In order to solve the problem, it is necessary to educate people on how to fill the ticket correctly. It will not be the perfect solution. So after the investigation, RelEng decided to use Fedora Infrastructure tools like Fedpkg to create the Unretirement tickets and motivate people to use this tool instead of creating a ticket by hand. The Fedpkg tool has a built-in feature that can be useful for creating tickets, for example, in commands such as `fedpkg request-branch` and `fedpkg request-repo`, that also create a ticket with different types of requests. These commands create a ticket requesting a new repository or branch in the existing repository. The ticket is created in a specific tracker called *releng/fedora-scm-requests*[5], designed to receive all tickets for further automatic processing. Although this solution can work, it should be subject to restrictions and conditions. The following section proposes a new `fedpkg request-unretirement` command.

Proposal fedpkg command

In this thesis, fedpkg will be utilized as a part of the automation process. Fedpkg was selected because it has similar functionality that aligns with the automation requirements. For instance, the `fedpkg request-branch` command can create a ticket in the *releng/fedora-scm-requests* issue tracker. This feature is precisely what is needed, but there is a difference between requesting a branch and requesting an Unretirement. Fortunately, as fedpkg is a command-line interface, it is relatively easy to extend its functionality to meet specific requirements. For creating a ticket, a new `fedpkg request-unretirement` command must work with the information provided by the requester. This information can be provided in the form of arguments, so the command is designed to handle the following arguments:

- repo: The name of the package that the requestor would like to unretire.
- namespace: The namespace of the repository to define a package.
- bz_url: A link to Bugzilla with a review request on the package; it is required in some cases.
- branches: A list of package branches that are requested to be unretired.

In the appendix A, a diagram of fedpkg processing is presented. Initially, fedpkg requests arguments, but none of these arguments are mandatory by default. Fedpkg is an intelligent tool that can obtain information on its own. For instance, if the user is in a repository on a particular branch and doesn't provide any arguments, the script will assume that the user wants to unretire the branch he is currently in. Additionally, it recognizes the user's Fedora Account for authentication purposes. If incorrect arguments are provided, an error will be raised. The requester should be cautious with the commands they are running. The request won't be processed if any of the data is incorrect or corrupted.

The requester must be aware of the checks that the command conducts for successful execution. These can be phrased as questions:

- If the branch was retired for longer than eight weeks, a Bugzilla URL argument should be provided. Is the Bugzilla URL provided?
- Is the user in the packager group?
- If BZurls is provided, does it have a fedora_review+ flag?

[5] https://pagure.io/releng/fedora-scm-requests

If the user's answer to any of the previous questions is negative, he should be informed of the changes he is required to make.

After a successful request, a ticket will be created. The current plain text format of the ticket is functional, but it would be beneficial to utilize a different format. JSON was chosen for its ability to handle the necessary fields in the ticket and its ease of management in subsequent processing scripts. The JSON ticket should include several fields:

- **name**: name of the package
- **type**: package namespace
- **branches**: list of branches that are needed to be unretired
- **review_bugzilla**: Bugzilla URL with a review on package or none

With this ticket created, the proposal part of automation on the fedpkg side is finished. When the ticket occurs in the system, it will send a Fedora Message, which will trigger processing on the Toddlers side.

Proposal of toddlers plugin

The task of a toddler (an individual plugin for toddlers with a unique responsibility) is to automatically process the ticket. This automation covers all the steps mentioned in the chapter about the current state of the package unretirement process (Chapter 3). The Toddlers tool was chosen because its main responsibility is to subscribe to the message topic and implement event-driven processes as a reaction to new messages appearing in the system. It is a plugin-based system, which makes the process of extension relatively easy.

The idea is to create a new plugin `pdc_unretire_packages`. This plugin, as an input, will get a message, which is represented as a JSON file. JSON schema of this file contains a few fields that are important for the automation process:

- issue title: Title of the ticket that was created. Typically, it is represented as an "Unretire" keyword + name of the package
- issue content: It is just a body of the ticket that contains all the fields mentioned in **proposal fedpkg command** section
- issue full URL: An Issue URL, which is used for commit message for the new commit. It serves as a reference to the provided processing.
- issue id: An ID of issue that were created, it is used to close issue if some errors will appear during processing.

It's important to note that Fedpkg and Toddlers are distinct tools. Although both are part of a single automation, some tests and checks will be duplicated in both. This modular approach allows releng to use either of these tools as a standalone component of Fedora Infrastructure. Additionally, it provides the flexibility to expand this functionality to meet new requirements in the future.

Appendix B displays a diagram of Toddler's processing. Upon receiving a message, Toddler will set up all necessary objects from Fedora tools for automation. This is primarily based on configuration values such as API keys, which are secured from the user.

The next step involves processing the issue. The toddler will process the message issue title to verify that the package actually exists. If not, it will make a comment and close the ticket. If provided data is corrupted, sending a comment and closing a ticket is the standard workflow.

After confirming the validity of the ticket, Toddler begins processing by cloning the package's git repository into a temporary directory. This step is necessary because Toddler runs inside a container, and attempting to clone the repository to a folder inside will result in an error. Subsequently, it will proceed with the action described in the previous chapter 3, handling errors appropriately and completing the task if everything progresses smoothly.

Complete picture of tools collaboration

In this section of the chapter, I will summarize the information, explain the use cases, and explain how the tools interact with each other. The requester sent a request for unretirement by running the Fedpkg command. The request is processed by creating a ticket in a specific ticket tracker. The use case diagram illustrates the process from the RelEng perspective.

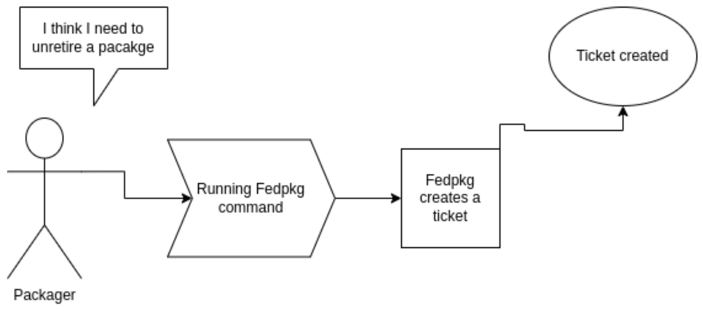


Figure 4.5: Fedpkg use case

Once this ticket appears in the system, it immediately sends a *Fedora Message* with JSON containing all information about the ticket and the data it carries. Toddlers follow every message. It asks its plugins if they are interested in such a ticket. If the answer is positive, the plugin receives the whole message and starts to process it. The following figure shows the RelEng perspective on this process.

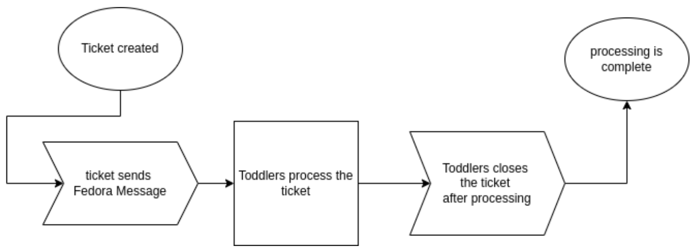

Ticket created
ticket sends Fedora Message
Toddlers process the ticket
Toddlers closes the ticket after processing
processing is complete

Figure 4.6: Toddlers processing

In this way, these tools interact with each other for a common purpose. This interaction does not happen directly but through Fedora Message, which links the system events into a coherent chain of actions.

Chapter 5

Implementation

This chapter examines the details of implementation. It does not describe every line of code but rather reviews interesting points necessary for understanding it. Initially, we will consider the environment and services used during development. A significant part is devoted to developing the tools involved in automation. Finally, the deployment process and the method of testing the code will be considered.

5.1 Enviroment used for development

It was clear that choosing Fedora Linux as the operating system for development was the right decision. The tools that will be utilized are part of Fedora Infrastructure and are designed to work specifically on Fedora Distribution.

For the automation, I have chosen Python as the programming language. Since the tools are already written in Python, the choice was not about the language but about the version. As we will see, further tests will be run on different versions of Python. At the time of writing, the default option was Python 3.10, but the code needs to be compatible with older versions. For this reason, some parts of the code are written using older syntax. Additionally, Python was a natural choice because Fedora Infrastructure has a Python module for almost every service and tool, making it easy to integrate different modules.

Red Hat is the primary sponsor of Fedora Linux. It's important to note that Fedora Infrastructure utilizes Red Hat's software instead of other alternatives. The automation tools are cloud applications, and they use configurations stored in the Ansible repository [1]. This paper does not focus on Ansible, so it won't provide a detailed explanation of the software. In general, Ansible stores roles, which are essentially a list of people authorized to run a specific playbook, and playbooks are collections of actions to be executed on systems.

The *OpenShift* platform was selected for deployment because it is well-optimized for use in Fedora Infrastructure. This thesis will not delve into an in-depth analysis of this platform but rather utilize it as a valuable tool for monitoring and tracking processing logs.

5.2 Fedpkg command implementation

To implement the automation, a new command needs to be created inside the `cli.py` file. The Fedpkg command implementation is divided into 3 parts: declaration of the command,

[1] https://pagure.io/fedora-infra/ansible

correction of arguments for processing, and processing the command. Here is a description of each of them.

Declaration of command

The first part of the command declaration involves creating a parser for a subcommand after fedpkg itself. It also includes adding a description for the command and a help page to assist users in deciding which arguments to provide. This is accomplished by using the `self.subparsers.add_parser()` method, which creates a subparser. Additionally, this section involves parsing arguments to process this command. Here is a detailed description of each argument:

- `-repo`: A string of package name. It is called `repo` because the package name is also a repository name of this package. This argument can be `None`.
- `-namespace`: A string of package namespace. It is used for assembling the package URL in the future. This argument can be `None`, as a default value it sets to `"rpms"`. The requestor can choose a namespace just from the list of district namespaces. This list is received by using `self.get_distgit_namespaces()` help method.
- `-bz_url`: A string of URL on Bugzilla. This URL should contain a link to existing and opened Bugzilla review requests on the package. It can be None, as a default value is used `None`.
- `-b`, `-branches`: A list of branches that the requester would like to Unretire a package on. Each branch inside the list should be separated by a comma. This argument can be `None`, as a default value it's setting `[„rawhide"]`. Rawhide is the last branch in a package; it is used as a development branch, so most requests would like to Unretire it.

Once all the necessary arguments are provided, it's a time to execute a command. Running this command triggers the `self.request_unretirement` method, which contains the logic of the command. The details of this logic will be outlined in the following sections. Here's an example of a command that the requester will use:

```
fedpkg request-unretirement --repo test_repo --namespace tests -b rawhide, f40, f39
```

Correction of arguments for processing command

The following section primarily focuses on checking for an adequate number of arguments without verifying their accuracy. As discussed in the previous chapter (see 4.5), *Fedpkg* is a clever tool that can potentially execute a command even without any arguments. For instance, if the user fails to specify a repository name or namespace, Fedpkg will default to the current repository and assume that the user intends to unretire the specific package in that repository. The text introduces the first error, `rpkgError`, which will be utilized throughout the command logic. It is named as such because **Fedpkg** extends the functionality of the **rpkg** tool. This error will be triggered whenever an issue arises. In the case where the user is not in the repository and fails to provide arguments, this error will be thrown with instructions on what changes are required to execute the command. Following this initial check, a hidden method `self._request_unretirement()` will be invoked to handle the main logic of the command. This hidden method is called with an expanded list of arguments from within Fedpkg. These arguments include:

- `logger`: A logger object.
- `repo_name`: The string of the repo name.
- `ns`: The string of pacakge namespace.
- `branches`: The list of branches that need to be unretired.
- `bugzilla_url`: The URL of the bugzilla review.
  Typically, the value of `self.args.bz_url`, None if not needed.
- `fas_name`: The string of fas name of the user. Typically, the value is `self.cmd.user`.
- `name`: A string representing which section of the config should be used. Typically, the value of `self.name`.
- `config`: A dict containing the configuration, loaded from file. Typically, the value of `self.config`.

Processing the command

This part of *Fedpkg* command is the most important one. It is logically divided into two tasks.

The first part of this method is to carry out the necessary checks and controls. There are a few important questions that need to be checked. Those were mentioned in the chapter with the proposal of a solution (see 4.5). It is better to start with checks for the last commit date on every branch that is required to be Unretired. It is done for a few reasons. If the package was retired more than eight weeks ago, it needs to be re-reviewed by Fedora policies[20]. It is done by opening a ticket in Bugzilla. This is why we are asking the user to provide a *Bugzilla URL* as one of the arguments. So, if the branch was actually retired more than 8 weeks ago and the user didn't provide *Bugzilla URL*, the processing will be automatically canceled, and the requester will be asked to provide it. In order to find out the last commit date, I wrote a helper function.

```
def get_last_commit_date(base_url, namespace, repo_name, branch):
url = "{0}/{1}/{2}.git".format(base_url, namespace, repo_name)
with tempfile.TemporaryDirectory() as temp_dir:
    try:
        repo = git.Repo.init(temp_dir, bare=True)
        repo.git.execute(['git', 'remote', 'add', 'origin', url])
        repo.git.execute(['git', 'fetch', 'origin', branch, '--depth=1'])
        commit_hash = repo.git.rev_parse('FETCH_HEAD')
        commit_date = repo.git.show('-s', '--format=%ct', commit_hash)
        return commit_date
    except git.exc.GitCommandError:
        raise rpkgError("Unable to get last commit date."
                        "Try to check repo name and namespace "
                        "if it exists.")
```

This function will create a temporary directory and fetch all necessary data. As a result it will return a last commit date in a proper format. Based on the difference between today's date and the commit date, it will decide whether **Bugzilla URL** should be provided.

The next check that should be implemented is a check of the requester's rights. In Fedora Infrastructure, we define rights through groups. The user can check his/her/their groups in the Fedora Account system[2] when he/she/they log in to the FAS account. For this purpose, I wrote a helper function:

```
def get_user_groups(username):
    c = fasjson_client.Client("https://fasjson.fedoraproject.org/")
    try:
        user_groups = c.list_user_groups(username=username).result
    except Exception as e:
        return []
    return user_groups
```

This function will create a `fasjsom_client` object. To use this functionality, the user should receive **Kerberos** authentication. The user must have a Fedora Account to be authenticated. Detailed information on how to receive it, the user can be found on this Fedora Wiki page[27]. But in most cases, the unretirement process requests a user familiar with Fedora **Kerberos** authentication. So, for them, it will be enough to run the following command:

```
fkinit -u USERNAME
```

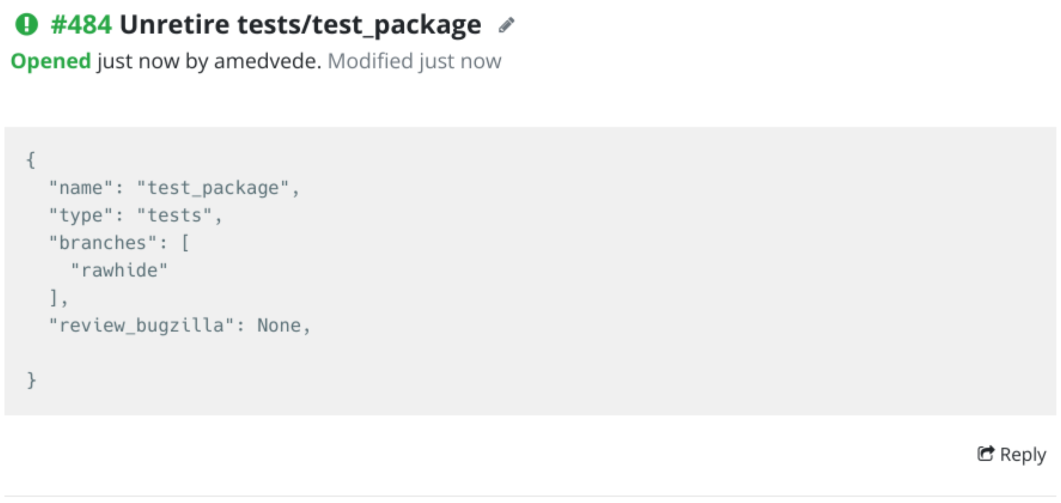
There is just one more check after this. If *Bugzilla URL* was provided and there is a need to check it, the command will use the build-in method that will get the bug (this is the name for Bugzilla tickets, such as review requests and others). This is achieved through the creation of *Bugzilla* client using *Fedpkg* utility for working with **Bugzilla** by this line `bz_client = BugzillaClient(bz_url)`.

The second part of processing the command involves opening a ticket, which is the easy part because it uses the same helper function as different Fedpkg commands that work with tickets, such as `fedpkg request-branch`. In this part of the script, the script simply retrieves fields from the config to define the location where the ticket will be opened and to obtain the correct token. After that, it will form the ticket body and title and create a ticket using this function:

```
new_pagure_issue(
    logger=logger,
    url=pagure_url,
    token=pagure_token,
    title=ticket_title,
    body=ticket_body,
    cli_name=name,
)
```

The result of this script will be an opened ticket in a specific ticket tracker that will look like this 5.1:

[2]https://accounts.fedoraproject.org/



#484 Unretire tests/test_package

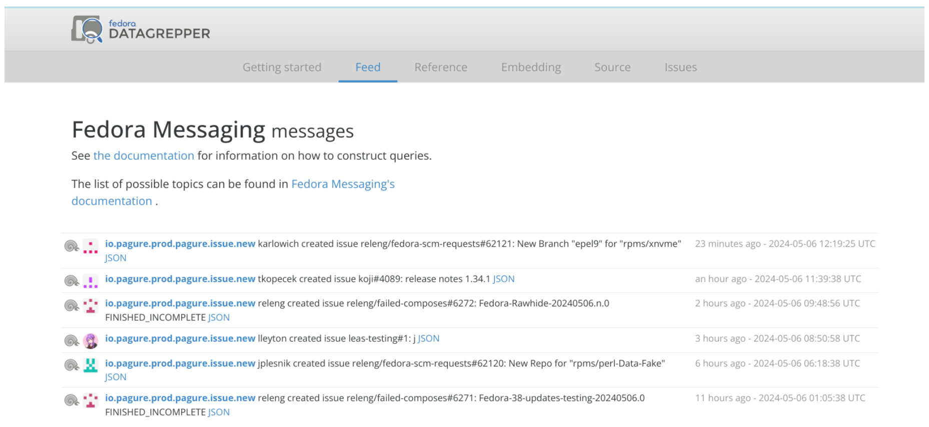
Opened just now by amedvede. Modified just now

```
{
  "name": "test_package",
  "type": "tests",
  "branches": [
    "rawhide"
  ],
  "review_bugzilla": None,

}
```

Figure 5.1: Ticket created by fedpkg command.

The creation of this ticket will automatically send a message. Users can view the messages using a tool called *Datagrepper*. The app's feed follows the message history and also provides an API where users can request a history of messages with specific topics. For example, all issues created on Pagure will send a message with the topic `io.pagure.prod.pagure.issue.new`. Here is an example of what the message history looks like:



Fedora Messaging messages

See the documentation for information on how to construct queries.

The list of possible topics can be found in Fedora Messaging's documentation .

```
io.pagure.prod.pagure.issue.new karlowich created issue releng/fedora-scm-requests#62121: New Branch "epel9" for "rpms/xnvme" JSON    23 minutes ago - 2024-05-06 12:19:25 UTC
io.pagure.prod.pagure.issue.new tkopecek created issue koji#4089: release notes 1.34.1 JSON    an hour ago - 2024-05-06 11:39:38 UTC
io.pagure.prod.pagure.issue.new releng created issue releng/failed-composes#6272: Fedora-Rawhide-20240506.n.0 FINISHED_INCOMPLETE JSON    2 hours ago - 2024-05-06 09:48:56 UTC
io.pagure.prod.pagure.issue.new lleyton created issue leas-testing#1: j JSON    3 hours ago - 2024-05-06 08:50:58 UTC
io.pagure.prod.pagure.issue.new jplesnik created issue releng/fedora-scm-requests#62120: New Repo for "rpms/perl-Data-Fake" JSON    6 hours ago - 2024-05-06 06:18:38 UTC
io.pagure.prod.pagure.issue.new releng created issue releng/failed-composes#6271: Fedora-38-updates-testing-20240506.0 FINISHED_INCOMPLETE JSON    11 hours ago - 2024-05-06 01:05:38 UTC
```

Figure 5.2: Messages history

The creation of a ticket is the logical end of fedpkg command processing. A further section will cover the implementation of the Toddlers plugin, which starts with a ticket.

5.3 Toddler plugin implementation

The *Toddlers* plugin starts to process the tickets as soon as a message occurs.

Structure of Toddlers

The Toddlers tool consists of three main files and a list of plugins. The base class that defines the required methods and attributes of every plugin is written in a file called `base.py`. The base class for every toddler plugin is called `ToddlerBase`. It defines two properties for the plugin and two abstract methods that should be implemented. Appendix C shows the structure of the base class, including the following methods:

- `name`: This property will be redefined as a plugin name.
- `amqp_topics`: This property will define a list of message topics that the plugin will follow.
- `accepts_topic`: This abstract method should be implemented in every plugin to return a boolean value whether this toddler is interested in messages from a specific topic.
- `process`: This abstract method will receive a message and config. It is the method where the processing of the ticket will start in every plugin.

To gain a deeper understanding, it's important to refer to the second key file that outlines the functionalities of toddlers. This file is named `runner.py` and consists of approximately 100 lines of code. Below, I will describe its main functionality without including any code snippets. When the consumer receives a message, it will iterate through each toddler and invoke the `accepts_topic()` method to identify toddlers interested in messages with that particular topic. Subsequently, it will trigger the `process()` function of the selected toddler with the necessary message and configuration for processing.

Start of toddler processing

From this point, my toddlers' plugin starts its functionality. After redefining base class properties, the first part is to ensure that the message with a topic that fits the plugin is actually a message that the plugin is interested in. Unretirement request ticket must have an "Unretire" keyword in a title, so this plugin firstly will ensure it. If the message passes this check, the script will set up a logger object. It is used all over the code to log every activity that is happening. Using a `_log` object is necessary to help RelEng engineers to debug if something goes wrong.

```
_log = logging.getLogger(__name__)
```

The second important step is to set up all necessary objects that will be used during processing based on variables obtained from the config. This includes creating a Pagure object to work with Pagure (This service can be seen as Fedora's Github).

```
self.pagure_io = pagure.set_pagure(config)
```

This object is utilized extensively throughout the code as it provides functionality for working with tickets. Therefore, every error handling process includes closing a Pagure ticket with a request for unretirement along with a proper comment. An example of a typical response to an error could be the following lines of code:

```
_log.info(msg)
self.pagure_io.close_issue(
    issue_id,
    namespace=PROJECT_NAMESPACE,
    message=msg,
    reason="Invalid",
)
return
```

Every error is followed by a specific message, which is logged, and the ticket is closed with the same message. This ensures that the requester will always know the reason why his request has faced a problem and will be able to correct it and reopen a ticket.

After setting up needed objects, the toddler will go to the main part of processing in the `process_ticket()` method.

Ticket processing

The beginning of the `process_ticket()` method involves recognizing the issue title and ensuring that a package with that name exists within the *Fedora Active Packages*. This is crucial for minimizing errors during processing. It's important to note that the *Toddlers* plugin was developed independently of the fedpkg command. This plugin allows for extending its functionality or adding new types of tickets that it can process. Because of this, some of the checks and tests performed on the fedpkg side will also be repeated on the Toddlers side.

The main part started with cloning the git repository into the temporary directory to work with it. This git repository of the package is placed on a service that is called *Pagure dist-git*. It is something like the storage of spec files for the upstream packages.

After cloning the repository, the script will get a list of branches from the message. The retired branch is marked by the fact that it does not store any file other than `dead.package`. So before making any changes, the script will ensure that every branch is really having `dead.package` file. This avoids the possibility that an attacker will use this functionality to try to get rid of the last commit.

The following method will re-evaluate the checks performed on the Fedpkg side and conduct additional tests.

```
if not self._is_package_ready_for_unretirement(
    issue_id=issue_id, issue_body=issue_body
):
    return
```

Is the package ready for unretirement? It sounds quite obvious; the reason for that is that this method actually contains three more checks. Some of them will repeat the checks that were made on the fedpkg side. Those checks are:

- Verify that the package wasn't retired for a reason: It is very important to keep the package retired if it has major reasons for it, such as license issues. This test will get the last commit message. If this message contains any forbidden words such as

„legal“ or „license,“ it will stop the Unretirement process until future investigation if this process can be done for this package.

- Verify if Bugzilla needs to be checked: This test will mostly repeat the test that was processed on the fedpkg side; it will also get information about the last commit date and, by comparing it with today's date, will decide if Bugzilla request needs to be checked.

- Verify bugzilla ticket: This test will ensure that if providing Bugzilla is required, then the Bugzilla ticket should have a `fedora_review+` flag.

After processing all tests, the script will start the process of unretirement described in chapter 3. It starts by reverting the last commit. This part will revert the last commit and create a new one. The new commit message will contain the link to the issue that was opened so that people who look at the history of commits will understand why this was done (see 3.6).

The next part involves unblocking tags (this is how we refer to branches on Koji) on Koji.

```
_log.info("Unblocking tags on Koji.")
if self._is_need_to_unblock_tags_on_koji(tags_to_unblock, package_name):
    self._unblock_tags_on_koji(issue_id, tags_to_unblock, package_name)
```

In the first line, it will check if the task needs to be done, as tags may already be unblocked on Koji in some cases. This is done using a specific object for working with Koji, `self.koji_session`. This object provides methods for retrieving data from Koji. In this case, I'm using `package_tags = self.koji_session.listTags(package=repo)` to get a list of tags associated with the package. If the package doesn't exist on Koji or if it doesn't have certain tags, the script will raise an error with the appropriate message.

The second line is actually about unblocking tags on Koji, and it also uses functionality provided by `self.koji_session`. In this case, the script will iterate through each tag and call the following method:

```
self.koji_session.packageListUnblock(taginfo=tag, pkginfo=repo)
```

After unblocking tags on Koji, the next step is to ensure that unretired packages won't become orphaned. An orphaned package is one that doesn't have a maintainer. It's common for a package to become retired after losing its maintainer. If the package becomes orphaned, the requester's FAS will be used as the maintainer for the package. The following lines outline the process.

```
if self.pagure_io.is_project_orphaned(
    namespace=namespace, repo=package_name
):
    self.pagure_io.assign_maintainer_to_project(
        namespace=namespace, repo=package_name, maintainer_fas=issue_opener
    )
```

Finally, the logical finish of processing is adjusting the *End Of Life (EOL)* on the *Product Definition Center (PDC)*. PDC is a center that contains metadata about packages. This tool will be discussed in the last chapter about future steps7. I wrote a utility that helps to adjust EOL on PDC. The syntax is quite strange, which is due to PDC being an outdated tool that has lost its maintainer. The code is as follows:

```python
def adjust_eol(global_component, component_type, branch, eol):
    """
    Adjusting eol of branch.

    Params:
        global_component: A sting name of the global component
        component_type: A string Type of component.
        branch: A string name of branch.
        eol: A string with date.
    """
    existing_branch_slas = get_branch_slas(global_component, component_type, branch)

    if existing_branch_slas is None:
        return

    pdc = get_pdc()

    payload = {"eol": eol, "branch_active": True}

    for branch_sla in existing_branch_slas:
        pdc["component-branch-slas"][branch_sla["id"]]._(payload)
```

The function is designed to request branch SLAs (Service Level Agreements) and update the SLA for every active branch based on Bodhi end-of-life dates for each release. This is the final step in the automation process; the package is now fully returned to the *Fedora Active Packages* list. However, the question remains: how will people use this? Deployment of these tools to the server is required for this purpose, and the next section will address this process.

5.4 Deployment

The Fedora Infrastructure has two types of servers for its tools and services: the Production server and the Staging server. The staging server is a copy of the production server and is used for long-term testing of changes to stabilize them and fix most of the bugs and errors that occur. To run the deployment process, the engineer should modify the *Ansible* configuration and add themselves to the role file to be able to run the playbook. This process can take some time while the configurations are updated, and after that, the engineer will be able to connect to the specific server and run the deployment process.

Toddlers is an *Openshift* app. That means Toddlers is running inside a container in Openshift. Fedpkg, on the other hand, is not, and deployment of a new Fedpkg version requires the Fedpkg maintainer to make a new patch. Because we have a lack of maintainers, at the moment of writing, the last commit for Fedpkg was accepted 3 months ago, and there is no exact day when the changes will appear in the system.

To deploy Toddlers, a pull request must first be merged into the staging branch, which exists to test changes that will go to production after some time. When the pull request is merged, it's time to connect to the `batcave01` server, which is used to run playbooks. To connect to the server, you need to run the following command:

```
ssh FAS_NAME@batcave01.iad2.fedoraproject.org
```

Your `FAS_NAME` is used to identify yourself. It is not necessary if the name of your computer is the same as your FAS.

Once on the server, it's time to run a playbook, accomplished by running the following command on the server. The option `-l "staging"` defines the type of server that new changes must be deployed to.

```
sudo rbac-playbook openshift-apps/toddlers.yml -l "staging"
```

To get sudo permission, you must provide your FAS password and your *FreeOTP token*. This token is used every time you try to log in or make any changes.

When the command finishes processing, the person who ran it will receive an email notification. After that, the new version of the tool will be deployed on Openshift.

5.5 Tests

The Fedora Project is an open-source distribution that imposes certain obligations regarding the quality of the code being added. This commitment includes covering new code with unit tests, an important feature to minimize the chance of bugs during deployment and ensure the quality of the code. This section is dedicated solely to the implementation of unit tests. The percentage of code coverage is used to estimate their success. Finally, we will describe the step-by-step process of running tests to verify code coverage.

Implementation of unit tests

Before diving into the implementation details, it's important to discuss test automation using the tools *Fedpkg* and *Toddlers*. For testing, we are utilizing a tool called *Tox*[3]. This tool aims to automate and standardize testing in Python and can be run manually or during the CI pipeline. Its functionality is determined by the config file `tox.ini`, which defines the list of environments and specifics for running tests in different environments, such as the list of Python versions the code should be executable on, typically three different Python versions for *Fedora Infrastructure* tools. An example of a configuration file for tests is listed in appendix D.

In the implementation of tests, I used the popular Python module `unittest`. For my Toddlers plugin, I created a file containing tests for the code,
named `test_pdc_unretirement_packages.py`. For each method and function, I created a test class that covers it. For example, for the method:

```
_is_package_ready_for_unretirement()
```

I created a class called:

```
class TestIsPackageReadyForUnretirement:
```

After creating the class, it must have a `setup_method()` which will *mock* some objects that will be used inside the function or method. The typical unit test should cover several cases. The main one is the successful processing of the function. The test must ensure that if the function or method receives the proper data, it will process without any errors. Other tests must cover every exceptional state that can occur inside the function.

[3]https://tox.wiki/en/4.15.0/

Chapter 6

Testing

The whole functionality is divided between two tools, **Fedpkg** and **Toddlers**, and even in those tools, the functions that I wrote are distributed between different files. Some of the helper functions were placed under the utility module, and some of them were written under the modules that work with other tools and services of Fedora Infrastructure, so the whole testing of functionality was problematic.

Another problem I encountered is that different tools require different rights, and I don't have some of those rights because of security issues. So, further testing processes will be divided into two sections dedicated to individual tools.

6.1 Running the tests

Here will be described how to run unit tests. It's an easy process that will require just a few steps.

1. Creating Fedora Account: The user must have a Fedora Account. The registration process is easy and everyone can do it.

2. Adding SSH Key: The user should generate an SSH Key and add this key to the specific field in his profile.

3. Cloning the repository: Everyone who has a FAS account is able to clone a repository placed on *Pagure*[1]. The user should find a repository of tools that he is interested in. After that, press the **clone** button and get the ssh link for the repository. By using the following command, the user will clone the repository locally: `git clone SSH_URL`.

4. Running tests: Move to the root folder of the repository and run `tox` command; it will last approximately 10 minutes to test the whole code.

5. Additional steps: User can set up different **tox** options to generate *HTML* page that shows covered methods, for example.

After those steps, the user can see the code's coverage in a terminal. Running such tests is the best way to ensure that the code processes as it should during development.

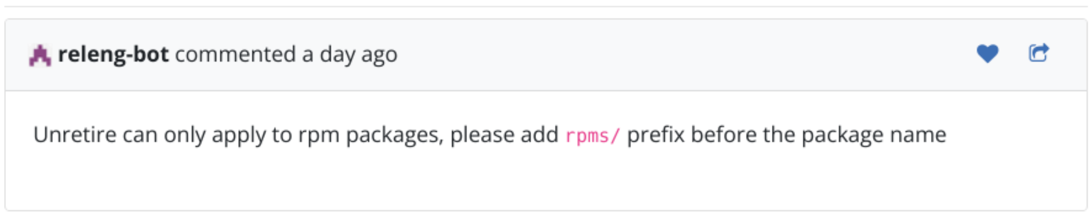
[1]https://pagure.io/

6.2 Fedpkg testing

The best way to test its functionality is to run unit tests. This is the main way how to ensure that it works. The *Fedpkg* project lacks maintainers, so many Pull Requests can lay unreviewed there for a long time; the same happened with my Pull Request. The best way to test is to wait until a new patch for this tool comes out, which will contain my changes as well. Just after that, Fedora RelEng engineers will start to motivate packagers to use this command instead of opening tickets themself. The sign that shows the functionality of this command is that other commands that create a ticket and use a similar command for that are working. Also, it has the checks, most of which will repeat on **Toddlers** side. This doesn't ensure that the command is working perfectly, but increasing the probability of it on a high level.

6.3 Toddlers Testing

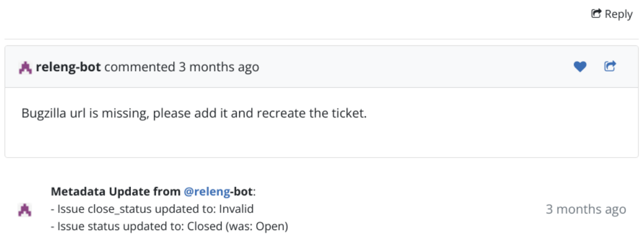
The toddler plugin is thoroughly covered by unit tests using the *Tox* tool. This tool displays the percentage of coverage for individual files and can also generate HTML with marked lines that have been tested. To ensure through testing, any changes made on the Toddlers side are first deployed on a *Staging* server, which serves as a playground for working with packages without affecting the production environment. While the staging server is useful, it's not perfect. Testing at different stages can help reduce the occurrence of bugs, but it doesn't guarantee a bug-free result. Ultimately, only time can provide the perfect result. I attempted to create tickets for any issues and monitored the bot's responses to them. Here are some of the comments that the bot left:


releng-bot commented a day ago

Unretire can only apply to rpm packages, please add `rpms/` prefix before the package name

Figure 6.1: Package doesn't have an "rpms" namespace.

This comment will be left, and the ticket will be closed if the requester tries to Unretire a package without a namespace or a package with a different namespace than "rpms". Because the Unretirement can be processed just on "rpms" packages.



releng-bot commented 3 months ago

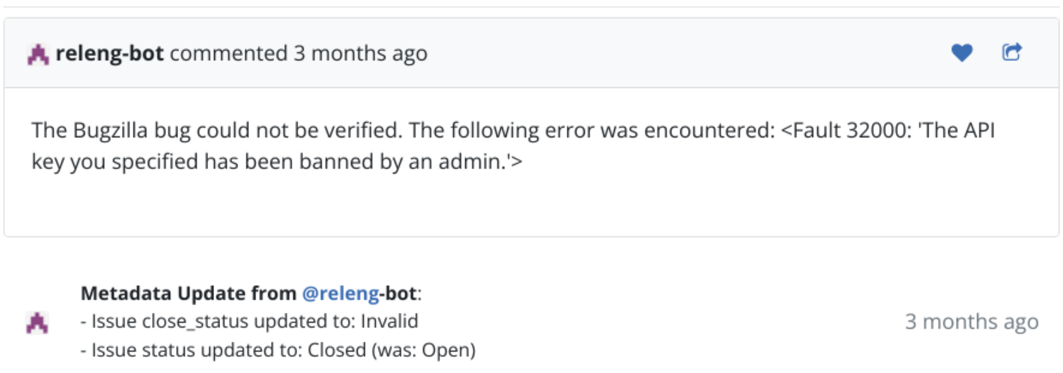
Bugzilla url is missing, please add it and recreate the ticket.

Metadata Update from @releng-bot:
- Issue close_status updated to: Invalid
- Issue status updated to: Closed (was: Open)

3 months ago

Figure 6.2: Bugzilla URL is missing, error handling.

This comment will be printed if the package was previously retired more than eight weeks ago and the other requester didn't provide any URLs in the ticket.


releng-bot commented 3 months ago

The Bugzilla bug could not be verified. The following error was encountered: <Fault 32000: 'The API key you specified has been banned by an admin.'>

Metadata Update from @releng-bot:
- Issue close_status updated to: Invalid
- Issue status updated to: Closed (was: Open)

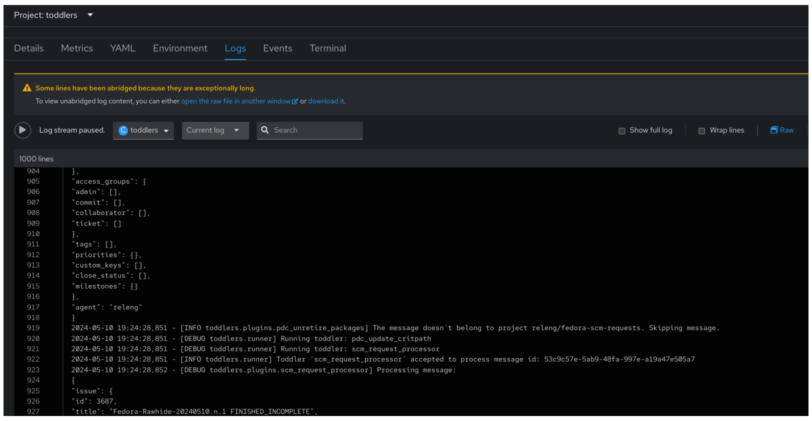
3 months ago

Figure 6.3: Bugzilla URL is missing, error handling.

This comment shows that the development of new features is a permanent process; this happened because I made a mistake in the code. This mistake was that I forgot that *Bugzilla* also has a staging version, and my code wasn't ready for this. Bugzilla's staging and production servers require different API keys, so I changed them in the code in updates.

6.4 Monitoring

Fedora Infrastructure allows engineers to monitor events that are happening with Openshift apps, such as Toddlers. The processing can be inspected through the *Openshift* web client. It makes it easy to follow logs and events that are happening inside the tool. The example of *Toddles* log is on the following figure6.4.



Some lines have been abridged because they are exceptionally long.
To view unabridged log content, you can either open the raw file in another window or download it.

```
},
"access_groups": {
"admin": [],
"commit": [],
"collaborator": [],
"ticket": []
},
"tags": [],
"priorities": {},
"custom_keys": [],
"close_status": [],
"milestones": {}
},
"agent": "releng"
}
2024-05-10 19:24:28,851 - [INFO toddlers.plugins.pdc_unretire_packages] The message doesn't belong to project releng/fedora-scm-requests. Skipping message.
2024-05-10 19:24:28,851 - [DEBUG toddlers.runner] Running toddler: pdc_update_critpath
2024-05-10 19:24:28,851 - [DEBUG toddlers.runner] Running toddler: scm_request_processor
2024-05-10 19:24:28,851 - [INFO toddlers.runner] Toddler 'scm_request_processor' accepted to process message id: 53c9c57e-5ab9-48fa-997e-a19a47e505a7
2024-05-10 19:24:28,852 - [DEBUG toddlers.plugins.scm_request_processor] Processing message:
{
"issue": {
"id": 3687,
"title": "Fedora-Rawhide-20240510.n.1 FINISHED_INCOMPLETE",
```

Figure 6.4: Example of Toddlers log.

6.5 Saved Time

The primary goal of automation is to save human time and free up resources. It is difficult to determine the exact amount of time saved, as the processing of Unretirement requests involves different engineers and varies in time. Let's estimate that one request takes an average of 15 minutes, and there have been 16 requests in the last month. This number is not precise and is based on the previous month's data at the time of writing, so it will be used as an estimate. After calculations, we find that this automation saves 4 hours monthly for the engineer. While this may not seem like much, this time is still significant. We encourage you, as a potential contributor, to join the automation process and help save even more time, as there is nothing more valuable than time.

Chapter 7

Conclusion

The main goal of this thesis was to design and implement an extension for the tools that are used in Fedora Infrastructure. Those extensions cover automation of the *Unretirement SOP*. The result was achieved by adding a new *Fedpkg* command that creates a ticket with an Unretirement request and by adding a new plugin for *Toddlers* tool, which follows the ticket tracker with such requests and processes them.

In the theoretical part, I learned about the Fedora package lifecycle and how the RelEng team controls it. To share knowledge, RelEng invented SOPs, which are guidelines for processing popular tasks. Some of them were fully automated, that automation was integrated into the RelEng work pipeline, but some of them were not. After some learning about them, my interest was tied by an *Unriterement SOP*. I heard that people in our team are quite angry that they still need to process it manually. At this point, I made my choice. By discussing the problem with my peer, he pointed out the tool that potentially can help in automation. This tool was *Toddlers*. At the start of writing, my plan was to write an extension just for it. But the problem was that it wouldn't be able to cover all of the requests because requestors were used to the person processing it, so they didn't think much about formatting a ticket. The solution was to extend this thesis by adding here automation of ticket creation on *Fedpkg* side.

In this work, I learned how different services are connected within the Fedora Infrastructure. I also learned about the whole Fedora community and understood how open-source development happens. Different people maintain different services, so updates in some of them can last a relatively long time.

In the future, some of those tools will be changed. For example, the last part of *Toddlers* processing is updating EOL on PDC. But PDC is an outdated tool that we will get rid of in the future; some of its endpoints will be moved to our different services, and this part of the code will be implemented. Also, we, as a *RelEng* team, would like to rewrite toddlers in the future. The reason for this is that if one of the *Toddlers* will be cycled, all of the plugins won't be able to work, so the idea is to add individual runners for every toddler. The *Fedpkg* tool is more stable in comparison with *Toddlers*, and changes are happening there really rarely.

Bibliography

[1] *Datagrepper's webpage* [online]. [cit. 2024-04-23]. Available at: https://apps.fedoraproject.org/datagrepper/.

[2] *Fedpkg repository* [online]. [cit. 2024-04-28]. Available at: https://pagure.io/fedpkg.

[3] *Product Definition Center* [online]. Red Hat, Inc. and others. [cit. 2024-01-16]. Available at: https://pdc.fedoraproject.org/.

[4] *Red Hat Bugzilla Website* [online]. Red Hat, Inc. and others. [cit. 2024-01-18]. Available at: https://bugzilla.redhat.com/.

[5] *Releng repository* [online]. Red Hat, Inc. and others. [cit. 2024-01-15]. Available at: https://pagure.io/releng.

[6] *Toddlers repository* [online]. [cit. 2024-04-24]. Available at: https://pagure.io/fedora-infra/toddlers.

[7] *Fedora Package Life Cycle notes* [online]. Red Hat, Inc. and others., 20. april 2015 [cit. 2024-01-18]. Available at: https://fedoraproject.org/wiki/Fedora_Package_Lifecycle_notes.

[8] *Branching* [online]. Red Hat, Inc. and others., 2016 [cit. 2024-01-18]. Available at: https://docs.pagure.org/releng/sop_branching.html.

[9] *Fedora Release Engineering* [online]. Red Hat, Inc. and others., 2016 [cit. 2024-01-19]. Available at: https://docs.pagure.org/releng/.

[10] *Fedora Release Engineering Overview* [online]. Red Hat, Inc. and others., 2016 [cit. 2024-01-20]. Available at: https://docs.pagure.org/releng/overview.html.

[11] *Fedora Release Engineering SOPs* [online]. Red Hat, Inc. and others., 2016 [cit. 2024-01-20]. Available at: https://docs.pagure.org/releng/sop.html.

[12] *Retire Orphaned Packages* [online]. Red Hat, Inc. and others., 2016 [cit. 2024-01-18]. Available at: https://docs.pagure.org/releng/sop_retire_orphaned_packages.html.

[13] *Unretiring a package branch* [online]. Red Hat, Inc. and others., 2016 [cit. 2024-01-17]. Available at: https://docs.pagure.org/releng/sop_unretire.html.

[14] *Fedora Release Criteria* [online]. Red Hat, Inc. and others., 12. october 2017 [cit. 2024-01-19]. Available at: https://fedoraproject.org/wiki/Fedora_Release_Criteria.

[15] *Milestone freezes* [online]. Red Hat, Inc. and others., 12. october 2017 [cit. 2024-01-18]. Available at: https://fedoraproject.org/wiki/Milestone_freezes.

[16] *Modelling Dependencies* [online]. Red Hat, Inc. and others., 29. april 2017 [cit. 2024-01-18]. Available at: https://fedoraproject.org/wiki/Infrastructure/Factory2/Prehistory/ModellingDeps.

[17] *Fedora Messaging* [online]. Red Hat, Inc. and others., 2018 [cit. 2024-01-19]. Available at: https://fedora-messaging.readthedocs.io/en/stable/.

[18] *The Community Platform Engineering Team* [online]. Fedora Project, 11. august 2019 [cit. 2024-01-19]. Available at: https://docs.fedoraproject.org/en-US/cpe/.

[19] *Release Validation Test Plan* [online]. Red Hat, Inc. and others., 06. august 2020 [cit. 2024-01-21]. Available at: https://fedoraproject.org/wiki/QA:Release_validation_test_plan.

[20] *Policy for Orphan and Retired Packages* [online]. Fedora Project, 07. september 2021 [cit. 2024-01-20]. Available at: https://docs.fedoraproject.org/en-US/fesco/Policy_for_orphan_and_retired_packages/.

[21] *Working with Community platform engineering* [online]. Fedora Project, 01. june 2021 [cit. 2024-01-18]. Available at: https://docs.fedoraproject.org/en-US/cpe/working_with_us/.

[22] *End Of Life* [online]. Fedora Project, 16. august 2023 [cit. 2024-01-15]. Available at: https://docs.fedoraproject.org/en-US/infra/release_guide/release_eol/.

[23] *Fedora Messaging* [online]. Red Hat, Inc. and others., august 2023 [cit. 2024-03-20]. Available at: https://readthedocs.org/projects/jcline-fedmsg/downloads/pdf/reorg-docs/.

[24] *Fedora Messaging* [online]. Fedora Project, 2023 [cit. 2024-01-22]. Available at: https://docs.fedoraproject.org/en-US/infra/developer_guide/messaging/#_messaging.

[25] *Fedora Packaging Guidelines* [online]. Fedora Project, 23. november 2023 [cit. 2024-01-17]. Available at: https://docs.fedoraproject.org/en-US/packaging-guidelines/.

[26] *Fedora Release Engineering Troubleshooting Guide* [online]. Fedora Project, 04. april 2023 [cit. 2024-01-16]. Available at: https://docs.fedoraproject.org/en-US/infra/releng_misc_guide/troubleshooting.

[27] *Kerberos Wiki* [online]. Red Hat, Inc. and others., 11. february 2023 [cit. 2024-04-27]. Available at: https://fedoraproject.org/wiki/Infrastructure/Kerberos.

[28] *Package Retirement Process* [online]. Fedora Project, 01. june 2023 [cit. 2024-01-19]. Available at: https://docs.fedoraproject.org/en-US/package-maintainers/Package_Retirement_Process.

[29] *Package Review Process* [online]. Fedora Project, 07. september 2023 [cit. 2024-01-19]. Available at: https://docs.fedoraproject.org/en-US/package-maintainers/Package_Review_Process/.

[30] *Package Update Guide* [online]. Fedora Project, 22. november 2023 [cit. 2024-01-16]. Available at: https://docs.fedoraproject.org/en-US/package-maintainers/Package_Update_Guide/#branched_milestone_freezes.

[31] *Updates Testing* [online]. Red Hat, Inc. and others., 21. november 2023 [cit. 2024-01-14]. Available at: https://fedoraproject.org/wiki/QA:Updates_Testing.

[32] *AMQP 0-9-1 Model Explained* [online]. Broadcom Inc., 2024 [cit. 2024-04-28]. Available at: https://www.rabbitmq.com/tutorials/amqp-concepts#what-is-amqp.

[33] *Cheat Cubes* [online]. Fedora Project, 13. may 2024 [cit. 2024-04-25]. Available at: https://docs.fedoraproject.org/en-US/commops/design-assets/cheat-cubes/.

[34] *Fedora Linux* [online]. Wikipedia, The Free Encyclopedia., 2024 [cit. 2024-01-20]. Available at: https://en.wikipedia.org/w/index.php?title=Fedora_Linux&oldid=1193071744.

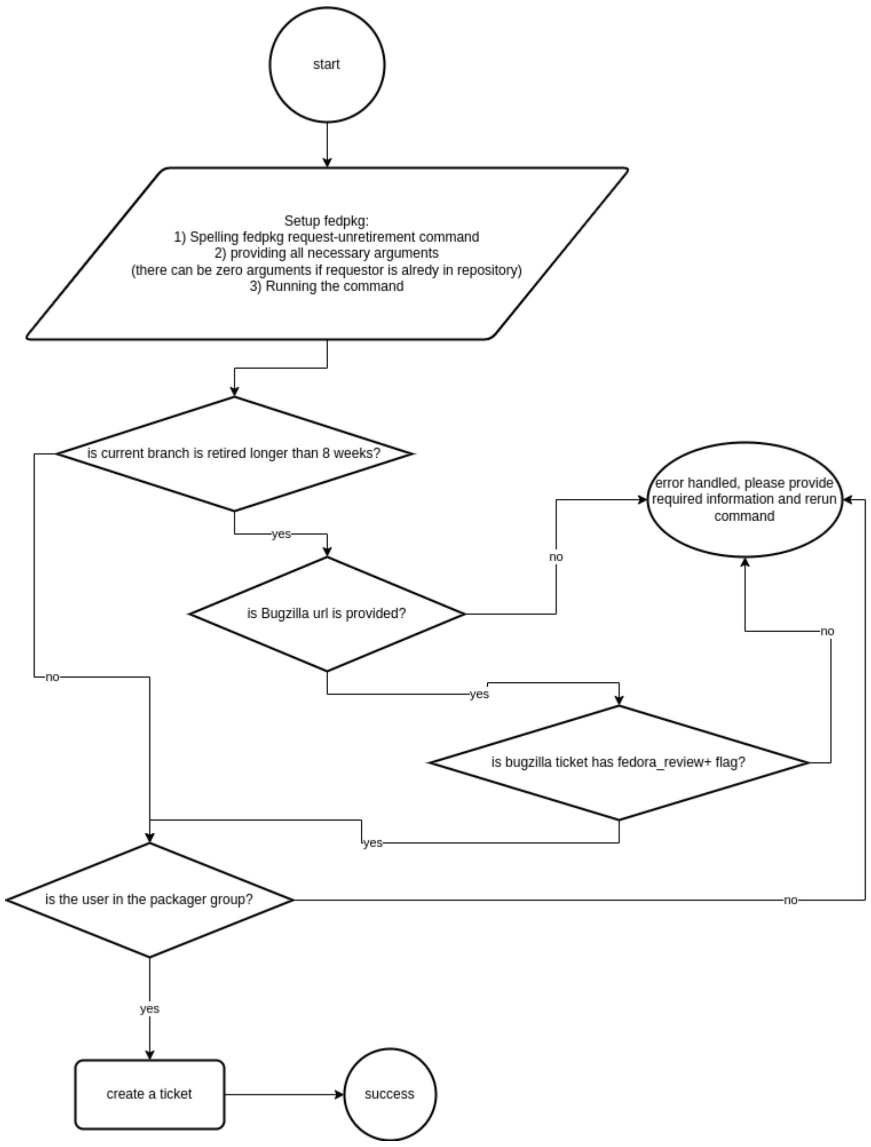
[35] *Fesco Updated Policy* [online]. Fedora Project, 22. january 2024 [cit. 2024-01-17]. Available at: https://docs.fedoraproject.org/en-US/fesco/Updates_Policy/.

[36] Klíč, K. *Fedpkg presentation* [online]. [cit. 2024-04-23]. Available at: https://fedoraproject.org/w/uploads/1/1c/Fedpkg-presentation.pdf.

[37] Mike McLean, D. G. *Koji* [online]. Red Hat, Inc. and others., 2017 [cit. 2024-01-20]. Available at: https://docs.pagure.org/koji.

Appendix A

Fedpkg command Diagram



start
Setup fedpkg:
1) Spelling fedpkg request-unretirement command
2) providing all necessary arguments
(there can be zero arguments if requestor is alredy in repository)
3) Running the command
is current branch is retired longer than 8 weeks?
yes
no
is Bugzilla url is provided?
no
error handled, please provide required information and rerun command
yes
is bugzilla ticket has fedora_review+ flag?
no
yes
is the user in the packager group?
no
yes
create a ticket
success

Figure A.1: Fedpkg command processing diagram

Appendix B

Toddlers plugin diagram

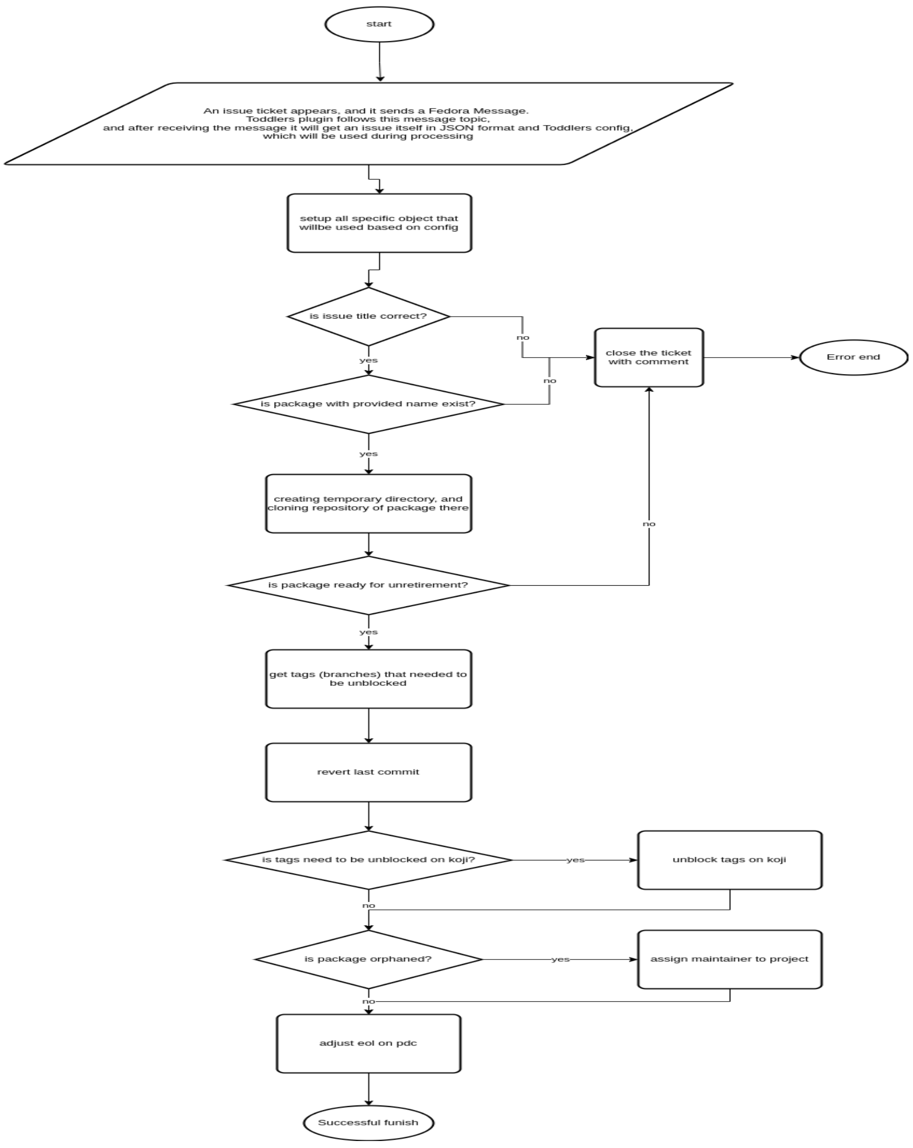

start
An issue ticket appears, and it sends a Fedora Message.
Toddlers plugin follows this message topic,
and after receiving the message it will get an issue itself in JSON format and Toddlers config,
which will be used during processing
setup all specific object that willbe used based on config
is issue title correct?
no
yes
close the ticket with comment
Error end
no
is package with provided name exist?
yes
creating temporary directory, and cloning repository of package there
no
is package ready for unretirement?
yes
get tags (branches) that needed to be unblocked
revert last commit
is tags need to be unblocked on koji?
yes
unblock tags on koji
no
is package orphaned?
yes
assign maintainer to project
no
adjust eol on pdc
Successful funish

Figure B.1: Toddler plugging processing diagram.

Appendix C

Toddlers Base class

```
class ToddlerBase(object):
    __metaclass__ = abc.ABCMeta

    @property
    @abc.abstractmethod
    def name(self):
        """Returns name of the plugin."""
        return "base"

    @property
    @abc.abstractmethod
    def amqp_topics(self):
        """Returns the list of topics of interest for this toddler in a format
        that can be used directly when connecting to amqp.
        For example, it supports items like:
            ``org.fedoraproject.#.buildsys.build.state.change``
        which is valid when subscribing to a queue in amqp but will not allow
        string based comparison with the topic extracted from the message.

        """
        return []

    @abc.abstractmethod
    def accepts_topic(self, topic):
        """Returns a boolean whether this toddler is interested in messages
        from this specific topic.
        """
        return

    @abc.abstractmethod
    def process(self, config, message):
        """Process a given message."""
        return
```

Appendix D

Tox configuration for Toddlers

```
[tox]
envlist = black,mypy,flake8,py3{9,10,11}
# If the user is missing an interpreter, don't fail
skip_missing_interpreters = True
skipsdist = True

[testenv]
deps =
    -r requirements.txt
    -r test-requirements.txt
sitepackages = True
setenv =
    PYTHONPATH={toxinidir}
commands =
    pytest {posargs}

[testenv:black]
deps =
    black
sitepackages = False
commands =
    black --check --diff .

[testenv:mypy]
basepython = python3.11
deps =
    {[testenv]deps}
    mypy
setenv =
    {[testenv]setenv}
commands = mypy --config-file {toxinidir}/mypy.cfg toddlers tests

[testenv:flake8]
deps =
    flake8
    flake8-import-order
sitepackages = False
commands =
    flake8 --ignore=W503 toddlers/ tests/ {posargs}
```