Comprehensive Feasibility Study and Technical Migration Specification: PTC Codebeamer to Tuleap Community Edition

1. Executive Feasibility Assessment

1.1 Strategic Context and Scope

The transition from PTC Codebeamer, a proprietary, enterprise-grade Application Lifecycle Management (ALM) solution, to Tuleap Community Edition (CE), a libre open-source ALM platform, represents a profound shift in infrastructure strategy. This report evaluates the feasibility of this migration not merely as a data transfer exercise, but as a complex systems integration project requiring significant architectural re-engineering.

The feasibility of this migration is strictly classified as **Technically Possible but Operationally Complex**. Unlike migrations between homogenous systems, moving from Codebeamer to Tuleap CE involves bridging two fundamentally different architectural philosophies: Codebeamer's rigid, object-oriented hierarchy versus Tuleap's flexible, link-based artifact ecosystem. Furthermore, the restriction to Tuleap's *Community Edition* significantly alters the feasibility landscape. While Tuleap Enterprise Edition offers commercial support and advanced import tools (such as native Jira or ReqIF importers), the Community Edition requires a "do-it-yourself" approach, necessitating the development of a custom Extract-Transform-Load (ETL) middleware.[1]

This document outlines a high-effort, high-fidelity migration path relying on custom Python development. Organizations lacking internal DevOps maturity or Python proficiency are advised against this path, as Tuleap CE lacks the "wizard-based" import tools found in its

Enterprise counterpart.[3]

1.2 The Feasibility Matrix

The following analysis breaks down feasibility across critical dimensions, weighing the capabilities of Codebeamer against the constraints of Tuleap CE.

Feasibility Dimension	**Status**	**Critical Analysis**
Data Structure API	**Low**	Codebeamer allows deep introspection of schemas via Swagger/OpenAPI.[4] However, Tuleap CE's REST API explicitly lacks endpoints to *create* Trackers or fields programmatically.[5] This forces a "Structure-First" approach where target projects must be manually configured before data injection.
Artifact Injection	**High**	Once the structure exists, Tuleap's POST /artifacts and PUT /artifacts/{id} endpoints are robust, allowing the creation of items with complex field values, text bodies, and links.[6]
Hierarchy Migration	**Medium**	Codebeamer uses inherent containment (Folders/Items). Tuleap 12.5+ uses a typed link system (_is_child).[8]

		Migration requires a two-pass algorithm: first creating all nodes, then knitting them together with links.
History Fidelity	**Critical Risk**	The Tuleap REST API creates artifacts with the current timestamp and the API key owner as the author.[8] It is largely impossible to natively "backdate" changesets in Tuleap CE without direct database manipulation (highly discouraged). The "Submitter" can sometimes be masqueraded via specific administrative configurations, but generally, historical audit trails will be flattened into a static text field (e.g., "Legacy History") rather than native system events.
Rich Text/Wiki	**Medium**	Codebeamer uses a proprietary Wiki markup or HTML.[11] Tuleap uses CommonMark (Markdown) or HTML.[12] A conversion layer (e.g., html2text or pandoc) is required in the ETL pipeline.

1.3 Economic and Resource Implications

While the licensing cost of Tuleap CE is zero, the Total Cost of Ownership (TCO) for the migration is front-loaded into engineering hours.

- **Infrastructure:** Tuleap CE requires a dedicated RHEL/Rocky Linux 9 server.[1]
- **Development:** A senior engineer must write, test, and validate the Python ETL scripts (estimated 80-160 hours for complex projects).
- **Maintenance:** Without Enterprise support, the organization assumes full liability for patches, upgrades, and API breakages.[2]

1.4 Conclusion on Feasibility

The migration is **feasible** under the condition that the organization accepts a "Snapshot Migration" rather than a "Full History Replay." The target Tuleap environment will contain the correct current state of requirements, bugs, and tests, but the granular history of *who changed what field five years ago* will likely be archived in a read-only format rather than native Tuleap history. If this trade-off is acceptable, the technical path described below provides a robust solution.

---

2. Architectural Gap Analysis: Source vs. Target

Understanding the dissonance between PTC Codebeamer and Tuleap CE is prerequisite to writing the migration code. The "impedance mismatch" between the two data models dictates the transformation logic.

2.1 Data Model Ontology

PTC Codebeamer (The Source)
Codebeamer organizes data hierarchically:

- **Project:** The root container.
- **Tracker:** A type definition (e.g., "Customer Requirement," "Bug"). Trackers have strict schemas.
- **Tracker Item:** The unit of data. Items can be parents of other items within the *same* tracker or *different* trackers.
- **Associations:** Loose links between items (e.g., "depends on," "derived from").[13]
- **Baselines:** Snapshots of projects at a point in time. (Note: Baselines are generally **not**

migratable to Tuleap CE via API and must be archived as documents).

Tuleap CE (The Target)
Tuleap's model is flatter but highly relational:

- **Project:** The root container.
- **Tracker:** A customized collection of Artifacts.
- **Artifact:** The unit of data (analogous to Codebeamer Item).
- **Artifact Links:** The primary method for hierarchy and relationships.
- **Tracker Hierarchy:** Historically, Tuleap defined hierarchy at the Tracker level (e.g., "Epics contain Stories"). In modern versions (12.5+), this is more flexible, utilizing the _is_child link type between artifacts regardless of strict tracker parentage configurations.[8]

Gap Identification:
The primary gap is the Fields Definition. Codebeamer API v3 provides a schema endpoint to read field definitions.14 Tuleap CE does not allow writing field definitions via API.5 Therefore, the ETL script cannot say "Create Field 'Severity' with values 'High/Medium/Low'". Instead, the script must say "Find Field ID corresponding to 'Severity' and insert value ID corresponding to 'High'".7

2.2 Identification of Migratable Assets

Not all data in Codebeamer can move to Tuleap CE.

Asset Type	**Migratability**	**Target Destination in Tuleap**
Requirements	Yes	"Requirements" Tracker (Artifacts)
Test Cases	Yes	"Test Cases" Tracker (Artifacts) or TTM [15]
Bugs/Tasks	Yes	"Bugs"/"Tasks" Trackers (Artifacts)
Rich Text Desc.	Yes (Transformed)	Artifact Text Field (Markdown/HTML)

Attachments	Yes	Artifact File Field (via Temp File Upload)
Comments	Partial	"Follow-up" comments (Flattened history)
Users	Partial	Must exist in Tuleap directory (LDAP/Local)
Baselines	No	N/A (Archive as PDF/Doc export)
Tracker Config	No	Manual Recreation Required

---

3. Pre-Migration Strategy: The "Structure-First" Approach

Because Tuleap's API cannot create trackers, the migration depends on a rigorous "Structure-First" preparation phase. This involves manually creating the "skeleton" of the project in Tuleap before pouring in the "flesh" (data) from Codebeamer.

3.1 Target Environment Preparation (Manual Configuration)

The Systems Administrator must perform the following actions in the Tuleap Web UI prior to running any code.

1. **Install Tuleap CE:** Follow the standard installation guide for RHEL/Rocky Linux 9.[1] Ensure the tuleap-plugin-api-explorer is installed to assist in debugging.[1]
2. **Create the Target Project:** Initialize a new project (e.g., project-migration-target).[3]
3. **Tracker Recreation:**
    - Open Codebeamer and list all active Trackers (e.g., Bugs, Requirements, Change Requests).
    - In Tuleap, go to the "Trackers" service and create a new Tracker for each.

  - **Field Replication:** For every field in Codebeamer (e.g., "Priority"), create a corresponding field in the Tuleap Tracker.
    - *String/Text Fields:* Map directly.
    - *Select Boxes:* You must manually populate the list of values in Tuleap (e.g., create "High", "Medium", "Low") to match Codebeamer. This is critical because the API writes values by **ID**, not by label.[7]
  - **Permission Setup:** Ensure the user account running the migration (the API agent) has "Submit" and "Update" permissions on all trackers and fields.[18]

3.2 The Field Mapping Discovery Protocol

Once the empty trackers exist in Tuleap, we must generate a "Rosetta Stone" mapping file. The migration script needs to know that Codebeamer Tracker 1005 maps to Tuleap Tracker 84, and Codebeamer Field Priority maps to Tuleap Field ID 1340.

This discovery is achieved via a helper script (provided in Section 4) that queries Tuleap's GET /trackers/{id} endpoint to download the structure.[7]

3.3 User Identity Management

Tuleap uses internal integer IDs for users. Codebeamer uses its own IDs or usernames.

- **Strategy:** Create a CSV map users.csv mapping codebeamer_username to tuleap_user_id.
- **Provisioning:** If users do not exist in Tuleap, they must be created via LDAP sync or the POST /users (if available/enabled in CE) or site administration tools before data migration.[19] The API for user creation is often restricted or tied to specific site admin privileges.[20]

---

4. Technical Implementation: The Python ETL Pipeline

We will now define the Python code required to execute this migration. The code is modular, designed to be run as a CLI tool.

Prerequisites:

- Python 3.9+
- requests library (pip install requests)
- html2text library (pip install html2text) for converting Codebeamer HTML to Markdown (optional but recommended for cleaner data).

4.1 Configuration and Constants module

This module handles authentication and the crucial IDs derived from the "Structure-First" phase.

Python

```python
# config.py
import os

# --- SOURCE: PTC CODEBEAMER ---
CB_BASE_URL = "https://codebeamer.company.com/cb/api"
CB_USER = os.getenv("CB_USER", "admin")
CB_PASS = os.getenv("CB_PASS", "password")
# Basic Auth is standard for Codebeamer REST API v3
CB_AUTH = (CB_USER, CB_PASS)

# --- TARGET: TULEAP CE ---
TLP_BASE_URL = "https://tuleap.company.org/api/v1"
# Tuleap uses X-Auth-AccessKey header. Generate this in User Profile > Keys.
TLP_API_KEY = os.getenv("TLP_API_KEY", "tlp-k1-xxxxxxxxxxxx")
TLP_HEADERS = {
    "Content-Type": "application/json",
    "X-Auth-AccessKey": TLP_API_KEY,
    "Accept": "application/json"
}

# --- MAPPINGS (The Rosetta Stone) ---
# These IDs must be discovered manually or via the discovery script.
# Structure: { CB_TRACKER_ID: TULEAP_TRACKER_ID }
TRACKER_MAP = {
    2001: 101,  # CB Requirements -> Tuleap Requirements
    2002: 102,  # CB Bugs -> Tuleap Defects
    2003: 103   # CB Tasks -> Tuleap Tasks
}

# Structure: { TULEAP_TRACKER_ID: { 'cb_field_name': TULEAP_FIELD_ID } }
FIELD_MAP = {
    101: { # Tuleap Requirements Tracker
        "name": 1201,          # Title/Summary
        "description": 1202,   # Description
        "status": 1203,        # Select Box
        "priority": 1204       # Select Box
    },
    102: { # Tuleap Defects Tracker
        "name": 1301,
        "description": 1302,
        "status": 1303,
        "severity": 1304
    }
}

# Structure: { TULEAP_FIELD_ID: { "CB_Value_Label": TULEAP_VALUE_ID } }
# Used for Select Boxes. Tuleap requires the ID of the value, not the string.
VALUE_MAP = {
    1203: { # Status Field in Requirements
        "Draft": 501,
        "Accepted": 502,
        "Obsolete": 503
    },
    1304: { # Severity in Defects
        "Critical": 601,
        "Major": 602,
        "Minor": 603
    }
}
```

4.2 Module 1: The Extractor (Codebeamer)

This module handles pagination and fetches item details. Codebeamer's API limits page sizes (typically 50-500), so a while loop is necessary.[13]

Python

```python
# extractor.py
import requests
from config import CB_BASE_URL, CB_AUTH

def get_items_from_tracker(tracker_id):
    """
    Generator that yields all items from a specific Codebeamer tracker.
    Handles pagination automatically.
    """
    page = 1
    page_size = 50 # Conservative size to avoid timeouts

    while True:
        url = f"{CB_BASE_URL}/v3/trackers/{tracker_id}/items"
        params = {"page": page, "pageSize": page_size}

        try:
            print(f" Fetching page {page} for Tracker {tracker_id}...")
            resp = requests.get(url, auth=CB_AUTH, params=params)
            resp.raise_for_status()
            data = resp.json()

            # Codebeamer API v3 structure: { "items": [... ], "total": 100,... }
            items = data.get('items',)

            if not items:
                break

            for item in items:
                # Fetch full details for each item (list view is often partial)
                yield get_full_item_details(item['id'])

            page += 1

        except Exception as e:
            print(f" Failed to fetch from Codebeamer: {e}")
        break

def get_full_item_details(item_id):
    """
    Fetches the complete JSON for a single item, including description and custom fields.
    """
    url = f"{CB_BASE_URL}/v3/items/{item_id}"
    resp = requests.get(url, auth=CB_AUTH)
    return resp.json() if resp.status_code == 200 else None
```

4.3 Module 2: The Transformer

This engine converts Codebeamer's format into Tuleap's expected payload. It handles the mapping of field IDs and values.

Key Insight on Rich Text: Codebeamer descriptions are often complex HTML. Tuleap supports HTML, but converting to Markdown often yields cleaner results for the long term. We will pass the HTML directly for fidelity but strip dangerous tags if necessary.[11]

Key Insight on Select Boxes: Tuleap strictly requires bind_value_ids for select boxes (pulldowns). Sending the string "High" will fail; we must send the Integer ID corresponding to "High".[6]

Python

```
# transformer.py
from config import FIELD_MAP, VALUE_MAP

def transform_to_tuleap_payload(cb_item, tlp_tracker_id):
    """
    Converts a Codebeamer item dict into a Tuleap artifact creation payload.
    """
    cb_fields = cb_item.get('fields',) # CB structure is complex, simplified here
    # Note: CB API v3 usually puts standard fields (name, description) at the root
    # and custom fields in a list.

    tlp_values =
    field_defs = FIELD_MAP.get(tlp_tracker_id, {})

    # 1. Handle Title/Summary
    if 'name' in field_defs:
        tlp_values.append({
            "field_id": field_defs['name'],
            "value": cb_item.get('name', 'Untitled')
        })

    # 2. Handle Description
    if 'description' in field_defs:
        # Codebeamer descriptions can be 'Wiki' or 'Html' format.
        # Tuleap accepts HTML in text fields.
        desc = cb_item.get('description', '')
        desc_format = cb_item.get('descriptionFormat', 'PlainText')

        # Simple passthrough. For advanced use, integrate html2text here.
        tlp_values.append({
            "field_id": field_defs['description'],
            "value": desc
        })

    # 3. Handle Status (Select Box)
    if 'status' in field_defs:
        # CB status might be an object: {"id": 1, "name": "Draft"}
        cb_status_obj = cb_item.get('status')
        if cb_status_obj:
            cb_status_label = cb_status_obj.get('name')
            tlp_field_id = field_defs['status']

            # Look up the ID in our VALUE_MAP
            if tlp_field_id in VALUE_MAP:
                tlp_value_id = VALUE_MAP[tlp_field_id].get(cb_status_label)

                if tlp_value_id:
                    # Tuleap Syntax for Select Boxes: bind_value_ids array
                    tlp_values.append({
                        "field_id": tlp_field_id,
                        "bind_value_ids": [tlp_value_id]
                    })
                else:
                    print(f" No mapping for status '{cb_status_label}'")
    # Construct Final Payload
    payload = {
        "tracker": {
            "id": tlp_tracker_id
        },
        "values": tlp_values
    }

    return payload
```

4.4 Module 3: The Loader (Artifact Creation)

This module posts the data to Tuleap.

Python

```
# loader.py
import requests
from config import TLP_BASE_URL, TLP_HEADERS

def create_artifact(payload):
    """
    POST /artifacts
    Creates the item in Tuleap and returns the new ID.
    """
    url = f"{TLP_BASE_URL}/artifacts"
    try:
        resp = requests.post(url, headers=TLP_HEADERS, json=payload)
        resp.raise_for_status()
        new_data = resp.json()
        print(f" Created Artifact ID {new_data['id']}")
        return new_data['id']
    except requests.exceptions.HTTPError as e:
        print(f" Creation failed: {e.response.text}")
        return None
```

4.5 Module 4: Hierarchy Reconstruction (The Second Pass)

This is a critical step for feasibility. Codebeamer hierarchy is often defined by parent references. Tuleap creates hierarchy via _is_child links.[8] We cannot create the link until *both* parent and child exist in Tuleap.

Algorithm:

1. Store a mapping of CB_ID -> TULEAP_ID in a database or dictionary during the initial load.
2. Iterate through the Codebeamer items again (or the saved map).
3. If a Codebeamer item had a parent, find the Tuleap equivalent of that parent.
4. Update the Tuleap child artifact to add a link to the Tuleap parent.

Python

```python
# linker.py
import requests
from config import TLP_BASE_URL, TLP_HEADERS

def link_artifacts_hierarchy(child_id, parent_id):
    """
    Establishes a Parent/Child relationship in Tuleap.
    This uses the Artifact Links field (Type: _is_child).
    """
    url = f"{TLP_BASE_URL}/artifacts/{child_id}"

    # We need the field ID for "Artifact Links" in the child's tracker.
    # Assume we discovered this is ID 9999 for this example.
    LINK_FIELD_ID = 9999

    # Payload to UPDATE the artifact with a new link
    payload = {
        "values":
            }
        ]
    }
    try:
        # PUT to update
        resp = requests.put(url, headers=TLP_HEADERS, json=payload)
        resp.raise_for_status()
        print(f"[LINK] Linked {child_id} as child of {parent_id}")
    except Exception as e:
        print(f" Linking failed: {e}")
```

4.6 Module 5: Attachment Migration (Advanced)

Handling binary files requires reading from Codebeamer and uploading to Tuleap. Tuleap's file upload process via REST is two-fold:

1. **Upload** the file content to a temporary holding area.
2. **Bind** the temporary file to a specific Artifact field.[21]

Important: For large files, Tuleap supports the TUS protocol. For simplicity, the example below uses the artifact_temporary_files endpoint suitable for smaller documents.

Python

```
# attachment_mgr.py
import requests
from config import CB_BASE_URL, CB_AUTH, TLP_BASE_URL, TLP_HEADERS, TLP_API_KEY

def migrate_attachments(cb_item_id, tlp_artifact_id, tlp_file_field_id):
    # 1. Get list of attachments from Codebeamer
    cb_url = f"{CB_BASE_URL}/v3/items/{cb_item_id}/attachments"
    att_list = requests.get(cb_url, auth=CB_AUTH).json().get('attachments',)

    for att in att_list:
        att_id = att['id']
        filename = att['name']

        # 2. Download Content
        dl_url = f"{CB_BASE_URL}/v3/attachments/{att_id}/content"
        file_content = requests.get(dl_url, auth=CB_AUTH).content
        # 3. Upload to Tuleap (Temporary Area)
        # Note: The header for upload often requires just the AccessKey, not JSON type
        upload_url = f"{TLP_BASE_URL}/artifact_temporary_files"
        files = {'file': (filename, file_content)}
        headers = {'X-Auth-AccessKey': TLP_API_KEY}

        print(f" Uploading {filename}...")
        up_resp = requests.post(upload_url, headers=headers, files=files)

        if up_resp.status_code == 201:
            temp_id = up_resp.json()['id']

            # 4. Bind Temp File to Artifact
            # We assume we are appending. This overwrites if not handled carefully.
            # To append, you often need to read existing file IDs first.
            bind_payload = {
                "values": [{
                    "field_id": tlp_file_field_id,
                    "value": [temp_id] # Array of temp IDs
                }]
            }
            requests.put(f"{TLP_BASE_URL}/artifacts/{tlp_artifact_id}",
                         headers=TLP_HEADERS, json=bind_payload)
            print(f" Attached {filename} to {tlp_artifact_id}")
        else:
            print(f" Upload failed: {up_resp.text}")
```

---

5. Advanced Topics and Deep Insights

5.1 The "Legacy History" Pattern

A major feasibility constraint is the inability to preserve the submitted_on and submitted_by fields authentically in Tuleap CE.10
The Insight: Do not attempt to hack the database to fix dates. This corrupts the data integrity

of Tuleap.
The Solution: Create a "Large Text" field in Tuleap named Legacy Audit Trail. During the Transform phase, iterate through the Codebeamer history endpoint, format the history as a Markdown table (Date | User | Action), and inject this string into the Legacy Audit Trail field. This preserves the information for legal/audit purposes, even if the system metadata shows the migration date.

5.2 Workflow Transitions

In Codebeamer, items have lifecycles. In Tuleap, Trackers have Workflows.
The Problem: If you migrate a Bug with status "Closed", but the Tuleap Workflow requires a transition from "New" -> "Assigned" -> "Resolved" -> "Closed", the API call creating a "Closed" artifact might fail validation.
The Solution:

1. **Disable Workflows** in Tuleap during migration (Admin Console > Trackers > Workflow > Uncheck "Enforce").
2. Perform the migration.
3. Re-enable Workflows. Tuleap generally validates transitions *on change*, not on existing static data.

5.3 Handling Rich Text Dissonance

Codebeamer descriptions often contain inline images referenced by internal URLs (e.g., /cb/display/123).
The Insight: Moving the HTML text without moving the images results in broken images in Tuleap.
The Solution: The Transformer module requires an HTML parser (BeautifulSoup) to detect <img> tags. The script must download the referenced image from Codebeamer, upload it as an attachment to Tuleap, and then rewrite the HTML source in the description to point to the new Tuleap file URL. This is a highly complex sub-project within the migration.

---

6. Validation and Cutover

6.1 Data Integrity Verification

Post-migration, automated validation is required. Do not rely on spot checks.

1. **Count Comparison:** Total active items in CB Tracker X must equal Total active artifacts in Tuleap Tracker Y.
2. **Field Sampling:** Randomly select 100 IDs. Compare the Priority value in source vs target.
3. **Link Integrity:** Traverse the Tuleap hierarchy. Ensure no orphans exist that should have parents.

6.2 The "Big Bang" Decision

For Tuleap CE, a **Big Bang** migration (switching everyone over a weekend) is recommended over a **Piecemeal** (coexistence) strategy.[22]

- **Reasoning:** Synchronization between Codebeamer and Tuleap is incredibly difficult to build bi-directionally. The "Coexist" strategy [23] requires expensive middleware (like OpsHub) which negates the cost benefit of moving to the Community Edition.
- **Recommendation:** Freeze Codebeamer (Read-Only), run the migration scripts, validate, and open Tuleap.

---

7. Conclusion

The migration from PTC Codebeamer to Tuleap Community Edition is a significant engineering undertaking. It is **feasible**, but heavily dependent on the creation of the custom ETL middleware described in this report.

The primary feasibility risks are **Structure Creation** (which must be manual) and **History Fidelity** (which is lost/flattened). However, for organizations seeking to eliminate licensing costs and move to an open architecture, the technical path is clear. The Python implementation provided here serves as the kernel for a production migration tool. By strictly adhering to the "Structure-First" preparation and the "Two-Pass" loading strategy (Data then Links), a successful transition of requirements, defects, and tasks can be achieved.

Summary of Recommendations

1. **Audit:** Map every field and value before writing code.
2. **Develop:** Build the Python ETL using the modular approach (Extractor, Transformer, Loader).
3. **Test:** Perform a dry run on a staging Tuleap instance.
4. **Accept:** Acknowledge the loss of native timestamps in exchange for platform freedom.
5. **Cutover:** Execute a Big Bang migration to minimize synchronization complexity.

Works cited

1. Install packages — Tuleap latest version documentation, accessed on November 28, 2025, https://docs.tuleap.org/installation-guide/step-by-step/install-packages.html
2. Get started with Tuleap The all-in-one solution for Agile Management & DevOps, accessed on November 28, 2025, https://www.tuleap.org/explore
3. Importing Jira projects to Tuleap, accessed on November 28, 2025, https://www.tuleap.org/integration/importing-jira-projects-to-tuleap
4. Tracker Item Operations - PTC Support Portal, accessed on November 28, 2025, https://support.ptc.com/help/codebeamer/r2.1/en/codebeamer/developers_guide/11375769.html
5. Is there a way to create a tracker using Tuleap API REST? - Stack Overflow, accessed on November 28, 2025, https://stackoverflow.com/questions/52260271/is-there-a-way-to-create-a-tracker-using-tuleap-api-rest
6. Various examples — Tuleap latest version documentation, accessed on November 28, 2025, https://docs.tuleap.org/user-guide/integration/rest/api.html
7. 3. Update an artifact — Tuleap latest version documentation, accessed on November 28, 2025, https://docs.tuleap.org/user-guide/integration/rest/quick-start/update.html
8. Artifact Update — Tuleap latest version documentation, accessed on November 28, 2025, https://docs.tuleap.org/user-guide/trackers/usage/artifact-update.html
9. define any type of artifact as children - story #18857 - Tuleap, accessed on November 28, 2025, https://tuleap.net/goto?key=story&val=18857&group_id=101
10. Submit an artifact — Tuleap latest version documentation, accessed on November 28, 2025, https://docs.tuleap.org/user-guide/trackers/usage/submit-artifact.html
11. Codebeamer - Java Rest API client Example - PTC Support Portal, accessed on November 28, 2025, https://support.ptc.com/help/codebeamer/r2.1/en/codebeamer/developers_guide/4164110.html
12. Artifact Fields — Tuleap latest version documentation, accessed on November 28,

2025, https://docs.tuleap.org/user-guide/trackers/usage/artifact-fields.html
13. austinmh12/pybeamer: Python library for interacting with codeBeamer's v3 API - GitHub, accessed on November 28, 2025, https://github.com/austinmh12/pybeamer
14. REST API (v1) - PTC Support Portal, accessed on November 28, 2025, https://support.ptc.com/help/codebeamer/r2.2/en/codebeamer/developers_guide/117612.html
15. Tuleap Enterprise Edition Features, accessed on November 28, 2025, https://www.tuleap.org/product/tuleap-enterprise-edition-features
16. Using REST API to create Project - request #9747 - Tuleap, accessed on November 28, 2025, https://tuleap.net/plugins/tracker/?aid=9747
17. REST with XML - Tuleap documentation, accessed on November 28, 2025, https://docs.tuleap.org/user-guide/integration/rest/xml.html
18. have detailed permissions in REST API - story #13654 - Tuleap, accessed on November 28, 2025, https://tuleap.net/goto?key=story&val=13654&group_id=101
19. Project Import — Tuleap latest version documentation, accessed on November 28, 2025, https://docs.tuleap.org/administration-guide/projects-management/export-import/project-import.html
20. REST API — Tuleap latest version documentation, accessed on November 28, 2025, https://tuleap-documentation-mildred.readthedocs.io/en/latest/rest/auth.html
21. Upload file attachments with the artifact modal - story #8391 - Tuleap, accessed on November 28, 2025, https://tuleap.net/plugins/tracker/?aid=8391
22. Migration | PTC, accessed on November 28, 2025, https://www.ptc.com/en/success-paths/get-started-with-codebeamer/set-up/migration
23. PTC's Codebeamer Migration - IBM DOORS - YouTube, accessed on November 28, 2025, https://www.youtube.com/watch?v=iPTuNA1wVNg