hdl-component-manager Documentation

Release 0.14

Jeremiah C Leary

Aug 11, 2019

Contents:

CHAPTER 1

Overview

HDL Component Manager (HCM) is a tool to manage Hardware Description Language (HDL) IP blocks in an SVN repository. It simplifies the sharing of hdl components between projects without having to perform merges.

With HCM you can:

1. Add new components
2. Switch between versions components
3. Publish updates to existing components
4. Track pedigree of components
5. Manage multiple versions of components

1.1 Why HCM?

HCM was created after attempting to share components between two programs. A merge was attempted from one program to another, and it did not go cleanly. There were multiple instances where I thought a merge was successful, only to find out it was not.

I noticed support for packaging of HDL code lags software implementations. Software has many package managers, e.g. PIP, APT, RPM, and YUM. HCM is an attempt to provide similar capabilities of those package managers for HDL development.

1.2 Key Benefits

- Provides a method to control versions of IP
- Controls the distribution of HDL code
- Can be used to control vendor IP

1.3 Key Features

- Follows the Major.Minor.Patch method of version control
- Works with SVN repositories
- Automates publishing of code to a central location
- Automates installing and upgrading of code
- Supports multiple repositories
- Supports externals
- Language independent (VHDL, Verilog, System Verilog)

CHAPTER 2

Installation

There are two methods to install HCM: PIP and Git Hub.

2.1 PIP

The most recent released version is hosted on PyPI. It can be installed using **pip**.

```
pip install hcm
```

This is the preferred method for installing HCM.

2.2 Git Hub

The latest development version can be cloned from the git hub repo.

```
git clone https://github.com/jeremiah-c-leary/hdl-component-manager.git
```

Then installed using the setup.py file.

```
python setup.py install
```

CHAPTER 3

Usage

HCM can be invoked by issuing **hcm** at the command line prompt:

```
$ hcm
usage: hcm [-h]
           {browse,create,download,install,uninstall,list,publish,show,validate,
↪version}
           ...

Provides configuration management for HDL components.

positional arguments:
  {browse,create,download,install,uninstall,list,publish,show,validate,version}
    browse              List components available for installation.
    create              Creates a component repo
    download            Downloads components without installing them.
    install             Adds a component from the component repo
    uninstall           Removes installed components
    list                Lists components and their versions
    publish             Adds components to the component repo
    show                Displays information about installed components
    validate            Verifies manifest of installed component
    version             Displays HCM version information

optional arguments:
  -h, --help            show this help message and exit
```

HCM has ten subcommands: browse, create, download, install, uninstall, list, publish, show, validate, and version.

3.1 browse

Use the **browse** subcommand to list components available for installation. The arguments for the subcommand can be listed using the *-h* option:

```
$ hcm browse -h
usage: hcm browse [-h] [component]

positional arguments:
  component   Component to browse

optional arguments:
  -h, --help  show this help message and exit
```

3.2 create

Use the **create** subcommand to create a component directory in the repository. The arguments for the subcommand can be listed using the *-h* option:

```
$ hcm create -h
usage: hcm create [-h] url

positional arguments:
  url         location to create the base component repo

optional arguments:
  -h, --help  show this help message and exit
```

3.3 install

Use the **install** subcommand to add or upgrade a component from a repository. The arguments for the subcommand can be listed using the *-h* option:

```
$ bin/hcm install -h
usage: hcm install [-h] [--version VERSION] [--url URL] [--force] [--external]
                   [--dependencies] [--upgrade]
                   component

positional arguments:
  component           Component name to install

optional arguments:
  -h, --help          show this help message and exit
  --version VERSION   Major.Minor.Patch version of component to install.
  --url URL           location of component directory in repo
  --force             Install component ignoring any local changes
  --external          Install as an external
  --dependencies      Install dependencies
  --upgrade           Upgrade dependencies to latest version
```

3.4 uninstall

Use the **uninstall** subcommand to remove installed components. The arguments for the subcommand can be listed using the *-h* option:

```
$ hcm uninstall -h
usage: hcm uninstall [-h] component

positional arguments:
  component   Installed Component name to install

optional arguments:
  -h, --help  show this help message and exit
```

3.5 list

Use the **list** subcommand to check the versions of components you have installed. The arguments for the subcommand can be listed using the *-h* option:

```
$ hcm list -h
usage: hcm list [-h] [--all]

optional arguments:
  -h, --help  show this help message and exit
  --all       Includes directories that are not under HCM control
```

3.6 publish

Use the **publish** subcommand to push a version of a component to a repository. The arguments for the subcommand can be listed using the *-h* option:

```
$ hcm publish -h
usage: hcm publish [-h] -m M [--url URL] component version

positional arguments:
  component   Component name to publish
  version     Major.Minor.Patch version to publish

optional arguments:
  -h, --help  show this help message and exit
  -m M        Commit message
  --url URL   Base URL of the component repository
```

3.7 show

Use the **show** subcommand to display information about an installed component. The arguments for the subcommand can be listed using the *-h* options:

```
$ hcm show -h
usage: hcm show [-h] [--manifest] [--upgrades] [--updates] [--modifications]
                component

positional arguments:
  component        Component to display information
```

(continues on next page)

(continued from previous page)

```
optional arguments:
  -h, --help       show this help message and exit
  --manifest       Displays manifest for all files in component
  --upgrades       Lists upgrade versions and their log entries
  --updates        Lists versions with newer publishes and their log entries
  --modifications  Lists committed modifications for component
```

3.8 validate

Use the **validate** subcommand to compare the component manifest against what is currently installed. The arguments for the subcommand can be listed using the *-h* options:

```
$ hcm validate -h
usage: hcm validate [-h] [--report] component

positional arguments:
  component   Component to display information

optional arguments:
  -h, --help  show this help message and exit
  --report    Reports differences
```

3.9 version

Use the **version** subcommand to display version information for HCM.

3.10 Environment Variables

HCM will use the **HCM_URL_PATHS** environment variable as a replacement for the **–url** command line option. HCM uses the paths in the variable to know which component repositories to interact with.

CHAPTER 4

Browsing Components

Use the **browse** subcommand to view information about components available for installation.

4.1 Example: listing available components

```
$ hcm browse

Component      Version      URL
---------      --------     -------------------------------
bishop         1.1.0        http://svn/my_repo/comps
castle         1.0.0        http://svn/external_repo/blocks
pawn           1.0.0        http://svn/external_repo/blocks
rook           3.0.0        http://svn/my_repo/comps
```

Column	Description
Component	The name of the component installed.
Version	The latest version available for the component.
URL	The URL the component can be installed from.

4.2 Example: listing versions of a component

Adding a component name to the end of the command will list all the versions and their log entries.

```
$ hcm browse rook
rook versions available:

Version: 3.0.0
------------------------------------------------------------------------
r59 | jeremiah | 2019-06-16 08:26:14 -0500 (Sun, 16 Jun 2019) | 2 lines
```

(continues on next page)

(continued from previous page)

```
Updating hcm.json files to correct format.

------------------------------------------------------------------------

Version: 1.6.0
------------------------------------------------------------------------
r59 | jeremiah | 2019-06-16 08:26:14 -0500 (Sun, 16 Jun 2019) | 2 lines

Updating hcm.json files to correct format.

------------------------------------------------------------------------

Version: 1.5.0
------------------------------------------------------------------------
r59 | jeremiah | 2019-06-16 08:26:14 -0500 (Sun, 16 Jun 2019) | 2 lines

Updating hcm.json files to correct format.

------------------------------------------------------------------------

Version: 1.4.0
------------------------------------------------------------------------
r59 | jeremiah | 2019-06-16 08:26:14 -0500 (Sun, 16 Jun 2019) | 2 lines

Updating hcm.json files to correct format.

------------------------------------------------------------------------

Version: 1.3.0
------------------------------------------------------------------------
r59 | jeremiah | 2019-06-16 08:26:14 -0500 (Sun, 16 Jun 2019) | 2 lines

Updating hcm.json files to correct format.

------------------------------------------------------------------------

Version: 1.2.0
------------------------------------------------------------------------
r59 | jeremiah | 2019-06-16 08:26:14 -0500 (Sun, 16 Jun 2019) | 2 lines

Updating hcm.json files to correct format.

------------------------------------------------------------------------

Version: 1.1.0
------------------------------------------------------------------------
r59 | jeremiah | 2019-06-16 08:26:14 -0500 (Sun, 16 Jun 2019) | 2 lines

Updating hcm.json files to correct format.

------------------------------------------------------------------------

Version: 1.0.0
------------------------------------------------------------------------
r10 | jeremiah | 2019-05-20 21:39:51 -0500 (Mon, 20 May 2019) | 1 line
```

(continues on next page)

(continued from previous page)

```
initial release
------------------------------------------------------------------------
```

CHAPTER 5

Create Component Directory

HCM will create the component directory and place it in an existing SVN repository. This top level directory can be anywhere. However, it is commonly placed either at the root or under the tags directory.

5.1 Creating a component directory

Use the **create** subcommand to create the component directory in a repository:

```
$ hcm create http://svn/my_repo/components
INFO:Creating component directory http://svn/my_repo/components
INFO:Add "http://svn/my_repo/components" to the HCM_URL_PATHS environment variable.
```

HCM will create any level of hierarchy necessary to create the given URL path.

> **Warning:** If the URL already exists, an error will be reported

Adding the URL to the **HCM_URL_PATHS** environment variable will give HCM visibility to those repositories.

```
$ export HCM_URL_PATHS=http://svn/acme/components,$HCM_URL_PATHS
```

Note: The separator is a comma and not a colon.

CHAPTER 6

Downloading Components

Use the **download** subcommand to pull down a specific version of a component without installing it. This can be useful when trying to merge two versions of a component.

6.1 Example: Downloading a Component

After viewing the component repository, we decide to download version 1.1.0 of the component rook.

```
$ hcm download rook 1.1.0
INFO:Downloading component rook version 1.1.0
INFO:Download complete
```

HCM will use the paths in the **HCM_URL_PATHS** environment variable. It will search each path for a matching component name and version.

CHAPTER 7

Installing Components

Use the **install** subcommand to add a component to your current working copy. The URL path to the component and the version are required to install a new component. This can be found using a repository browser or the **svn ls** subcommand.

7.1 Example: Installing a Component

After viewing the component repository, we decide to pull in version 3.0.0 of the rook component.

```
$ hcm install rook --version 3.0.0
INFO:Installing component rook version 3.0.0
INFO:Validating all files for component rook are committed.
INFO:Removing local component directory
INFO:Installation complete
```

HCM will use the paths in the **HCM_URL_PATHS** environment variable. It will search each path for a matching component name and version.

7.2 Example: Installing the latest version of a component

If the version argument is not use, then HCM will install the latest version of the component.

```
$ hcm install rook
INFO:Installing component rook
INFO:Validating all files for component rook are committed.
INFO:Removing local component directory
INFO:Installation complete
```

7.3 Example: installing component when files under the component directory are not committed

HCM validates every file under the local component directory is checked in. If this is not the case, then HCM will not install over the existing directory.

```
$ ../../bin/hcm install rook --version 3.0.0
INFO:Installing component rook version 3.0.0
INFO:Validating all files for component rook are committed.
ERROR:The following files must be committed or removed:
M       rook/rtl/rook.vhd
```

This behavior can be overridden by using the **–force** command line option.

```
$ hcm install rook --version 3.0.0 --force
INFO:Installing component rook version 3.0.0
INFO:Removing local component directory
INFO:Installation complete
```

7.4 Example: installing from an external repo

When installing from an external repo, HCM must use the **svn export** command.

```
$ hcm install pawn --version 1.0.0 --url http://svn/external_repo/blocks
INFO:Installing component pawn version 1.0.0
INFO:Validating all files for component pawn are committed.
INFO:Removing local component directory
INFO:Installation complete
```

Performing an **svn status** command shows a new directory has been created.

```
$ svn status
?       pawn
```

The directory must be added using the **svn add** command. . .

```
$ svn add pawn
A         pawn
A         pawn/hcm.json
A         pawn/rtl
A         pawn/rtl/pawn.vhd
```

. . . and then committed.

```
$ svn commit pawn
```

Note: The last two steps are left to the user to perform.

7.5 Example: Installing using an external

HCM can install components using externals. An external is a essentially a pointer to directory in a repository.

```
$ hcm install pawn --version 3.0.0 --external
INFO:Installing component pawn version 3.0.0
INFO:Validating all files for component pawn are committed.
INFO:Removing local component directory
INFO:Updating externals
INFO:Installation complete
```

Checking the svn status of the current directory...

```
$ svn status
 M       .
X        castle
X        pawn
```

...shows the properties of the existing directory have been modified and pawn is an external. The directory must be committed to keep the change to 3.0.0 of pawn.

7.6 Example: Installing Component and It's Dependents

HCM can keep track of dependencies between components using dependency file. This file is generated by the user and committed with the component before it is published. If HCM detects this file, it install any of the components listed.

```
$ hcm install rook --dependencies
INFO:Installing component rook
INFO:Removing local component directory
INFO:Installing dependencies
INFO:Checking for dependencies of rook
INFO:Installing component king
INFO:Removing local component directory
INFO:Checking for dependencies of king
INFO:Installing component castle
INFO:Removing local component directory
INFO:Checking for dependencies of castle
INFO:  No Dependencies found
INFO:Installing component pawn
INFO:Removing local component directory
INFO:Checking for dependencies of pawn
INFO:  No Dependencies found
INFO:Installing component queen
INFO:Removing local component directory
INFO:Checking for dependencies of queen
INFO:Installation complete
```

In this example the **dependencies.yaml** file for rook contained the following:

```
---
requires:
  queen:
  king:
```

and the queen component contained a **dependencies.yaml** file:

```
---
requires:
  rook:
  king:
  pawn:
```

and the king component contained a **dependencies.yaml** file:

```
---
requires:
  pawn:
  castle:
```

HCM will break an circular dependencies and only install a component once.

7.7 Example: Installing Component and Install the Latest Version of Dependents

If a component has already been installed, HCM will not install over it. This behavior can be modified by using the *–upgrade* command line argument. HCM will install the latest version of every dependent component if this argument is used.

```
$ hcm install rook --dependencies --upgrade
INFO:Installing component rook
INFO:Removing local component directory
INFO:Installing dependencies
INFO:Checking for dependencies of rook
INFO:Installing component king
INFO:Removing local component directory
INFO:Checking for dependencies of king
INFO:Installing component castle
INFO:Removing local component directory
INFO:Checking for dependencies of castle
INFO:  No Dependencies found
INFO:Installing component pawn
INFO:Removing local component directory
INFO:Checking for dependencies of pawn
INFO:  No Dependencies found
INFO:Installing component queen
INFO:Removing local component directory
INFO:Checking for dependencies of queen
INFO:Installation complete
```

CHAPTER 8

Uninstalling Components

Use the **uninstall** subcommand to remove components from your current working copy.

8.1 Example: Uninstalling a Component

We have decided we no longer need the component bishop.

```
$ hcm uninstall bishop
INFO:Uninstalled component bishop
```

The component must be committed for the change to be permanent:

```
$ svn commit -m "Removed bishop component"
```

8.2 Example: Uninstalling an externalled component

HCM will modify the svn:externals attribute and perform an update.

```
$ hcm uninstall rook
INFO:Uninstalled component rook
```

The parent directory must be committed for the change to be permanent:

```
$ svn commit . -m "Removed bishop component"
```

CHAPTER 9

Listing Components

Use the **list** subcommand to view information about installed components.

9.1 Example: listing installed components

```
$ hcm list

Component      Version      Upgrade      Status     URL
---------      --------     --------     ------     -------------------------------
bishop         1.1.0        None           U        http://svn/my_repo/comps
castle         1.0.0        None         E U        http://svn/external_repo/blocks
pawn           1.0.0        3.1.0        E U        http://svn/external_repo/blocks
rook           3.0.0        3.0.3         MU        http://svn/my_repo/comps
new_comp       -----        -----            N      -----
```

The upgrade column shows the latest published version available. There may be several versions between what is installed and what is published. Use a repository browser to decide whether to upgrade a component.

Column	Description
Component	The name of the component installed.
Version	The version of the installed component.
Upgrade	The latest published version of the component.
Status	Flags indicating information about the component. E = Component was installed as an external. M = Component has commited modifications. U = Component has uncommitted modifications. N = Directory is not under SVN control.
URL	The base URL the component was installed from.

CHAPTER 10

Publishing Components

Use the **publish** subcommand to add or update components in a repository.

There are a couple of requirements before a component can be published.

1. component directory must be checked into SVN
2. component directory must be status clean
3. **HCM_URL_PATHS** should be defined

HCM uses **svn copy** commands to publish components. This ensures a history is maintained for the component development.

10.1 Example: Publishing a new component

A new component can be published, but HCM must be told where to publish the component. This can be done by setting the **HCM_URL_PATHS** environment variable or using the *–url* command line argument. If only one path is defined in **HCM_URL_PATHS**, then HCM will use it as the publish location. Using the *–url* command line argument will override **HCM_URL_PATHS**.

Note: Publishing is restricted to the current repository.

```
$ hcm publish bishop 1.0.0 --url http://svn/acme/chess/components -m "Initial release␣
↪of bishop."

INFO:Publishing component bishop as version 1.0.0
INFO:Validating all files for component bishop are committed.
INFO:Validating component exists in component directory...
INFO:Creating component in component directory.
INFO:Searching for hcm.json file...
WARNING:Did not find hcm.json for component bishop.
INFO:Creating default hcm.json file...
```

(continues on next page)

(continued from previous page)

```
INFO:Updating version...
INFO:Updating source URL...
INFO:Creating manifest...
INFO:Writing configuration file bishop/hcm.json
INFO:Adding configuration file to component directory
INFO:Committing bishop/hcm.json file
INFO:Component published
```

HCM will create a configuration file named **hcm.json** and commit it to the working copy. This file contains information related to the component.

10.2 Example: Publishing an update to a component

If a component has been updated, the updates can be published. Since the **hcm.json** file exists, the *–url* argument is not required. HCM will use the information in the **hcm.json** file to determine where the component will be published.

```
$ hcm publish bishop 1.1.0 -m "Fixing movement bug."

INFO:Publishing component bishop as version 1.1.0
INFO:Validating all files for component bishop are committed.
INFO:Searching for hcm.json file...
INFO:Validating component exists in component directory...
INFO:Updating version...
INFO:Updating source URL...
INFO:Creating manifest...
INFO:Writing configuration file bishop/hcm.json
INFO:Adding configuration file to component directory
INFO:Committing bishop/hcm.json file
INFO:Component published
```

HCM will update the **hcm.json** file with the new version and commit it to the working copy before it is committed to the component directory.

10.3 Example: Using a file for the commit message

Publishing supports using a file for the commit message. This is used instead of the *-m* command line option

```
$ hcm publish bishop 1.1.0 -F release_notes.txt

INFO:Publishing component bishop as version 1.1.0
INFO:Validating all files for component bishop are committed.
INFO:Searching for hcm.json file...
INFO:Validating component exists in component directory...
INFO:Updating version...
INFO:Updating source URL...
INFO:Creating manifest...
INFO:Writing configuration file bishop/hcm.json
INFO:Adding configuration file to component directory
INFO:Committing bishop/hcm.json file
INFO:Component published
```

CHAPTER 11

Showing Components

Use the **show** subcommand to view detailed information about a component. This includes the component name, it's version, URL. Optionally, every file that makes up the component can be listed along with it's md5sum.

11.1 Example: Viewing information about rook

```
$ hcm show bishop
----------      ------------------------------------------------------------------------
 ↪---
Component       bishop
Version         1.1.0
URL             http://svn/my_repo/comps
Source          http://svn/my_repo/trunk/project_chess/components/bishop@26
----------      ------------------------------------------------------------------------
 ↪---
```

11.2 Example: Viewing manifest

```
$ hcm show rook --manifest
----------      ------------------------------------------------------------------------
 ↪-
Component       rook
Version         3.0.0
URL             http://svn/my_repo/comps
Source          http://svn/my_repo/trunk/project_chess/components/rook@15
----------      ------------------------------------------------------------------------
 ↪-

Manifest
--------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
10019aef04979acfac88673bc5dc6133    rook/lay/filelist.tcl
a461fa565f1f7822bcdbda0b450df476    rook/rtl/rook.vhd
```

Note: The manifest is extracted from the hcm.json file. It does not include any local changes to the files. Use the **validate** subcommand to compare the manifest against what is installed.

11.3 Example: Viewing available upgrades

All available upgrades and their log entries can be listed.

```
$ hcm show rook --upgrades
------------     ------------------------------------------------------------------------
↪---
Component        rook
Version          3.0.0
URL              http://svn/my_repo/comps
Source           http://svn/my_repo/trunk/project_chess/components/rook@41
Dependencies     king, queen
------------     ------------------------------------------------------------------------
↪---

Available Upgrades
==================

Version: 4.0.0
------------------------------------------------------------------------
r42 | jeremiah | 2019-06-11 19:09:53 -0500 (Tue, 11 Jun 2019) | 1 line

 "testing dependencies"
------------------------------------------------------------------------

Version: 3.0.5
------------------------------------------------------------------------
r51 | jeremiah | 2019-06-13 19:37:16 -0500 (Thu, 13 Jun 2019) | 1 line

 "Updating hcm config to the latest version."
------------------------------------------------------------------------

Version: 3.0.4
------------------------------------------------------------------------
r49 | jeremiah | 2019-06-12 20:02:38 -0500 (Wed, 12 Jun 2019) | 1 line

 "Adding invalid component to test how HCM handles it."
------------------------------------------------------------------------

Version: 3.0.3
------------------------------------------------------------------------
r35 | jeremiah | 2019-05-30 22:00:03 -0500 (Thu, 30 May 2019) | 1 line

 "testing -m works"
------------------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
Version: 3.0.2
------------------------------------------------------------------------
r34 | jeremiah | 2019-05-30 21:58:38 -0500 (Thu, 30 May 2019) | 6 lines

This is a test of using the -F argument when publishing.

It should allow the use of a file instead of a single line for the commit message.



------------------------------------------------------------------------

Version: 3.0.1
------------------------------------------------------------------------
r33 | jeremiah | 2019-05-30 21:57:37 -0500 (Thu, 30 May 2019) | 6 lines

This is a test of using the -F argument when publishing.

It should allow the use of a file instead of a single line for the commit message.



------------------------------------------------------------------------
```

11.4 Example: Viewing available updates

Updates are slightly different than upgrades. Updates include all versions that were committed after the currently installed version.

```
$ hcm show rook --updates
------------     ------------------------------------------------------------------------
 ↪---
Component        rook
Version          3.0.0
URL              http://svn/my_repo/comps
Source           http://svn/my_repo/trunk/project_chess/components/rook@41
Dependencies     king, queen
------------     ------------------------------------------------------------------------
 ↪---

Available Upgrades
==================

Version: 3.0.5
------------------------------------------------------------------------
r51 | jeremiah | 2019-06-13 19:37:16 -0500 (Thu, 13 Jun 2019) | 1 line

 "Updating hcm config to the latest version."
------------------------------------------------------------------------

Version: 3.0.4
------------------------------------------------------------------------
r49 | jeremiah | 2019-06-12 20:02:38 -0500 (Wed, 12 Jun 2019) | 1 line
```

(continues on next page)

(continued from previous page)

```
 "Adding invalid component to test how HCM handles it."
------------------------------------------------------------------------

Version: 4.0.0
------------------------------------------------------------------------
r42 | jeremiah | 2019-06-11 19:09:53 -0500 (Tue, 11 Jun 2019) | 1 line

 "testing dependencies"
------------------------------------------------------------------------

Version: 3.0.3
------------------------------------------------------------------------
r35 | jeremiah | 2019-05-30 22:00:03 -0500 (Thu, 30 May 2019) | 1 line

 "testing -m works"
------------------------------------------------------------------------

Version: 3.0.2
------------------------------------------------------------------------
r34 | jeremiah | 2019-05-30 21:58:38 -0500 (Thu, 30 May 2019) | 6 lines

This is a test of using the -F argument when publishing.

It should allow the use of a file instead of a single line for the commit message.



------------------------------------------------------------------------

Version: 3.0.1
------------------------------------------------------------------------
r33 | jeremiah | 2019-05-30 21:57:37 -0500 (Thu, 30 May 2019) | 6 lines

This is a test of using the -F argument when publishing.

It should allow the use of a file instead of a single line for the commit message.



------------------------------------------------------------------------
```

11.5 Example: Viewing modifications

Modifications made to a component after installation can be viewed. The **–modifications** argument will display the log entries for every change since the last install. Both committed and uncommitted modifications will be shown.

```
$ hcm show rook --modifications
------------     ------------------------------------------------------------------------
↪---
Component        rook
Version          4.0.0
URL              http://svn/my_repo/comps
Source           http://svn/my_repo/trunk/project_chess/components/rook@41
Dependencies     king, queen
```

(continues on next page)

(continued from previous page)

```
------------     ------------------------------------------------------------------------
 ↪---

Uncommitted Modifications
=========================
A  +    rook
?       rook/rtl/movement.vhd
M  +    rook/rtl/rook-rtl.vhd

Committed Modifications
=======================
------------------------------------------------------------------------
r63 | jeremiah | 2019-06-21 06:13:40 -0500 (Fri, 21 Jun 2019) | 2 lines

Minor change to rook entity.

------------------------------------------------------------------------
r62 | jeremiah | 2019-06-21 06:05:24 -0500 (Fri, 21 Jun 2019) | 2 lines

Adding architecture.

------------------------------------------------------------------------
```

HCM will also indicate if no modifications were detected.

```
$ hcm show rook --modifications
------------     ------------------------------------------------------------------------
 ↪---
Component        rook
Version          4.0.0
URL              http://svn/my_repo/comps
Source           http://svn/my_repo/trunk/project_chess/components/rook@41
Dependencies     king, queen
------------     ------------------------------------------------------------------------
 ↪---

Uncommitted Modifications
=========================
No Uncommitted Modifications

Committed Modifications
=======================
No Committed Modifications
```

CHAPTER 12

Theory of Operation

This section details how HCM performs it's tasks.

12.1 Component Directory Structure

HCM can create the component directory and place it in your repository. It is commonly placed either at the root of the repository or under the tags directory.

The component directory contains individual directories for each component. Under each individual component name are the releases for that component. Each release directory follows the form of a three dot number: <Major>.<Minor>.<Patch>

The example below shows a component directory with three components: rook, king, and queen.

The **rook** component has three releases: 1.0.0, 1.1.0, and 2.0.0.

The **king** component has four releases: 1.0.0, 2.0.0, 2.1.0, and 3.0.0.

The **queen** component has four releases: 1.0.0, 2.0.0, 2.1.0, and 3.0.0.

```
components
|
+-- rook
|   |
|   +-- 1.0.0
|   +-- 1.1.0
|   +-- 2.0.0
|
+-- king
|   |
|   +-- 1.0.0
|   +-- 2.0.0
|   +-- 2.1.0
|   +-- 3.0.0
|
```

(continues on next page)

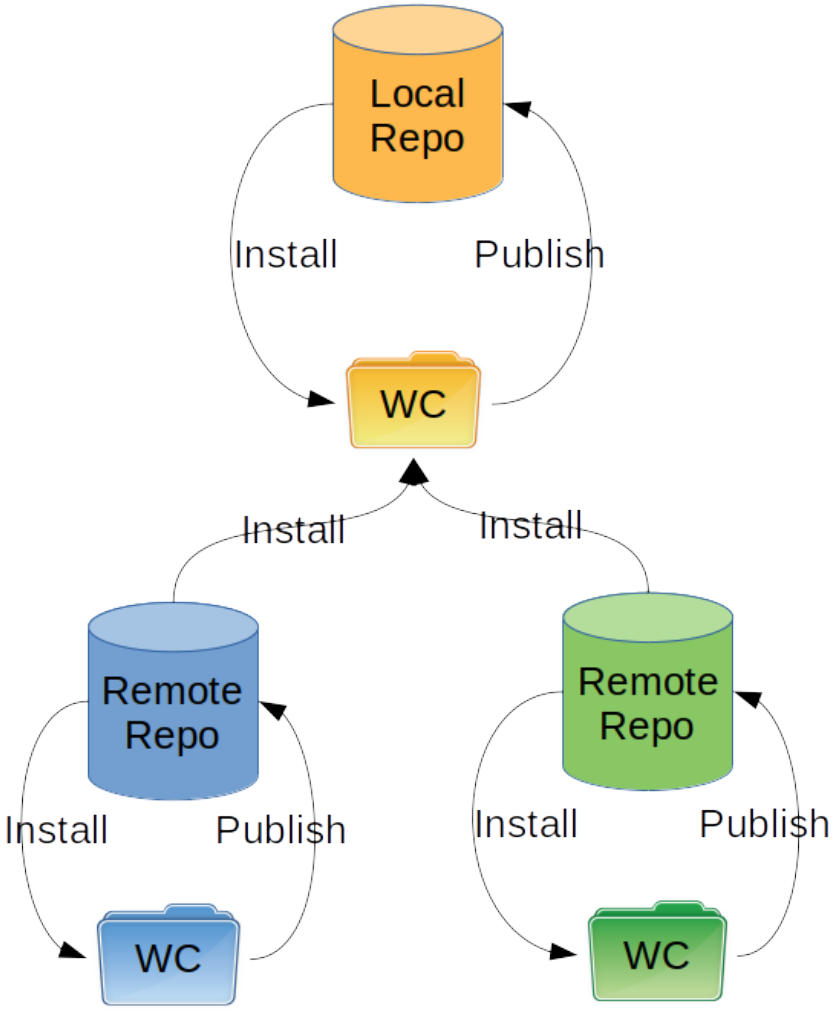
(continued from previous page)

```
+-- queen
    |
    +-- 1.0.0
    +-- 2.0.0
    +-- 2.1.0
    +-- 3.0.0
```

Differences between releases can be easily determined by comparing the two release URLs.

12.2 Repository Considerations

HCM makes several assumptions about the workflow with regards to repositories. The diagram below shows the workflow assuming there are three repositories.



The workflow makes these assumptions:

1. Publishing is restricted to within the local repo.

2. Installing can be performed from either a local or remote repo.

12.2.1 Ownership Considerations

Components should only be developed and published in a single repository. This ensures a single source of truth for the component.

In the diagram above:

- Code in the yellow repository is developed using the yellow working copy.
- Code in the blue repository is developed using the blue working copy.
- Code in the green repository is developed using the green working copy.

You can not develop blue components in the yellow working copy. Revision control tools do not handle crossing repositories well.

12.3 HCM configuration file

The HCM configuration file is a JSON file which contains information about the component. There is an HCM configuration file for each component and is updated with every version released.

```
{
  "name" : "rook",
  "version" : "1.0.0",
  "publish" : {
      "url" : "http://svn/my_repo/components"
  },
  "source" : {
      "url" : "http://svn/my_repo/chess_project/components/rook@1276",
      "manifest" : {
          "rook/rtl/rook.vhd" : "93ffadcc3b73c6292de35564192a99b4",
          "rook/lay/filelist.tcl" : "10019aef04979acfac88673bc5dc6133"
      }
  }
}
```

The JSON file starts with a single hash key named **hcm**. This uniquely identifies the information as belonging to HCM. It contains five other keys: url, name, version, source_url, and manifest.

Key	Description
name	name of the component.
version	version of the component that has been published.
publish:url	location of the component directory where this component has been published.
source:url	current URL path and revision where the component was published from.
source:manifest	key value pair of every file that makes up the component. The key is the name of the file relative. The value is an md5sum hash of that file.

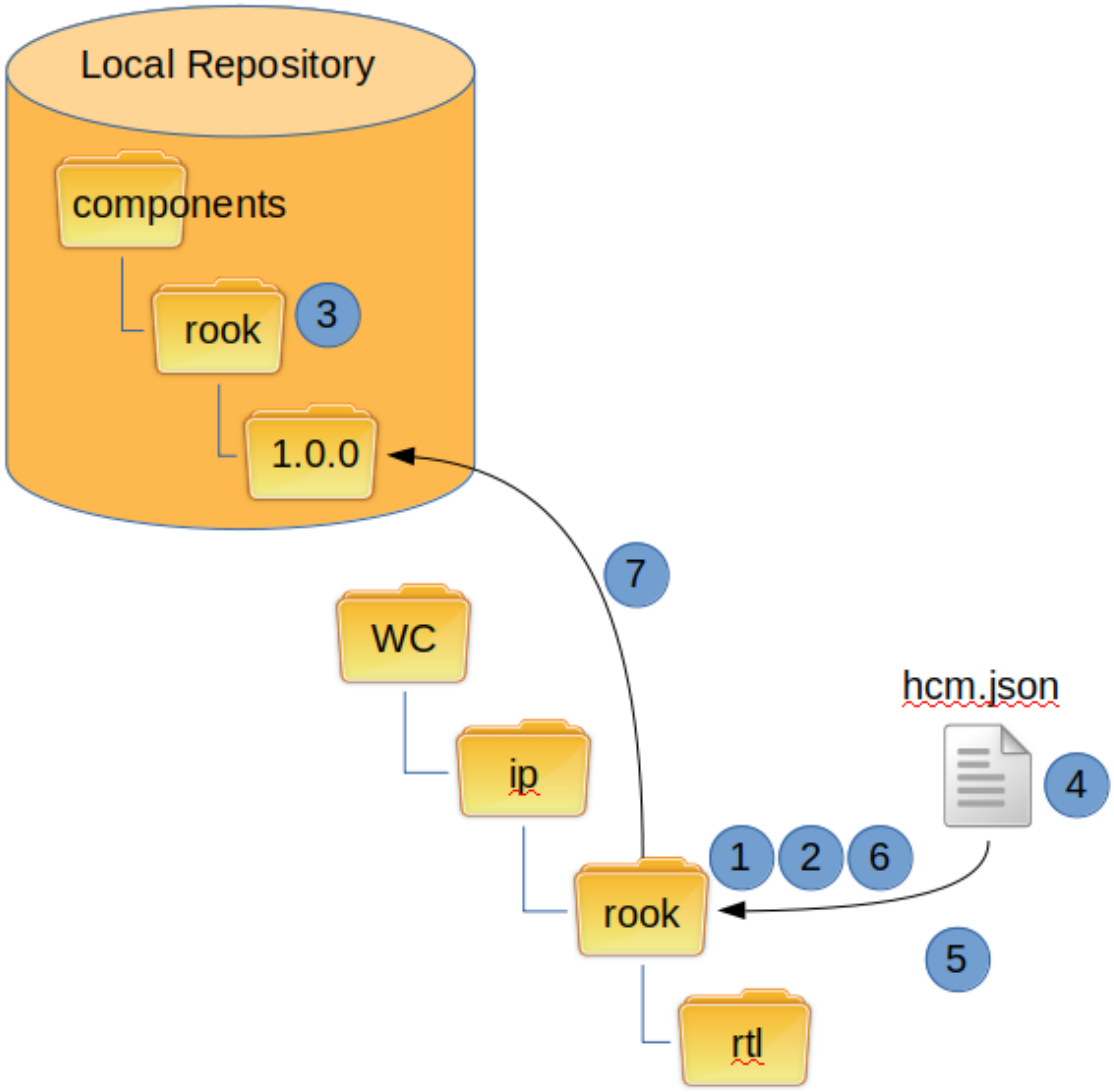
The manifest provides a quick method to validate any component to see if anything has changed. It can also assist in transferring components between repos.

12.4 Publishing

Publishing uses the svn copy command to take snapshots of a component. The command can only work within a repository.

In the diagram above, you can see all the publish actions take place between a repository and it's respective working copy.

To publish manually you would follow these steps:



1. Ensure requested component directory exists
2. Validate all files in a component to be published directory are committed.
    a. Any unversioned files must be deleted
    b. it must come back with a clean status

3. Create individual component directory in component directory if it does not exist.
4. Generate the hcm.json file if it does not exist
    a. Read hcm.json file it is does exist
    b. Update version and manifest fields
5. Add hcm.json file to the component directory
6. Commit hcm.json file to working copy
7. svn copy the local component directory to the published directory under the correct version

HCM will validate step 2 has been completed before performing steps 3 through 6.

12.5 Installing

Installing can take place from a local repo or a remote repo. If an install is performed from a local repo, then the svn copy command will be used. This provides history between what is installed and what has been published.

If an install is performed from a remote repo, then the svn export command will be used. This pulls the component from the other repo and copies it into the working copy. The files need to be added and then committed. History is not lost in this case, but it is a little more difficult to follow.

12.5.1 Local Install

This is the workflow for a local install:

1. Check if component version exists in repo
2. Check if local component directory is svn status clean
3. Delete component directory in working copy
4. SVN copy component version directory from local repo

HCM will perform all these steps.

Note: Committing the install is the responsibility of the user.

12.5.2 Remote Install

This is the workflow for a remote install:

1. Check if component version exists in repo
2. Check if local component directory is svn status clean
3. Delete component directory in working copy
4. Export component from external repo
5. Add exported component to working copy

HCM will perform all these steps.

Note: Committing the install is the responsibility of the user.

12.5.3 External Install

A component can be installed using an external instead of an svn copy or export. When using an external, the type of repo (external or local) does not matter.

This is the workflow for an external install:

1. Read externals from svn:externals property on current directory
2. Check the status of the local component directory
3. Check if component version exists in repo
4. Update svn:externals property with new component version
5. Update component directory

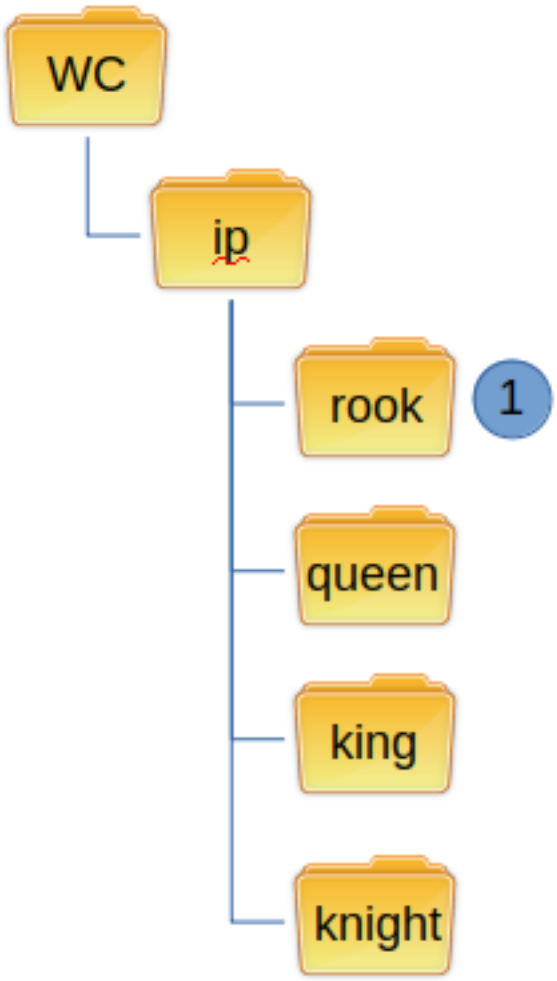
12.6 Uninstalling

Uninstalling removes an installed component. If the component was not an external install, then HCM will just use svn delete to remove the component. If the componet was an external install, then HCM will modify the svn:externals property to remove the component.

In either case, the user is responsible for the final commit to make the uninstall official.

12.6.1 Normal Uninstall

This is the workflow for a normal uninstall:



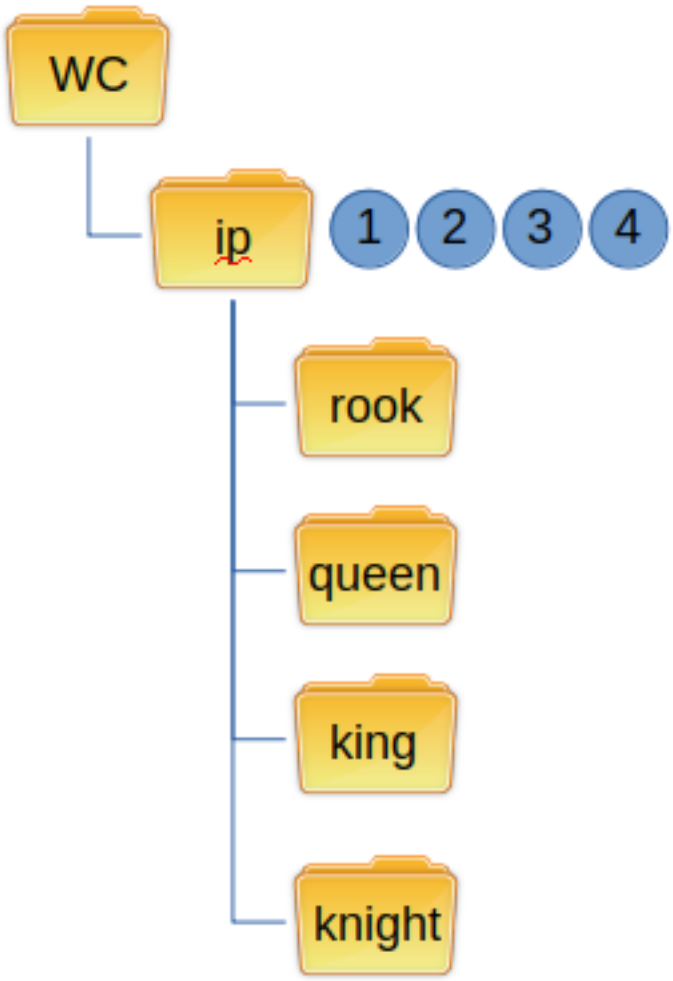
1. SVN delete component

Note: Committing the uninstall is the responsibility of the user.

12.6.2 External Uninstall

Uninstalling an external requires modifying the svn:externals property of the parent directory.

This is the workflow for an external uninstall:



1. Read externals from svn:externals property on current directory
2. Remove component from svn:externals property
3. Update svn:externals property with new component version
4. Update component directory

Note: Committing the uninstall is the responsibility of the user.

12.7 Dependencies

Dependencies between components can be indicated using a file named *dependencies.yaml*. This file is generated by hand by the user. This is an example dependency file for the **rook** component:

```
requires:
  queen:
```

(continues on next page)

(continued from previous page)

```
  king:
  castle:
```

When **rook** is installed, HCM will also install **queen**, **king**, and **castle** if they are not already installed.

12.7.1 Install with dependencies

This is the workflow for a local install:

1. Check if component version exists in repo
2. Check if local component directory is svn status clean
3. Delete component directory in working copy
4. SVN copy component version directory from local repo
5. Read dependencies.yaml file
6. Check if components in YAML file have been installed
7. Read dependencies.yaml file for those components that have been installed
8. Install components that have not been installed

This process repeats until all dependencies have been installed. Any duplicate dependencies are ignored. All circular dependencies are broken.

Note: HCM will install the latest version of a component when it is listed as a dependency.

12.8 Dependencies file

The HCM Dependencies file is a YAML file listing other components required.

```
---
requires:
   component1:
   component2:

   componentn:
```

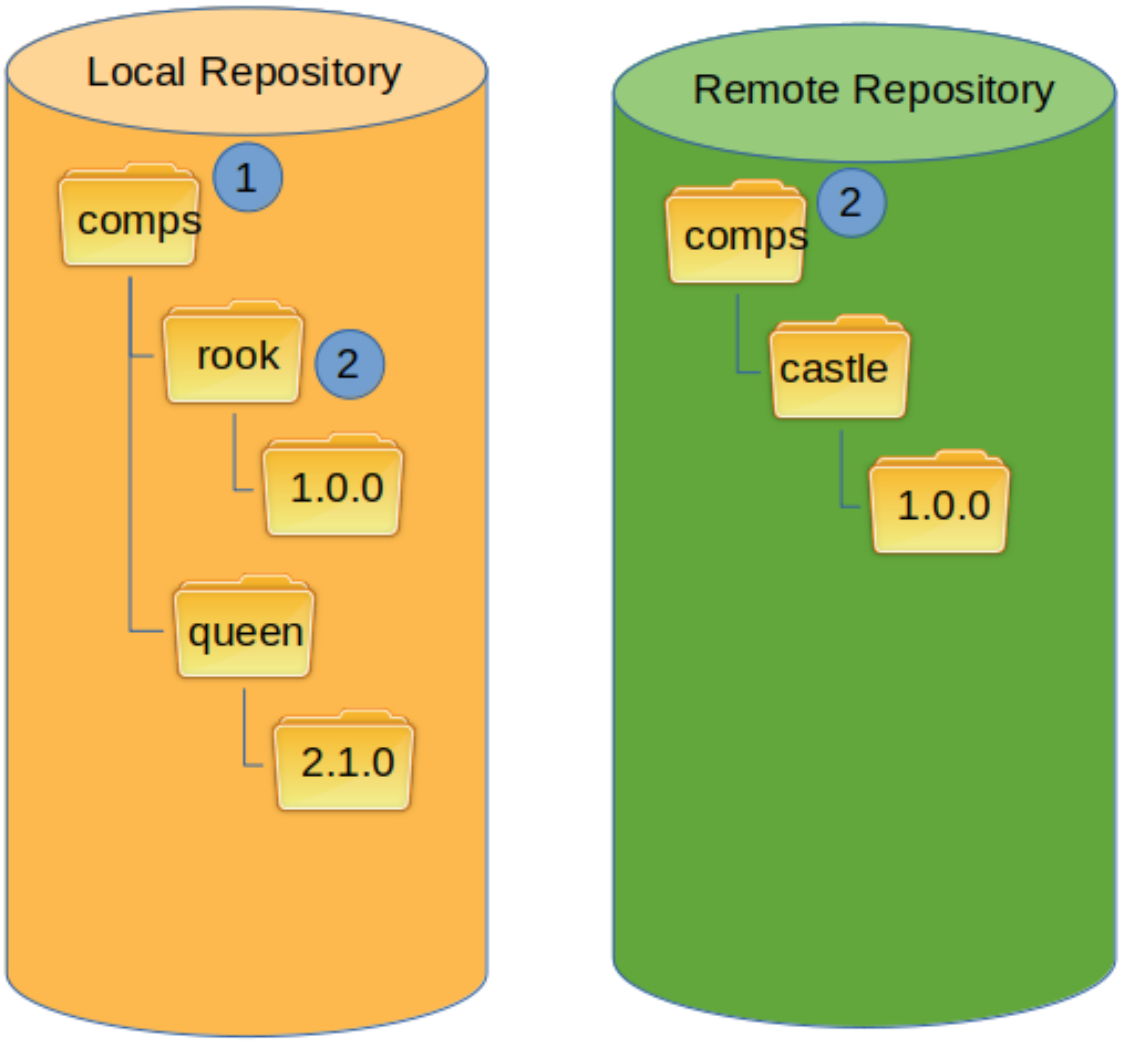
Each *component* entry is a key that currently does not have a value. This format was chosen to allow for easy extension of the dependency feature.

This file must be named **dependencies.yaml** and placed at the root of the component. HCM will search for this file when installing to see if any other components must be installed.

12.9 Browsing

Browsing checks the versions of the components available for installation.

The workflow for listing is shown below:



1. Get a listing of all directories in each path in the HCM_URL_PATHS environment variable
2. Get the latest version of each component

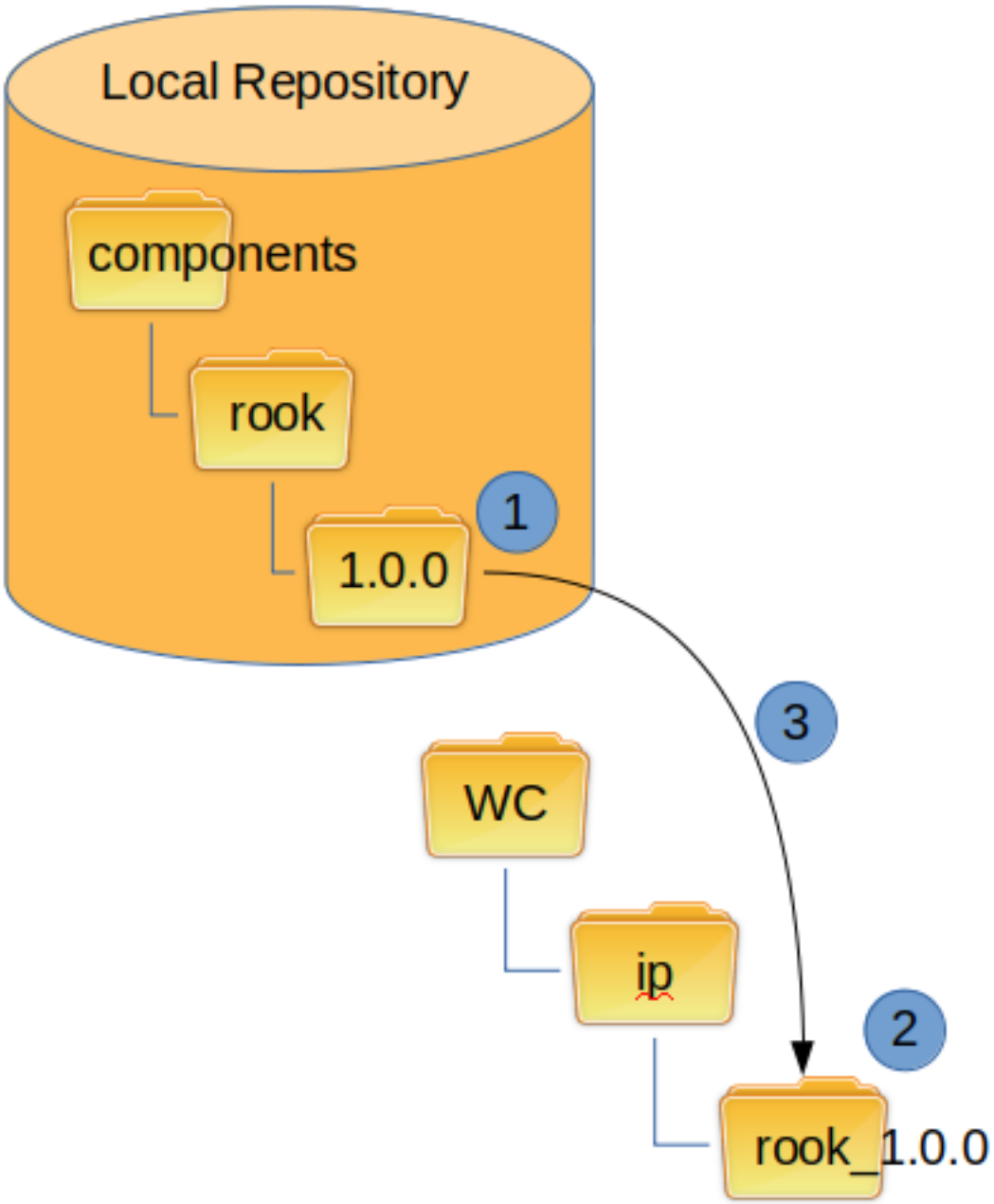
The information is then displayed for the user.

12.10 Downloading

Downloading can take place from a local repo or a remote repo.

If an install is performed from a remote repo, then the svn export command will be used. This pulls the component from the other repo and copies it into the working copy. The files need to be added and then committed. History is not lost in this case, but it is a little more difficult to follow.

This is the workflow for downloading:



1. Check if component version exists in repo
2. Removing existing download directory if it exists
3. Export component to using component name and version number

12.11 Listing

Listing checks the versions of the components currently installed.

The workflow for listing is shown below:

1. Get a listing of all directories
2. Read version from each hcm.json file

The information is then displayed for the user.

12.12 Detecting Modifications

Modifications to installed components is important in making several decisions:

1) When to publish
2) Whether to install

This is the workflow for checking is an installed component has modifications:

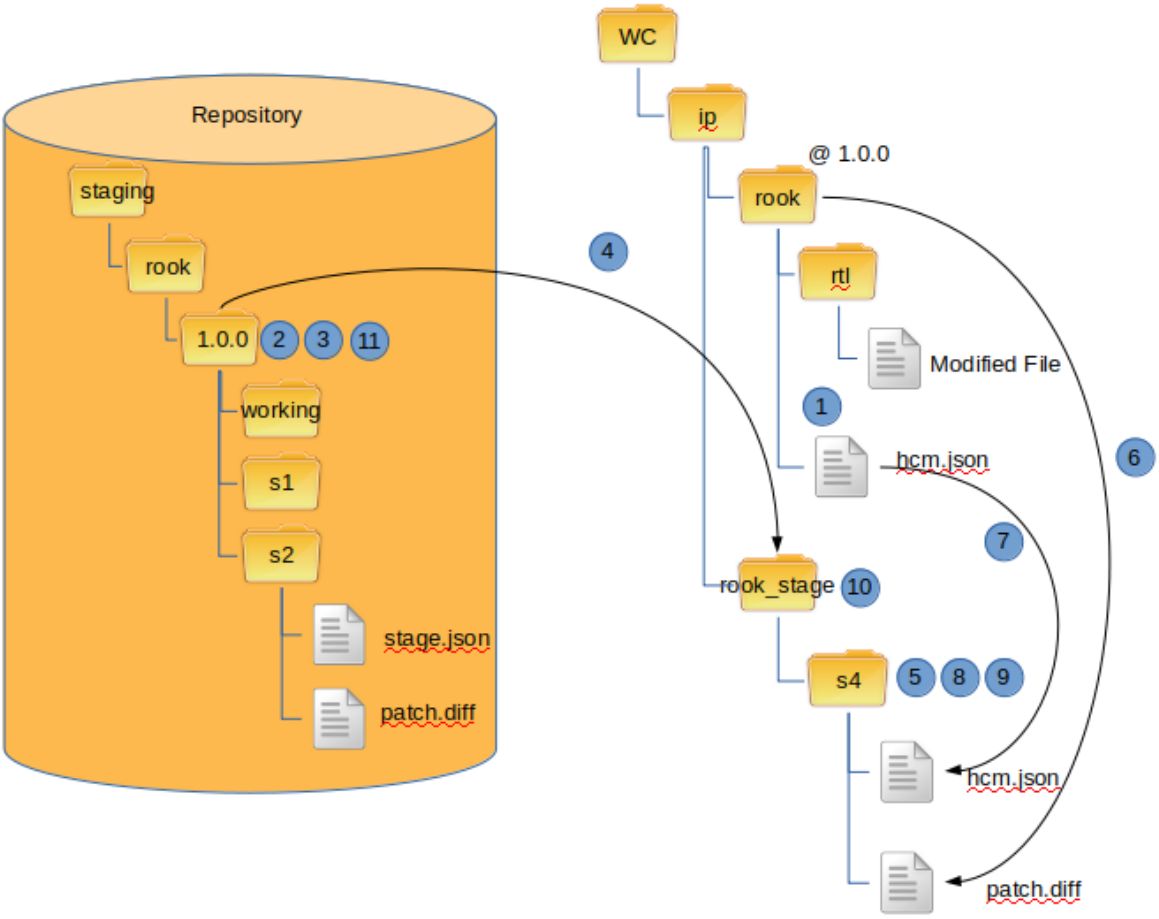
1. Perform an **svn status** and check for uncommitted modifications
2. Perform an **svn log -R** and parse out all the Last Changed Rev values.
3. Extract the Last Changed Rev of the hcm.json file
4. Compare the Last Changed Rev of every file and directory against the hcm.json file
    1. Any file with a rev higher than hcm.json is a committed modified file.

12.13 Staging

Note: This is still under development.

Staging is the first step in the Staging-Release method of publishing components. It uses svn diffs to transfer local changes to staging area for inclusion into a future component version. This command will work between repositories.

The following diagram shows the steps necessary to stage a component:



1. Read hcm.json file of component
2. Lock the version directory of the component in the staging repository
3. Perform an svn ls command to get a list of all directories
4. Perform svn checkout of staging version directory with the –depth empty option
5. Create new staging directory to hold staging information
6. Create diff of component and store under new staging directory
7. Update hcm.json file and include staging information
8. Add new staging directory
9. Commit new staging directory
10. Remove staging directory
11. Remove lock.

Note: This is still under development

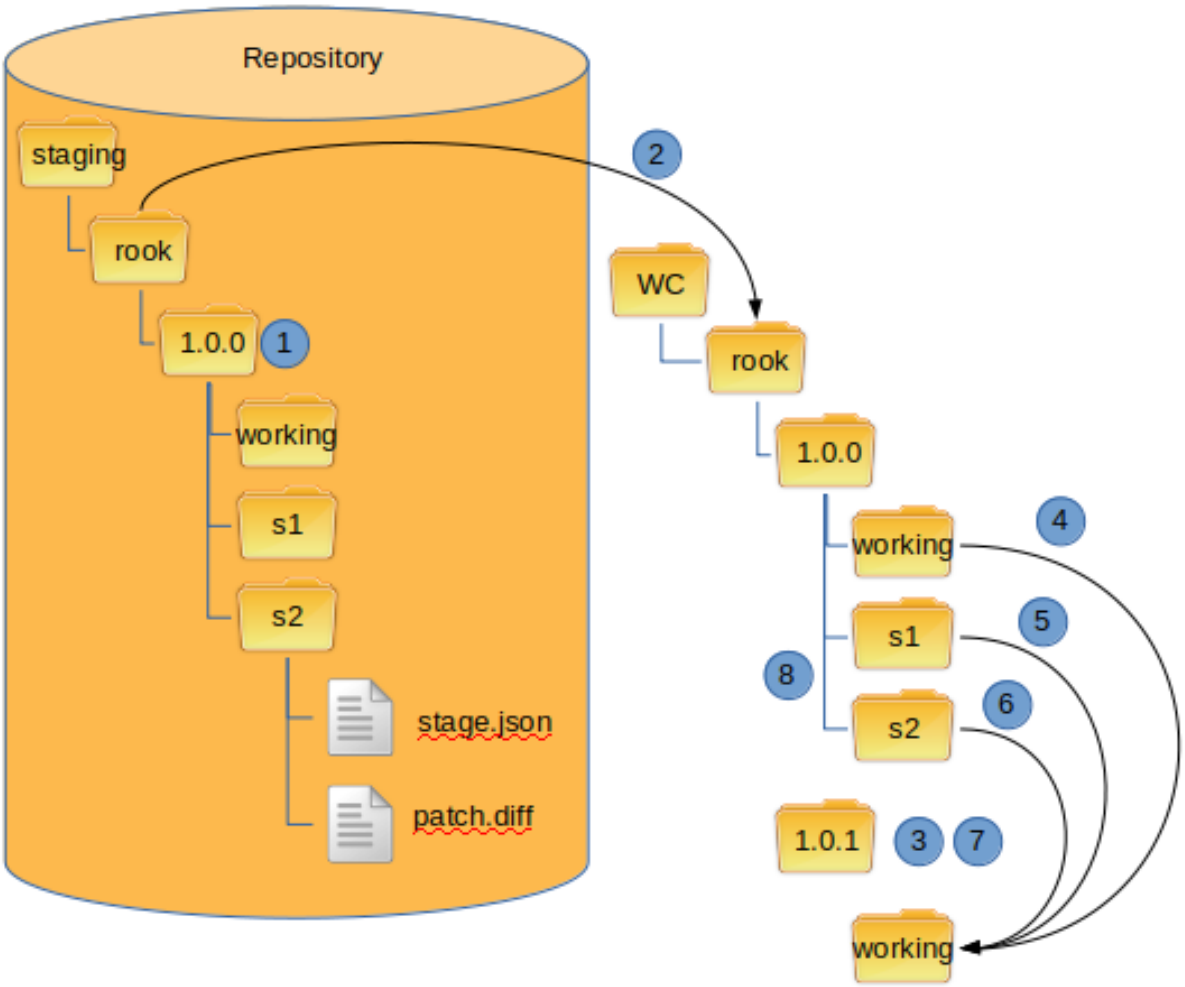
12.14 Merging Staged Updates

Note: This is still under development.

This is the second step in the Staging-Release method of publishing components. At this step, there are one or more updates to an existing version of a component. These updates must be vetted and any desired updates incorporated into a new version.

Given the nature of the effort, this step is entirely manual. HCM can not help at this step.

The following diagram shows the steps necessary to stage a component:



1. ??? Forgot what this step was. I don't think it was a lock.
2. Check out component that has staged updates
3. Review staged updates and svn add a new version directory based on the updates
4. svn copy the working directory on the current version
5. Apply the first patch to the new working directory and review the results.

6. Apply the successive patches to the new working directory and review the results.
7. Commit final updates to the new version
8. svn delete the stages that were incorporated

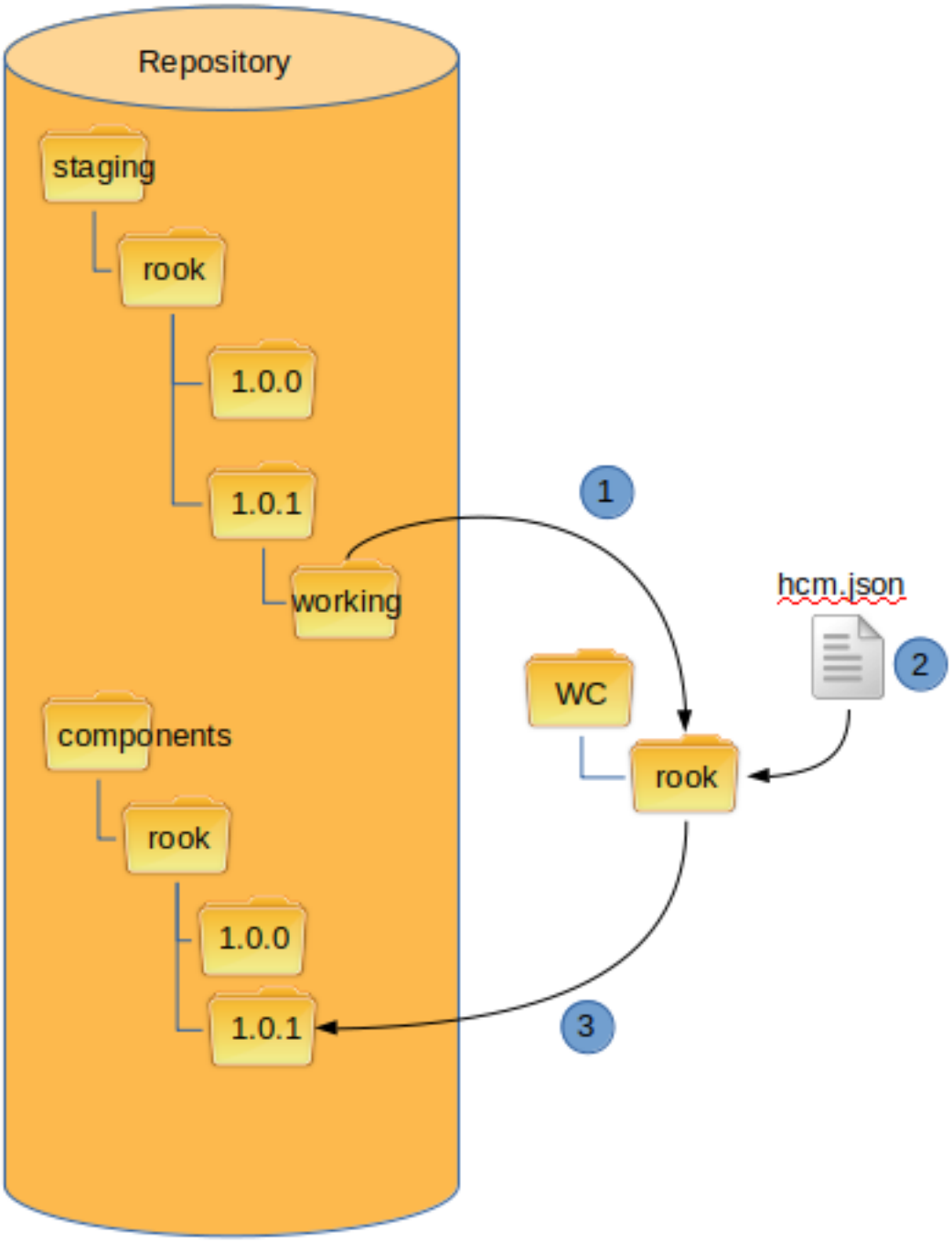
Note: This is still under development

12.15 Releasing Staged Component

Note: This is still under development.

This is the final step in the Staging-Release method of publishing components. At this step, all updates for a component version have been merged. The component is now ready to be released.

The following diagram shows the steps necessary to stage a component:



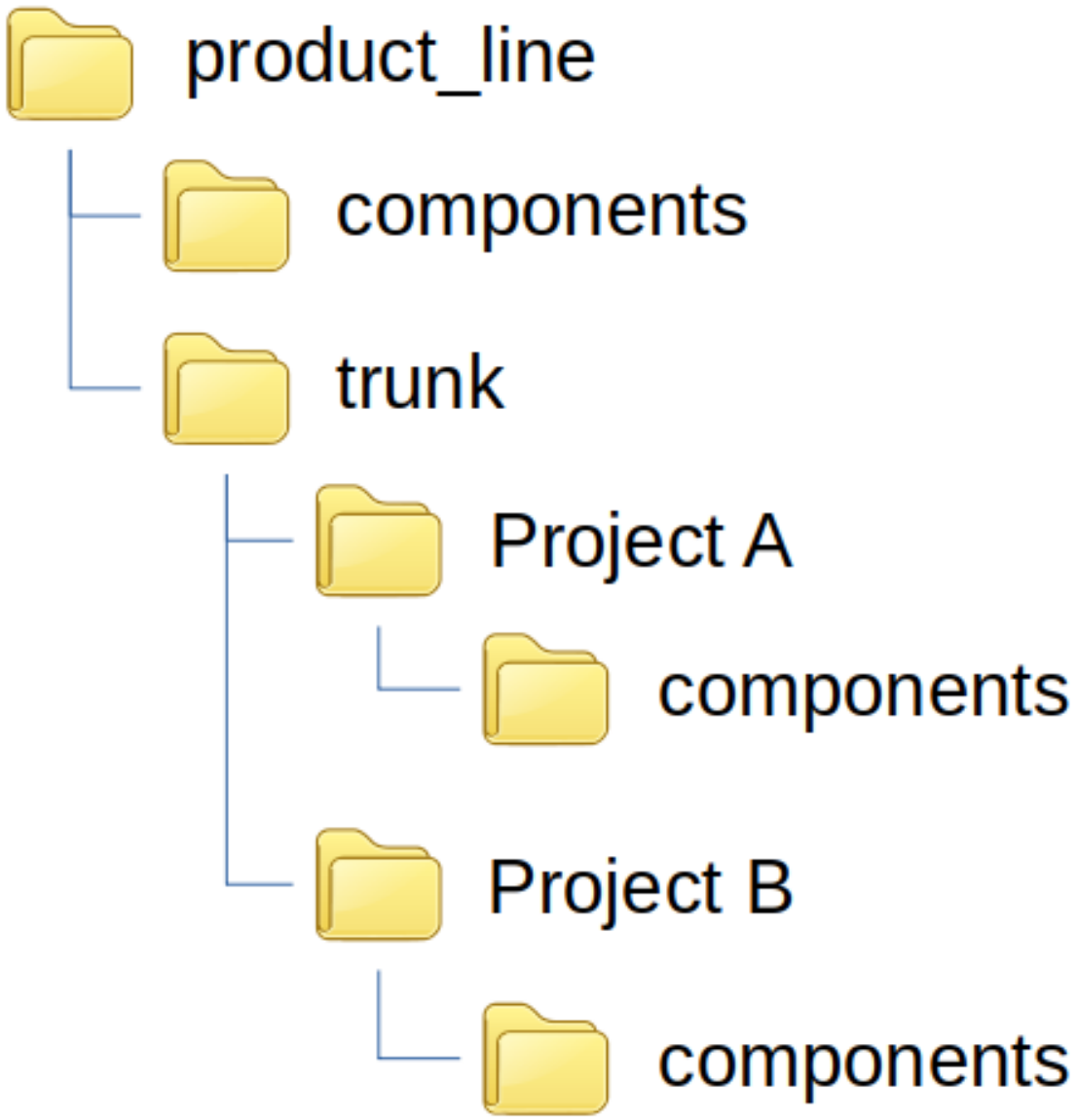
1. Check out staged component version to be released.
2. Update hcm.json file
3. Publish component to the new version directory

Note: This is still under development

CHAPTER 13

Scenarios

This section will cover various sitiations and how to handle them with HCM. Each scenario is assuming the repository structure shown below:



- The repository represent a product line
- The components directory is at the base of the repository
  - This is where components between products will be shared
- There are two projects under the trunk directory
- Under each project is a components directory
  - This is where components will be developed

13.1 Scenario 1: Share component between projects in same repo

13.1.1 Setup

In this scenario there are two projects: Project A (PA) and Project B (PB). PA has been in development for several months and has a stable code base. The code has been through several Quality Control Tools (QCTs), verification and has been tested on hardware.

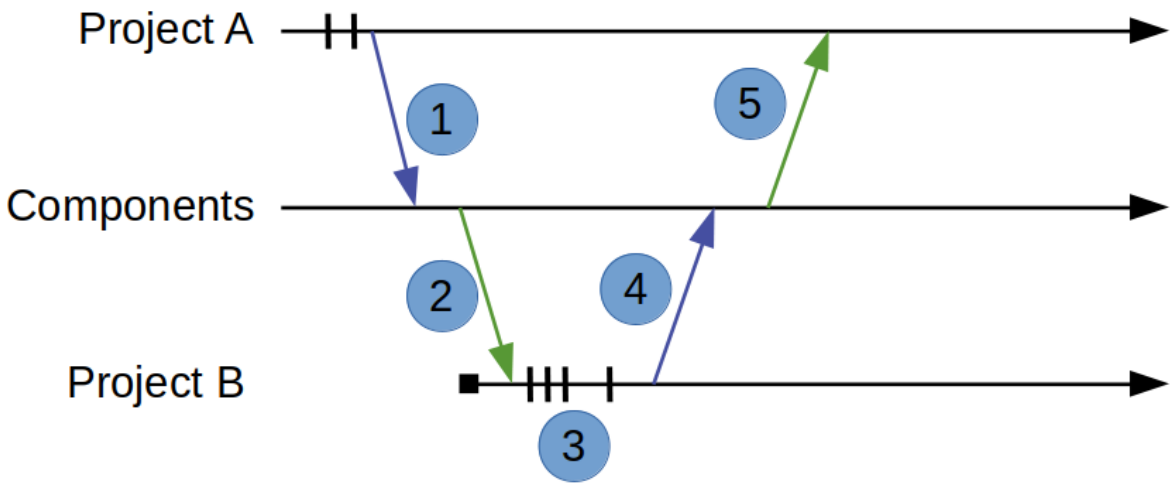
PB has started up and wants to re-use the rook component PA has developed.

13.1.2 Goals

1. Show how a single component can be transferred between PA and PB
2. Show how updates from PB can be fed back to PA
3. Show how this can be accomplished without merging

13.1.3 Workflow

The following diagram shows the steps in this scenario:



1. Project A publishes the rook component version 1.0.0 to the component directory.
2. Project B installs version 1.0.0 of the rook component to it's project working copy.
3. Project B finds a bug and commits local changes until the bug is fixed.
4. Project B publishes version 1.1.0 of the updated rook component to the component directory.
5. Project A sees there is an update and decides to install version 1.1.0 of the rook component.

These are the commands

Task	Project A	Project B
PA Publishes rook	hcm publish rook 1.0.0 –url $REPO_URL/components -f release_notes.txt	
PB installs rook		hcm install rook svn ci rook -m “Installing rook version 1.0.0”
PB fixes bug		vim rook/rtl/rook.vhd svn ci rook -m “Fixed bug.”
PB publishes rook		hcm publish rook 1.1.0 -f release_notes.txt
PA looks for updates	hcm list hcm show rook –upgrades	
PA installs rook	hcm install rook svn ci rook -m “Installing rook version 1.1.0”	

13.2 Scenario 2: Project needs to make an update but does not want to use a newer version

13.2.1 Setup

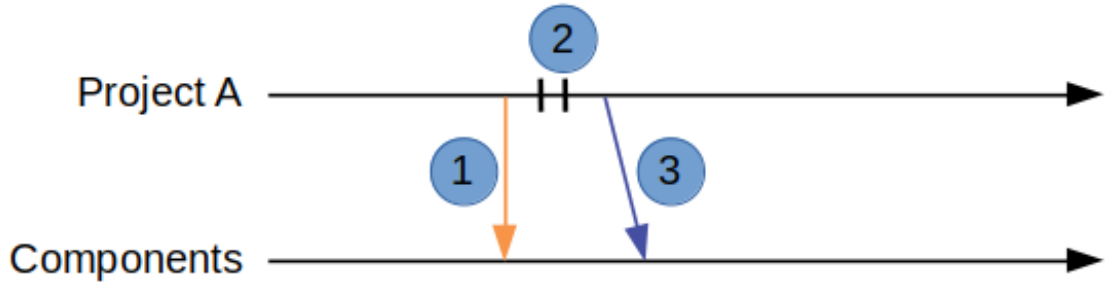
In this scenario Project A (PA) is getting ready to release code to production. PA is at version 1.0.0 of the rook component. PA has to make some minor changes to the rook component. PA checks for any updates to the rook component and finds a version 1.1.0. Due to the phase of the program, PA does not want to bring in the changes for version 1.1.0.

13.2.2 Goals

1. Show how Project A can still make local changes
2. Show how Project A can publish without affecting 1.1.0

13.2.3 Workflow

The following diagram shows the steps in this scenario:



1. Project A checks for updates to the rook component

2. Project A makes and commits changes locally
3. Project A publishes version 1.0.1 of the updated rook component to the component directory.

These are the commands

Task	Project A
PA looks for updates	hcm list hcm show rook –upgrades
PA makes changes	vim rook/rtl/rook.vhd svn ci rook -m “Added header.”
PA Publishes rook	hcm publish rook 1.0.1 -f release_notes.txt

13.3 Scenario 3: Project needs to make an update but sees an upgraded version

13.3.1 Setup

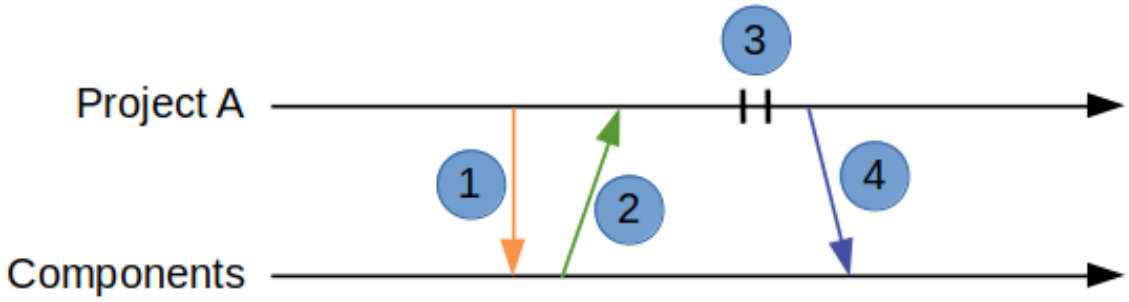
In this scenario Project A (PA) wants to make a change to a component. PA is at version 1.0.0 of the rook component. PA has to make some changes to the rook component. PA checks for any updates to the rook component and finds a version 1.0.1. PA wants to include changes from 1.0.1.

13.3.2 Goals

1. Show how Project A can make changes to an upgraded version of a component
2. Show how Project A can make changes available to other projects

13.3.3 Workflow

The following diagram shows the steps in this scenario:



1. Project A checks for updates to the rook component
2. Project A installs version 1.0.1
3. Project A makes and commits changes locally
4. Project A publishes version 1.1.0 of the updated rook component to the component directory.
5. Project A installs published version

These are the commands

Task	Project A
PA looks for updates	hcm list hcm show rook –upgrades
PA installs rook	hcm install rook svn ci rook -m “Installing rook version 1.0.1”
PA makes changes	vim rook/rtl/rook.vhd svn ci rook -m “Fixing bug with movement.”
PA Publishes rook	hcm publish rook 1.1.0 -f release_notes.txt

13.4 Scenario 4: Bringing in a component from a long lived branch

13.4.1 Setup

In this scenario Project A (PA) has a component it has modified on it’s own branch.

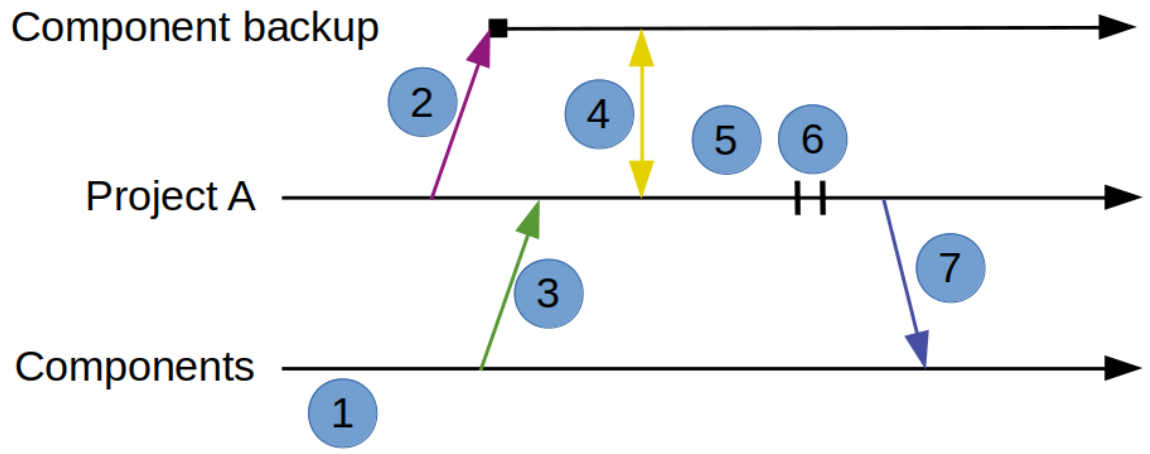
- The component is not under HCM control.
- There is a similar component that is under HCM control.
- The published version of the component is 1.1.0
- PA wants to merge their changes with what is published.

13.4.2 Goals

1. Show how Project A can merge their changes with the published version.
2. Show how Project A can make changes available to other projects
3. Show how Project A can start to use HCM to control the component

13.4.3 Workflow

The following diagram shows the steps in this scenario:



1. Project A checks the repository for the component
2. Project A copies the existing component directory to a backup
3. Project A installs the latest version of the component
4. Project A compares the contents of the backup and installed version
5. Project A merges the changes from the backup to the installed version manually
6. Project A commits changes locally
7. Project A publishes version 1.2.0 of the updated component to the component directory.

These are the commands

Task	Project A
PA checks repository	Use a repo browser or a web browser to check what version of the component is installed.
PA makes backup component	cp -r rook rook_backup
PA installs component	hcm install rook svn ci rook -m “Installing version 1.1.0 of rook”
PA compares backup to installed component	Using a tool specifically made for comparing directories will help.
PA merges backup and installed component	Move, add, modify, and/or delete files as necessary to change the installed version to the version you want to publish.
PA commits merged component	svn ci rook -m “Merged rook with version 1.1.0 of “rook”
PA publishes merged component	hcm publish rook 1.2.0 -f release_notes.txt

CHAPTER 14

Contributing

I welcome any contributions to this project.

There are several ways to contribute:

1. Bug reports
2. Code base improvements
3. Feature requests
4. Pull requests

14.1 Bug Reports

If you run into anything that is not handled correctly, please submit an issue. When creating the issue, use the **bug** label to highlight it. Fixing bugs is prioritized over feature enhancements.

14.2 Code Base Improvements

HCM started out to solve a problem and improve my Python skills. The learning part is still on going, and I am sure the code base could be improved. I run the code through *Codacy* and *Code Climate*, and they are very helpful. However, I would appreciate any suggestions to improve the code base.

Create an issue and use the **refactor** label for any code which could be improved.

14.3 Feature Requests

Let me know if there is anything I could add to make HCM easier to use.

If you have an idea for a new feature, create an issue with the **enhancement** label.

14.4 Pull Requests

Pull requests are always welcome. I am trying to follow a Test Driven Development (TDD) process. If you do add a new feature or fix a bug, I would appreciate a new or updated test to go along with the change. If not, then I will add a test to cover any updates.

I use *Travis CI* to run all the tests. *Codacy* and *Code Climate* are my quality control tools. Code coverage is reported by *Codcov*.

Travis CI will run these tools when a pull request is made. The results will be available on the pull request Github page.

CHAPTER 15

Indices and tables

- genindex
- modindex
- search