Package-izing a Problematic Solution

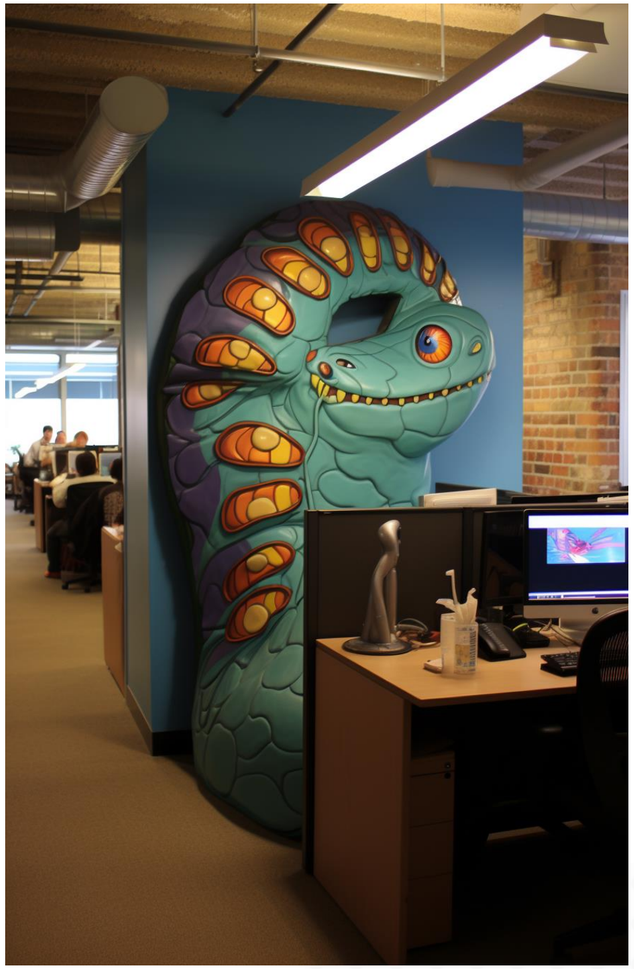
Chris B. Behrens

Software Architect

@chrisbbehrens

Code that hasn’t changed shouldn’t be rebuilt

The Package Project



A good customer

A very common problem

A very difficult runtime environment

No local debugging

Mostly just a handful of projects

Most of the work of the builds was waste

Package the stale dependencies and store them in Artifactory

And move to repository-per-project

The Universal Packaging Strategy

Presenting the Packaging Approach

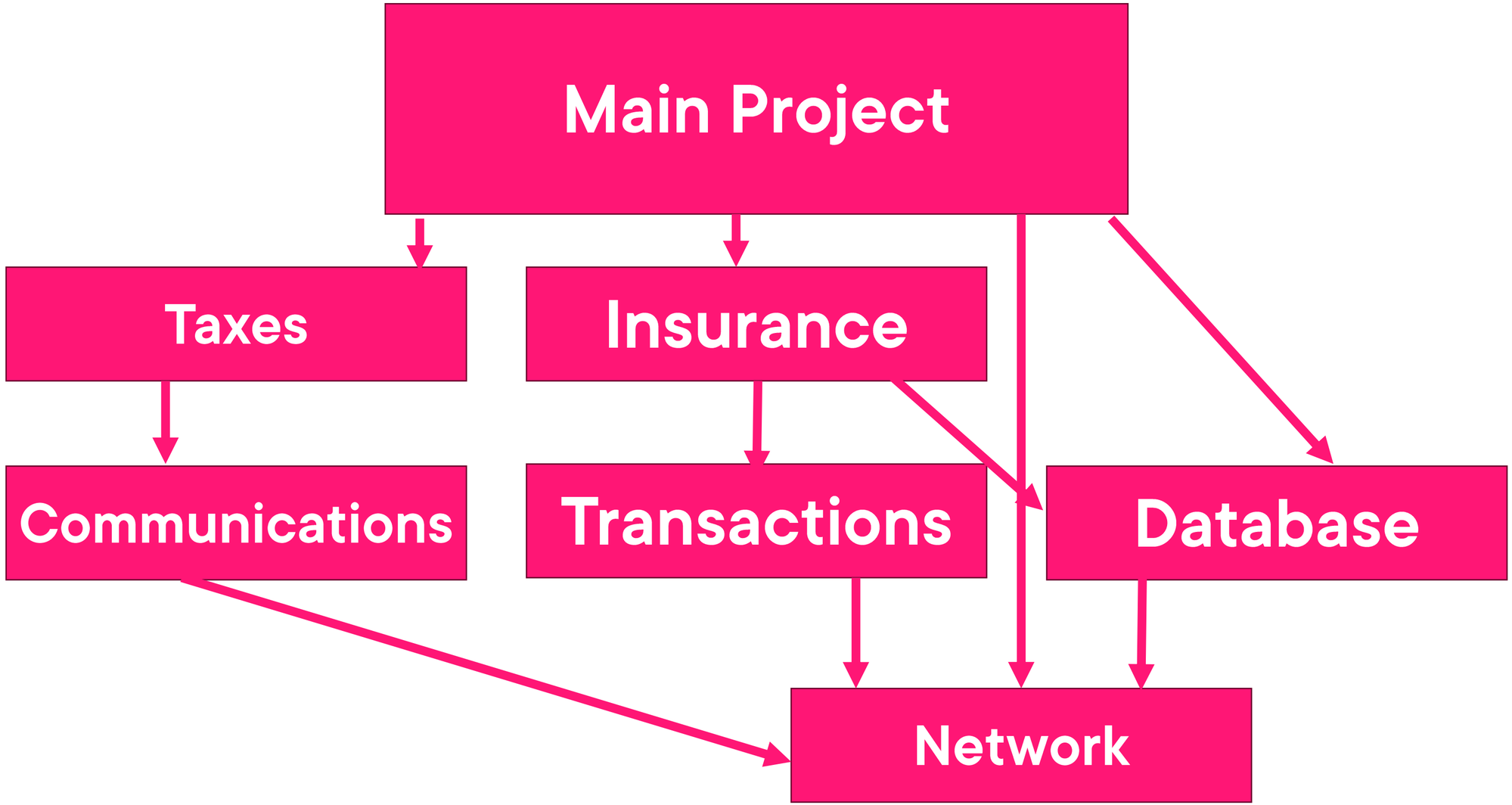
Opinions emerged

So, we mapped out the order of the migration

That was an hour wasted

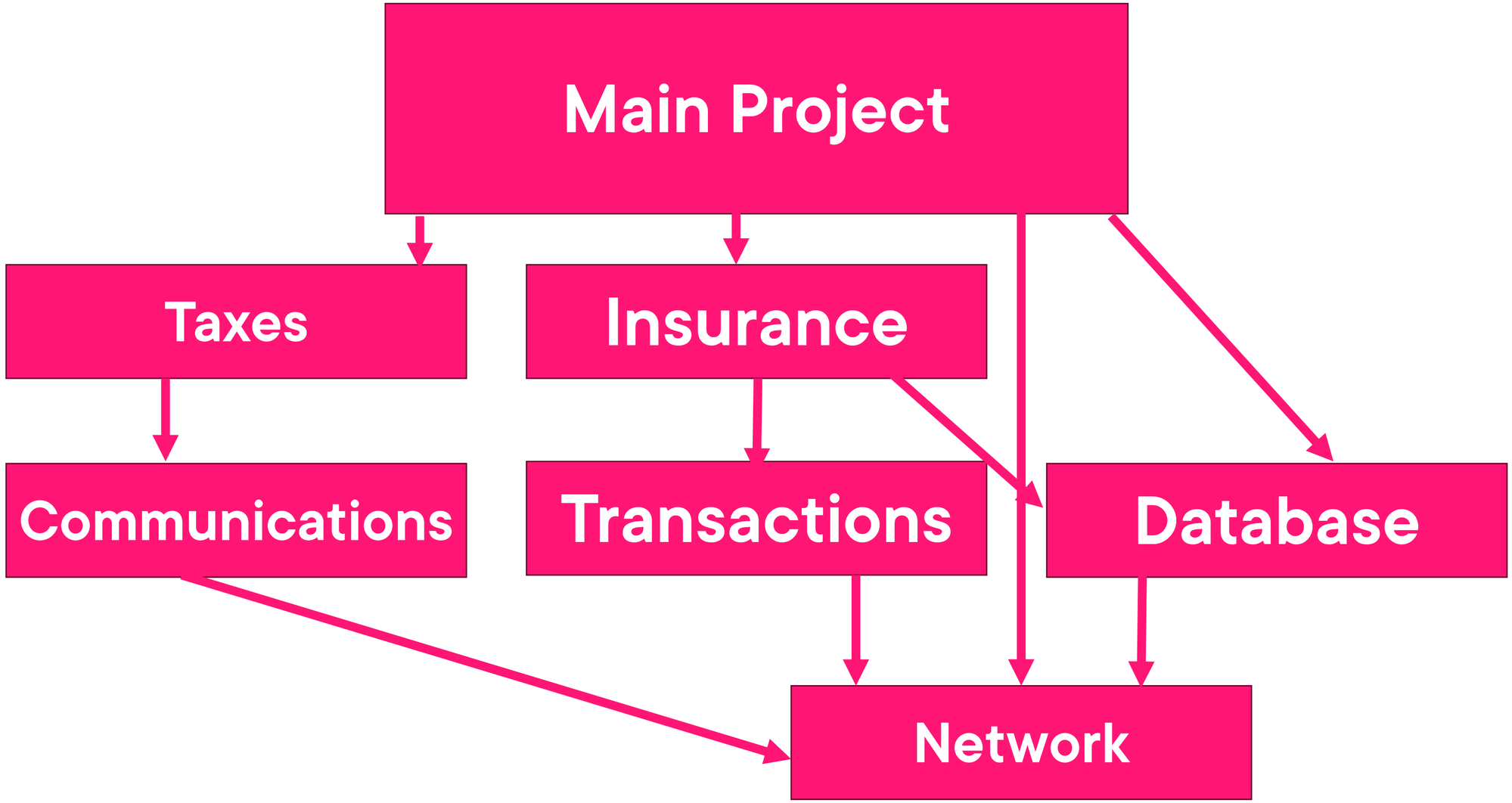
Every dependency chain has a set order it can be packaged

The Dependency Order



Well, you could just include the dependency dlls into the package along with the result of compilation...

The Right (But Painful) Way

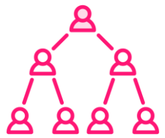


This Is Universal

This applies to all project hierarchies

A packaging cheat sheet

The Process



Identify base dependencies

Package them and identify the packages which rely on the project you just packaged and modify the project to rely on your new package

Using the project you modified to use the package in step #2, repeat step #2, i.e., package the project and modify dependent packages to point to the new package

Repeat until all projects except the top-level consumer are packaged

One of These Days

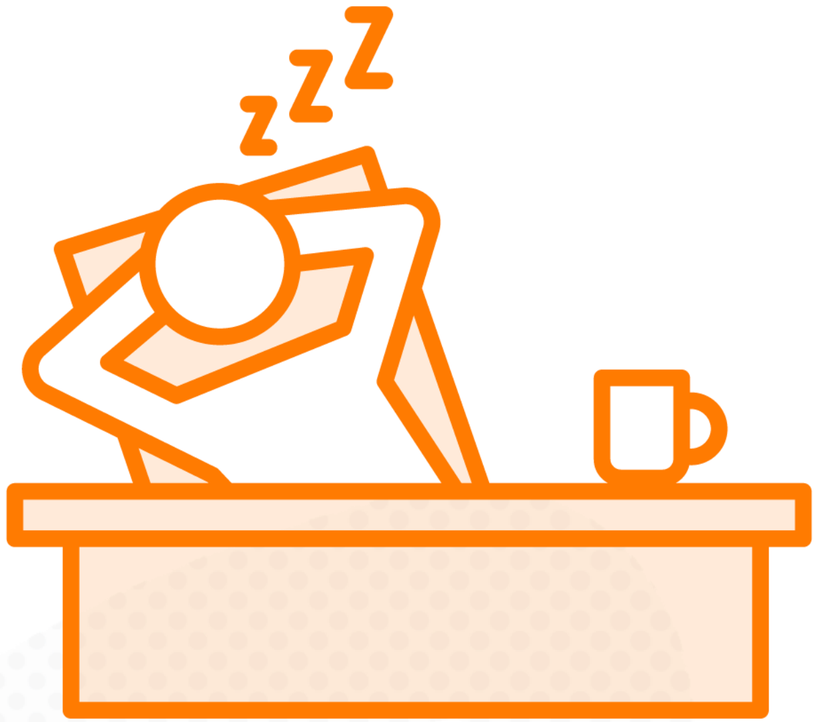
This process is entirely automatable

Two and four sound a lot like a loop

I've been meaning to create a command line tool, an add-in, some tool to do this

Automate this if you have a bunch of projects

Maybe you'll be me to the punch



Up Next:

A Simple Demonstration

Demo

- Look at our dependency order
- Figure out the canonical order of packaging
- Package the project at the bottom of the tree
- Push our package to Artifactory
- Modify the next project in the chain to rely on it

Repositories and Authentication

This Has Been Fine

Better to start with credentials

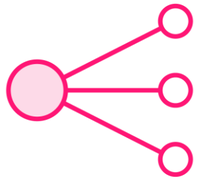
But Artifactory hasn't needed them yet

Nothing that isn't available everywhere else

Repository Types

Local: "A physical, locally-managed repository into which you can deploy artifacts."

Remote: "A caching proxy for a repository, which is managed at a remote URL."



Virtual: "An aggregated repository that combines both local and remote repositories under a common URL, which is used to create a controlled domain that facilitates searches and the resolution of artifacts."

One More Thing

We care about protecting our packages

We don't care remotely as much about our remote

Demo

- Look at our repo settings
- Test out anonymous authentication
- Force authentication for our local repo
- Authenticate with our current username and password
- Create a dedicated build credential set
- Authenticate with that

Demo

Create a new Jenkins Build

Which builds just our Satellite project

Then packs it

Then pushes the result up to Artifactory

Update our Communication package

To rely on the package instead of the project

Remove the Satellite project from the solution

Where We Go from Here

We could package the Communications project

You can set up debug symbols in the feed as well

What about the version?

How I usually do it

Pull from a file

Summary

A universal packaging strategy

The canonical order of dependency packaging

Determined that order from the build order

Learned about Artifactory repository types

Required authentication on our repos

Created a build to package and push our project to Artifactory

Executed our first build to verify that everything was continuing to work