



Sviluppo software in gruppi di lavoro complessi¹

Mattia Monga

Dip. di Informatica
Università degli Studi di Milano, Italia
mattia.monga@unimi.it

Anno accademico 2024/25, II semestre

Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging" nello sviluppo

¹ © 2025 M. Monga. Creative Commons Attribuzione — Condividi allo stesso modo 4.0 Internazionale. <http://creativecommons.org/licenses/by-sa/4.0/deed.it>



Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging" nello sviluppo

Lezione X: *Dependency hell*



Dove siamo?

Come ci si organizza? *"The tar pit"* Sviluppare *software* necessita sforzi collettivi coordinati: gruppi di lavoro complessi con obiettivi in rapida evoluzione e innumerevoli *concern* intrecciati rendono molto difficile la divisione del lavoro

Come si gestiscono i manufatti? La produzione del *software* consiste principalmente nella modifica di *file*: i sistemi di *configuration management* permettono di tenere sotto controllo l'evoluzione delle revisioni

Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging" nello sviluppo



Collaborare in un gruppo di lavoro complesso

La collaborazione ordinata richiede spesso parecchio lavoro aggiuntivo.

- Un caso in "famiglia": <https://github.com/scipy/scipy/pull/6658>
- Anche un programmatore eccezionalmente dotato come Sebastiano Vigna, deve spendere parecchie energie per *incastrare* il proprio contributo nello sforzo collettivo.
- Le *policy* aziendali (o di "kibbutz") sono ormai diventate una componente essenziale del lavoro dello sviluppatore

Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging" nello sviluppo

Dipendenze



Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging"
nello sviluppo

Qualsiasi applicazione **dipende** da componenti *software* fuori dal controllo del produttore:

- kernel
- device driver
- librerie di sistema
- **librerie di supporto**

65

Gnome-calculator



Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging"
nello sviluppo

```
$ ldd $(which gnome-calculator)
libgtk-3.so.0 => /lib/x86_64-linux-gnu/libgtk-3.so.0 (0x00007f1bbc404000)
libgdk-3.so.0 => /lib/x86_64-linux-gnu/libgdk-3.so.0 (0x00007f1bbc10d000)
libpango-1.0.so.0 => /lib/x86_64-linux-gnu/libpango-1.0.so.0 (0x00007f1bbbec1000)
libatk-1.0.so.0 => /lib/x86_64-linux-gnu/libatk-1.0.so.0 (0x00007f1bbbc9b000)
libgio-2.0.so.0 => /lib/x86_64-linux-gnu/libgio-2.0.so.0 (0x00007f1bb904000)
libgobject-2.0.so.0 => /lib/x86_64-linux-gnu/libgobject-2.0.so.0 (0x00007f1bbb6000)
libglib-2.0.so.0 => /lib/x86_64-linux-gnu/libglib-2.0.so.0 (0x00007f1bbb39d000)
libcalculator.so => /usr/lib/x86_64-linux-gnu/gnome-calculator/libcalculator.so
libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f1bba884000)
libpangocairo-1.0.so.0 => /lib/x86_64-linux-gnu/libpangocairo-1.0.so.0 (0x00007f1bba363000)
libcairo.so.2 => /lib/x86_64-linux-gnu/libcairo.so.2 (0x00007f1bba363000)
libgdk_pixbuf-2.0.so.0 => /lib/x86_64-linux-gnu/libgdk_pixbuf-2.0.so.0 (0x00007f1bb9d85000)
libxml2.so.2 => /lib/x86_64-linux-gnu/libxml2.so.2 (0x00007f1bb9d85000)
libgmodule-2.0.so.0 => /lib/x86_64-linux-gnu/libgmodule-2.0.so.0 (0x00007f1bb9d85000)
...
```

In totale 83 componenti!

66

Dipendenze di sviluppo



Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging"
nello sviluppo

Va un po' meglio con i linguaggi interpretati: alle dipendenze di sistema generalmente sopperisce l'interprete (ma non sempre: con la macchina virtuale Java per esempio può essere piuttosto faticoso utilizzare specifiche librerie grafiche).

- Un'applicazione usa librerie per non 'reinventare la ruota'
- Evitare la sindrome NIH
- Ma anche evitare le dipendenze inutili:
<https://redd.it/4bjss2>

Le dipendenze vanno il più possibile esplicitamente documentate e motivate

67

Dipendenze e pacchettizzazioni



Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging"
nello sviluppo

Abbiamo già discusso che distributori come Debian devono gran parte del loro successo alla ricca documentazione delle dipendenze:

- Ogni pacchetto è regolato da un *control* file, che specifica le caratteristiche
- le dipendenze: *Depends*, *Recommends*, *Suggests*, *Enhances*, *Pre-Depends*
- gli script da eseguire per mantenere l'integrità del sistema: *preinst*, *postinst*, *prerm*, *postrm*
- la priorità: *Required*, *Important*, *Standard*, *Optional*, *Extra*

68

Il "packaging" nello sviluppo



Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging" nello sviluppo

Il problema esiste non solo a livello di *sistema*, ma anche di singola *applicazione*. (DLL hell)

- Riproducibilità
- Ambienti di "scripting" per i quali non sono possibili compilazioni "statiche"
- Gestione di installazioni concorrenti di diverse versioni

69

Python



Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging" nello sviluppo

Esaminiamo il caso di Python, ma considerazioni analoghe valgono ormai per moltissime piattaforme di sviluppo (npm, stack, ...).

Onnipresenti poi i sistemi di distribuzione centralizzata:

- PHP Pear
- CPAN Perl
- CTAN T_EX
- MELPA Emacs '
- ...

70

Esempio: Python, documentazione delle dipendenze



Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging" nello sviluppo

Python fornisce un meccanismo per documentare le dipendenze di un'applicazione: `setup.py`

```
from setuptools import setup
```

```
setup(  
    name="MyLibrary",  
    version="1.0",  
    install_requires=[  
        "requests",  
        "bcrypt",  
    ],  
    # ...  
)
```

È un sistema che permette di preparare pacchetti (inizialmente chiamati "egg" ora "wheel" (of cheese...)) distribuibili e installabili, a patto che le dipendenze siano reperibili.

71

Python: distribuzione centralizzata



Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging" nello sviluppo

Esistono poi dei punti di distribuzione centralizzata: per esempio PYPI (Python Package Index

<https://pypi.python.org/pypi>)

E naturalmente un package manager: `pip install bcrypt`

72



Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging" nello sviluppo

Ma sempre piú spesso non vogliamo installazioni "system-wide", ma "user-wide" o addirittura "application-specific".

```
$ cd ~/usr/local/src/app/
```

```
$ python -m venv VENV
```

```
$ ls VENV
```

```
bin include lib lib64 pyvenv.cfg
```

L'ambiente va attivato: `source ./VENV/bin/activate`

73



Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging" nello sviluppo

```
$ pip install pippo
```

```
$ pip freeze > requirements.txt
```

```
$ pip install -r requirements.txt
```

74



Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging" nello sviluppo

- Può non essere banale tenere aggiornato un `virtualenv` → `pipenv` o `poetry`
- c'è il problema della "riproducibilità" in ambienti di esecuzione differenti (versione di Python, sistema operativo) → `uv`
- *Source distribution vs. wheel*
- Sistemi di distribuzione indipendenti, orientati al cross-platform (prendono il controllo dell'ecosistema Python): `CONDA`
- Un nuovo standard (PEP 751)

75



Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging" nello sviluppo

Versionamento semantico

<http://semver.org/spec/v2.0.0.html> è uno standard fondamentale perché questi sistemi possano funzionare.

- Numero di versione con tre token MAJOR.MINOR.PATCH
- MAJOR cambia quando ci sono cambiamenti incompatibili nelle API
- MINOR cambia con nuove funzionalità (ma *backwards-compatible*)
- PATCH solo bugfix

76