Ensuring usability of a scientific code base

Klaus Rechert Uni Freiburg

Jurek Oberhauser Uni Freiburg

Rafael Gieschke Uni Freiburg

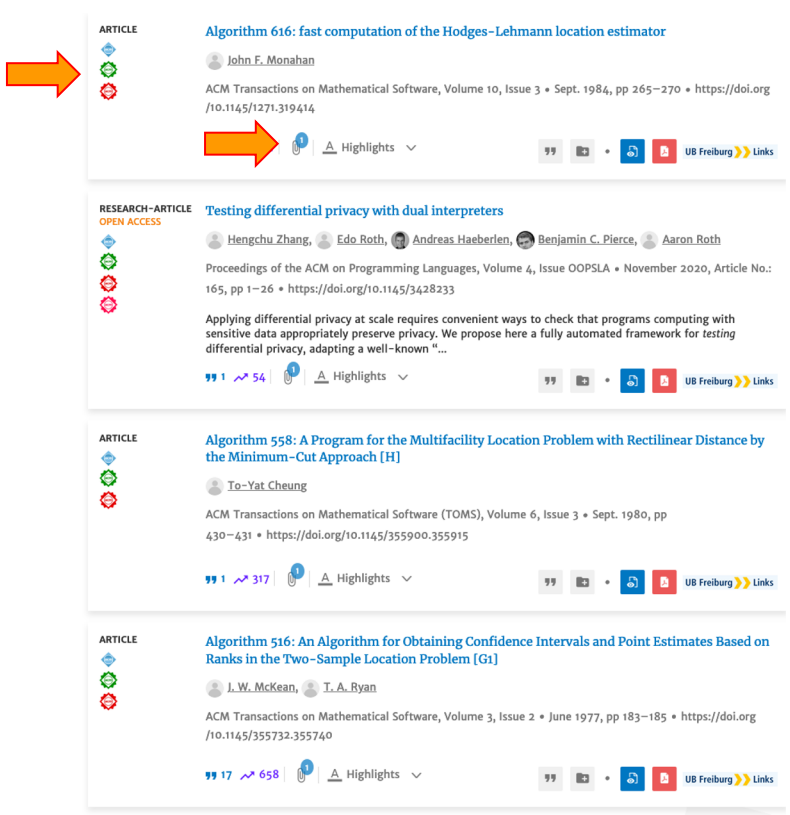
Software Preservation

- Software preservation is an important requirement to ensure re-use certain digital (data-)objects
- Re-using preserved software preservation requires infrastructure
  - Runtime dependencies (operating system, libraries, etc…)
  - Hardware or hardware equivalents (emulation)

https://www.softwarepreservationnetwork.org/emulation-as-a-service-infrastructure/

Scientific Software

- Scientific software (usually part of the publication) published as source code (archive)
  - Source code is human readable
  - Can be reviewed
  - Can be re-used and further developed by a scientific community
- To re-run with data the code has to be usually compiled to a binary or the appropriate runtime is required



Software Heritage

Software Heritage - a non-profit organisation to collect, preserve and share software.

„ Science relies more and more on software. To guarantee scientific reproducibility we need to preserve it [.] ...”

https://www.softwareheritage.org/mission/

Software Heritage

Software Heritage - a non-profit organisation to collect, preserve and share software.

„ Science relies more and more on software. To guarantee scientific reproducibility we need to preserve it [.] ...” https://www.softwareheritage.org/mission/

… and one should be able to re-run it.

Scientific Software Preservation

- Software became an essential part of computational science
- Software can be part of the publication (together with data and article)
- Software becomes an integral part of RDM
- Software then needs to be FAIR
- How?

→ FAIR 4 Research Software WG

https://rd-alliance.org/groups/fair-research-software-fair4rs-wg

→ Infrastructure and workflows are required

Archiving and Accessing Scientific Code

- Re-create (re-build) code from source requires another set of dependencies
  - → build dependencies
- Usually no formal description, implicit for a given time context

```
Unix:
  You need X11R6 and a "make" utility with the VPATH feature (e.g. GNU make).
  For serial, ethernet and audio support, you need pthreads. To use the GUI
  preferences editor, you also need GTK+ version 1.2 or better. On Linux, you
  need glibc 2.0 or better.
```

> **RPM packages**
>
> `protobuf protobuf-c protobuf-c-devel protobuf-compiler protobuf-devel protobuf-python`
>
> **Deb packages**
>
> `libprotobuf-dev libprotobuf-c-dev protobuf-c-compiler protobuf-compiler python-protobuf`

Re-Build (Scientific) Source Code

- Best outcome
  - Can be compiled with contemporary tool chain
- If not possible, tweaks to the code / build infrastructure are required
  - Libraries may have changed
  - Build tools may have changed
- Re-run of a successfully compiled binary does not automatically yield a valid / usable result, i.e. replicate a program’s run
  - Libraries or other dependencies may have changed semantics
  - Scientific workflows may require a specified set of dependencies (tool-chain) in exact versions

Example

```
import random
random.seed(1) # RNG initialization
x = 0
walk = []
for i in range(10):
        step = random.choice([-1,+1])
        x += step
        walk.append(x)
        print(walk)
# Saving output to disk
with open('results-R2.txt', 'w') as fd:
        fd.write(str(walk))
```

Python <3.2:
−1, 0, 1, 0, −1, −2, −1, 0, −1, −2

Python >=3.3:
−1, −2, −1, −2, −1, 0, 1, 2, 1, 0

From:
"Re-run, Repeat, Reproduce, Reuse, Replicate: Transforming Code into Scientific Contributions" von Fabien C. Y. Benureau und Nicolas P. Rougier.

Infrastructure and Requirements

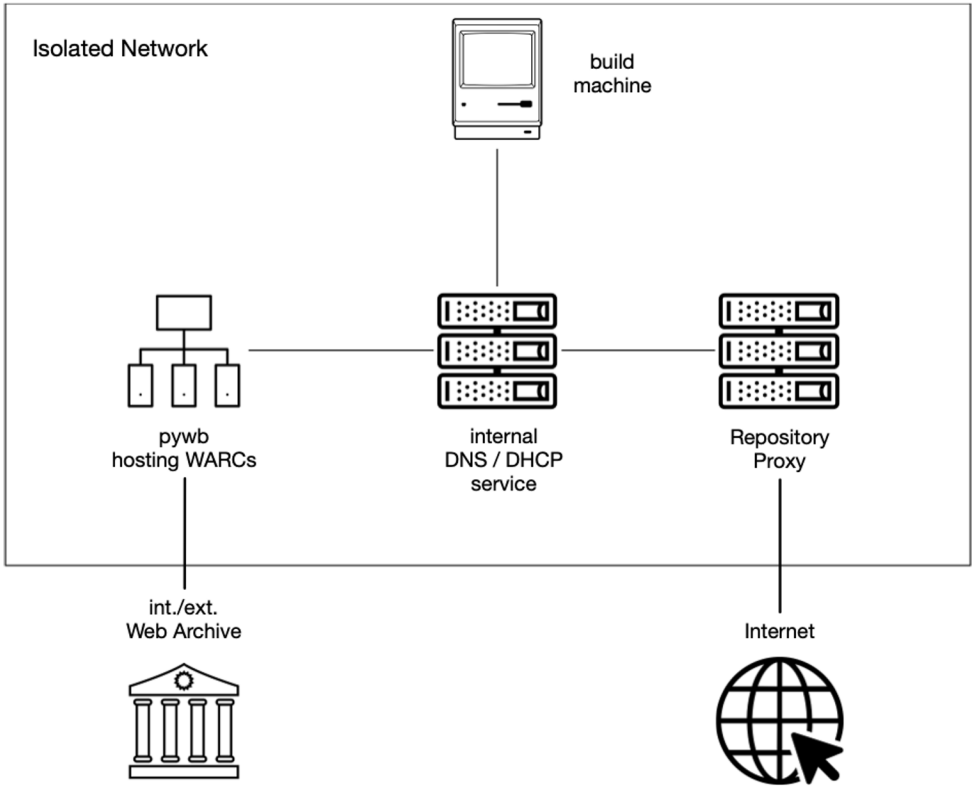
- Archival, management and access to historic (build) environments
  - Emulation infrastructure EaaS(I), CiTAR, etc.
  - Systematic collection of relevant “base” environments
  - Maintain/archive/manage external software repositories
    - Distribution repositories
    - Special “repositories” like npm, pip, CPAN, …

    → ideally have snapshots to allow implicit versioning

Historic Build Environments

Build environment

- Machine (emulator) + installed disk image
- Managed context
  - E.g. time/date settings
  - Isolated network with transparent repository mapping

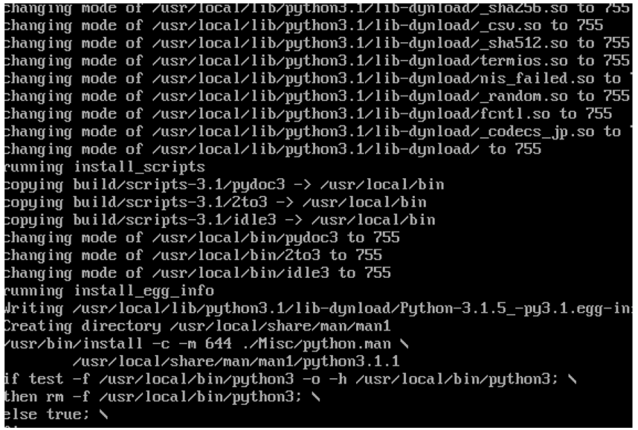


Infrastructure and Requirements

- Fetch source code “as an object”
  - EaaS plugin for Software Heritage https://github.com/Aeolic/swh-downloader
  - Supports retrieval of code by *commit hash* / Software Heritage persistent ID (SWHID)
  - Available as tar.gz archive
- Prepare code to be available in a selected environment
  - Inject as CD-ROM
    - Good operating system support, simple file system
    - Read-only medium, might complicate automation
  - Inject as auxiliary disk
    - Choose an OS supported file system
    - For automation: anticipate how the target OS manages aux. disks
  - Inject into “root” file system
    - Find root partition, detect file-system, mount and copy/extract code to user-defined directory
    - Preferred solution to support automation
  - Provide via network share (CIFS/NFS/…)

Infrastructure and Requirements

- Preparation and interactive build
  - Interactively explore build environment
  - Install dependencies as needed
  - Build!
- Keep successful builds (build environment and result) accessible
  - Keep as reference for quick inspection or/and to run with data
  - Keep as reference for other similar tasks, keep to build *knowledge*
  - Keep as reference to compare / audit binaries
  - Outcome 1:
    - Save build environment as derivative (with or without code and build result
    - Connect with source code (metadata)
  - Outcome 2:
    - Create a *build recipe* containing all preparation and build steps
    - Currently a shell / batch script
    - Requires formalization



Example Workflow

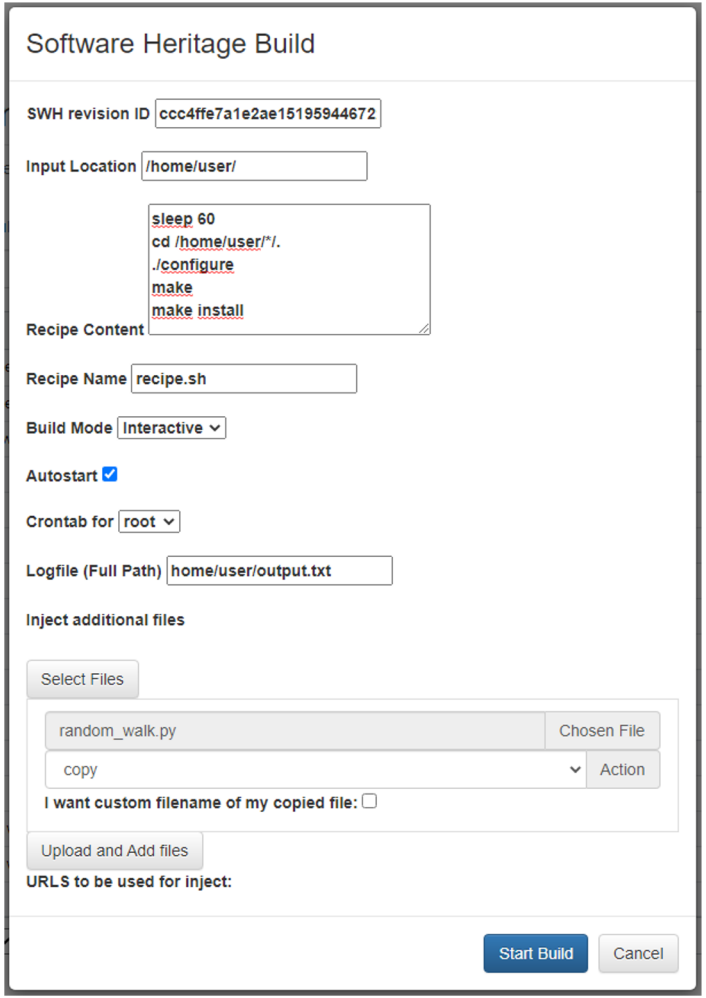
- Choose a target environment
  - either a plain base environment
  - Or previously prepared

Edubuntu 6.06 - Base V1
Fedora 1 Base V1
Fedora 7 Base V1
Mandrake 8.0 - Base V1
Mandrake Linux 5.1 - Base V1
Open SUSE 10.2 Base V1
OpenSolaris 2009.06
Red Hat 6.2 - Base V1
Red Hat 8.0 - Base V1
Red Hat 9.0 - Base V1
Scientific Linux 3.0.1 - Base V1
Scientific Linux 6.0 - Base V1
ScientificLinux 4.0 32bit
ScientificLinux 5.0 32bit
Slackware 9.0 Base V2
Ubuntu 10.04.4 Base V1
Ubuntu 10.10 + FreeCAD 0.10
Ubuntu 10.10 + R Commander 01.5-6
Ubuntu 10.10 + RKWard 0.5.3
Ubuntu 10.10 + RStudio 0.92.23
Ubuntu 10.10 Base V1
Ubuntu 4.10 Base V1
Ubuntu 5.04 Base v1
Ubuntu 5.10
Ubuntu 6.06 + FreeMat 2.0

Example environments available in the EaaSI public sandbox
https://eaasi-sandbox.softwarepreservationnetwork.org/eaasi

Example Workflow

- Choose a target environment
  - either a plain base environment
  - Or previously prepared
- Enter SWH revision ID
- Enter (or upload) recipe (content)
- Choose build mode
- Configure autostart
- If necessary, inject additional files
  - E.g. data to be processed



Infrastructure and Requirements

- Automation and API access
  - If a “recipe” is available, builds can be automated
  - POST JSON data to HTTP endpoint
    - Returns job ID
    - Wait / poll for job completion
    - Process build result
- Build result
  - Save environment, re-use with data or for next build step (pipeline)
  - Request and download result
    - ZIP/tar with binaries / output
    - Hash values or similar checks

```
{
  "softwareHeritage": {
    "revisionId": "ccc4ffe7a1e2ae151959446722bf1b58834f5e9f",
    "extract": false,
    "scriptLocation": "/libexec/swh-downloader/main.py"
  },
  "buildToolchain": {
    "environmentID": "42a20701-7aae-446a-a0e2-8f9c24ba5047",
    "inputDirectory": "/home/user/",
    "recipe": "make install",
    "autoStart": true,
    "cronUser": "root",
    "recipeName": "recipe.sh",
    "logFileLocation": "/home/user/output.txt",
    "injectOnly": false,
    "additionalInjects": ["blobstoreURL1", "blobstoreURL2"]
  }
}
```

Next Steps

- Gradually improve meta-data for past code contributions
  - Metadata “language” and representation still an open question
  - Automate (record) environment preparations
  - Share build environments and recipes
- *Continuous Access* for up-coming code contribution
  - Encourage better practice for current code contributions
    - Derive and describe dependencies
  - Encourage use of CI pipelines
    - Preserve CI setting and dependencies with code archive
    - Consolidate and archive CI dependencies (VM, docker image) if necessary
  - Integrate long-term infrastructure as soon as possible
    - Integrate automated *long-term builds* in CI/test pipelines