Improving workflow in global volunteer projects

Introduction to my Ph.D. research

Martin F. Krafft <madduck@debian.org>

Debian developer, author of *The Debian System*
Ph.D. student at the Univ. of Limerick, Ireland

FOSDEM 2006, 25/26 Feb 2006, Brussels, Belgium

Improving workflow in ~~global volunteer projects~~ Debian

Introduction to my Ph.D. research

Martin F. Krafft <madduck@debian.org>

Debian developer, author of *The Debian System*
Ph.D. student at the Univ. of Limerick, Ireland

FOSDEM 2006, 25/26 Feb 2006, Brussels, Belgium

... and some others (Zope, Plone, ...)

My motivation

- I live Debian
- Debian has some antiquated processes
  - Silly to say that Debian won't survive
  - Improvements could get us further though
- Facilitate contributions by *everyone*:

  ...

My motivation

If Jane Doe has 2 hours on a Saturday afternoon for Debian, she should not have to spend 1.5 of those figuring out how to contribute.

So you're a Ph.D.?
Wow! Just don't touch anything!

- Doing this as a Ph.D. has advantages:
  - Feeling good about spending all your time on Debian.
  - A great community.
  - Motivation to publish (think: documentation).
  - Research as backup.
  - Research as gateway to new approaches.

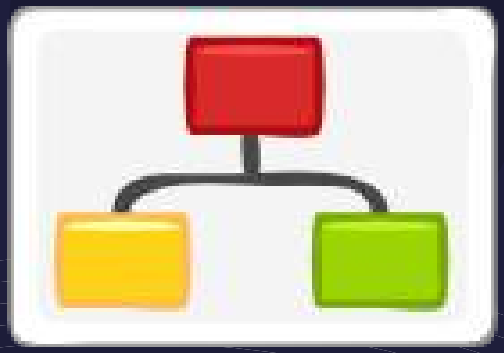
So you're a Ph.D.?
Wow! Just don't touch anything!

- Doing this as a Ph.D. is neutral:
  - A "Debian Ph.D." does not buy me anything within the project.
    (Thus motivation to actually produce results.)

So you're a Ph.D.?
Wow! Just don't touch anything!

- Doing this as a Ph.D. has disadvantages:
  - I will need to finish sometime. Urks.

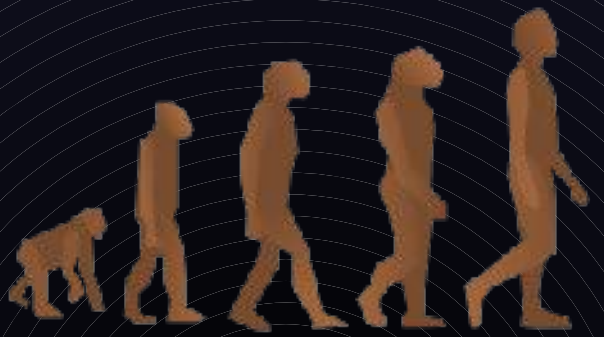
Research model



- "Participatory action research":
  - practically grounded (**not academic**)
  - aimed to solve an immediate problem
  - researcher is part of the study
  - "complex systems cannot be reduced, they are best studied by introducing changes"
    --> new AI reference

Back to the topic:

How can (some of) Debian's processes be improved/integrated to allow for more efficient cooperation?



Impact

- Make things easier for Debian contributors
- Allow for better integration of works by derivatives
- Give Debian (again) a competitive edge and more resources to walk new paths
- Also: business. "Employees as volunteers"

Research in two parts

Develop improved processes and workflows, and devise migration paths

Determine conditions under which volunteer developers adopt improved models

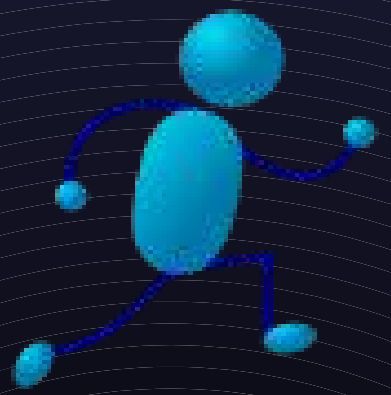
Finally, mix, stir well, enjoy!

Related concepts

- Motivation of volunteers
- Ideas are entertainment,
  real tools are worth considering
- Extreme programming / agile development
- Team development
- Documentation

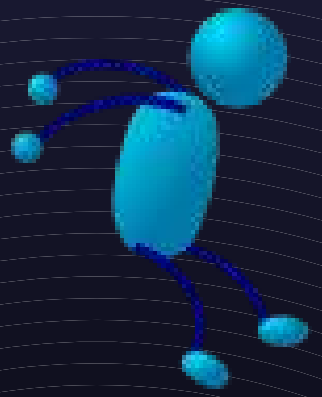
Approach

- Analyse previous adoption of new tools
  - debhelper / cdbs
  - debuild / pbuilder
  - lintian / linda
  - devscripts
  - dpatch
  - BTS user tags
  - VCS (*-buildpackage or other)
  - ...



Approach

- Previous publications and efforts
- Surveys
- Interviews with developers
- Use-case studies
- (A lot of) experimentation (trial and error)



Let's leave it at that...

on to more exciting stuff...!

Three examples of workflow improvements

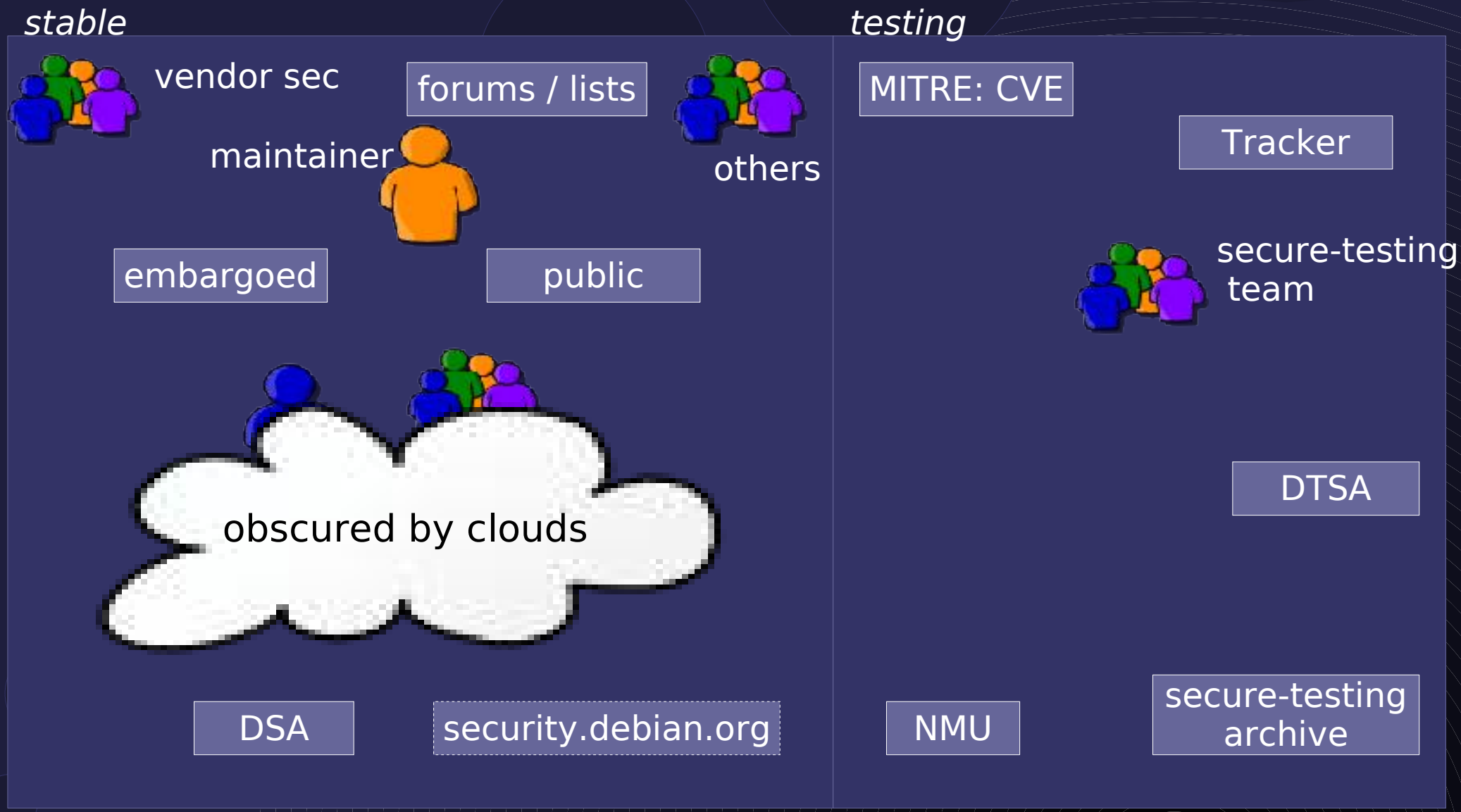
- Debian stable and testing security
- Flexible package & patch management
- The package upload

Remember...

... these are just ideas.

(They need a lot of feedback!)

Debian security: current situation



Debian security: proposed situation

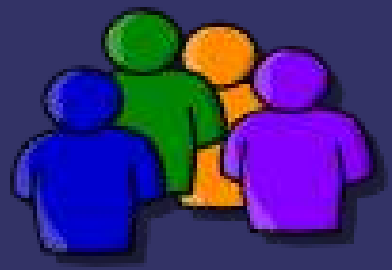
public

forums/list bugtraq, full-disc, CVE, etc.

others

review/QA

NMU



security team

maintainer

stable-security

testing security

DSA

DTSA

tracker

embargoed

maintainer

DSA

vendor sec

security.debian.org

embargoed security team

Debian security: status

- Work has begun after Debconf6
  - security.debian.org is now RR
  - Oldenburg security meeting
  - Fluctuations in the stable security team
  - dak restructuring to accomodate testing on security.debian.org
  - Bi-weekly coordination meetings

Debian security: tracker outlook

- Analyse the current tracker
  - Single 2Mb file in SVN
  - It's simple, that's why secure-testing likes it
  - People are concerned it won't scale
- Consider alternatives
  - Most important is not to alienate current contributors, who are used to the SVN tracker
  - Define and document a workflow
  - Consider using the BTS in a structured way

Flexible patch/package handling

- Differentiate between management and maintenance (somewhat arbitrary):
  - management is the package from outside: how is cooperation coordinated?
  - maintenance is inside the package: how are patches tracked

Package management

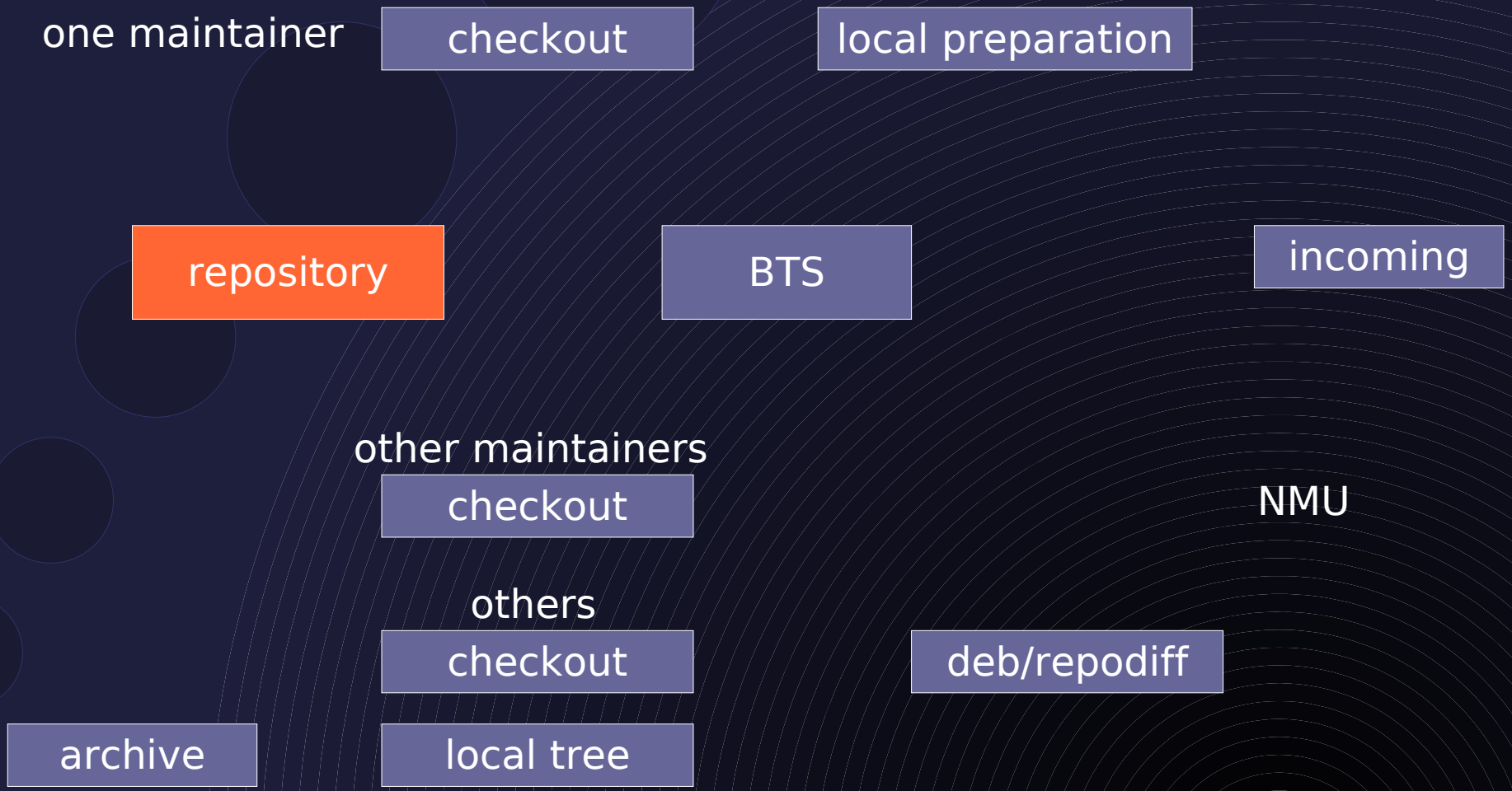
- Currently, package management is done in about three different ways:
  - local source tree
  - centralised repository
  - decentralised repository
- Probably some other ways, developers are creative

Package management: Local source tree

Package management: Local source tree

- Standard procedure
- Local tree, local package generation, upload
- Others can get the source with apt-get and prepare NMUs/diffs to be sent to the BTS
- This will (and should) always work
- Does not scale to co-maintenance with more than 2 maintainers or many/frequent contributions

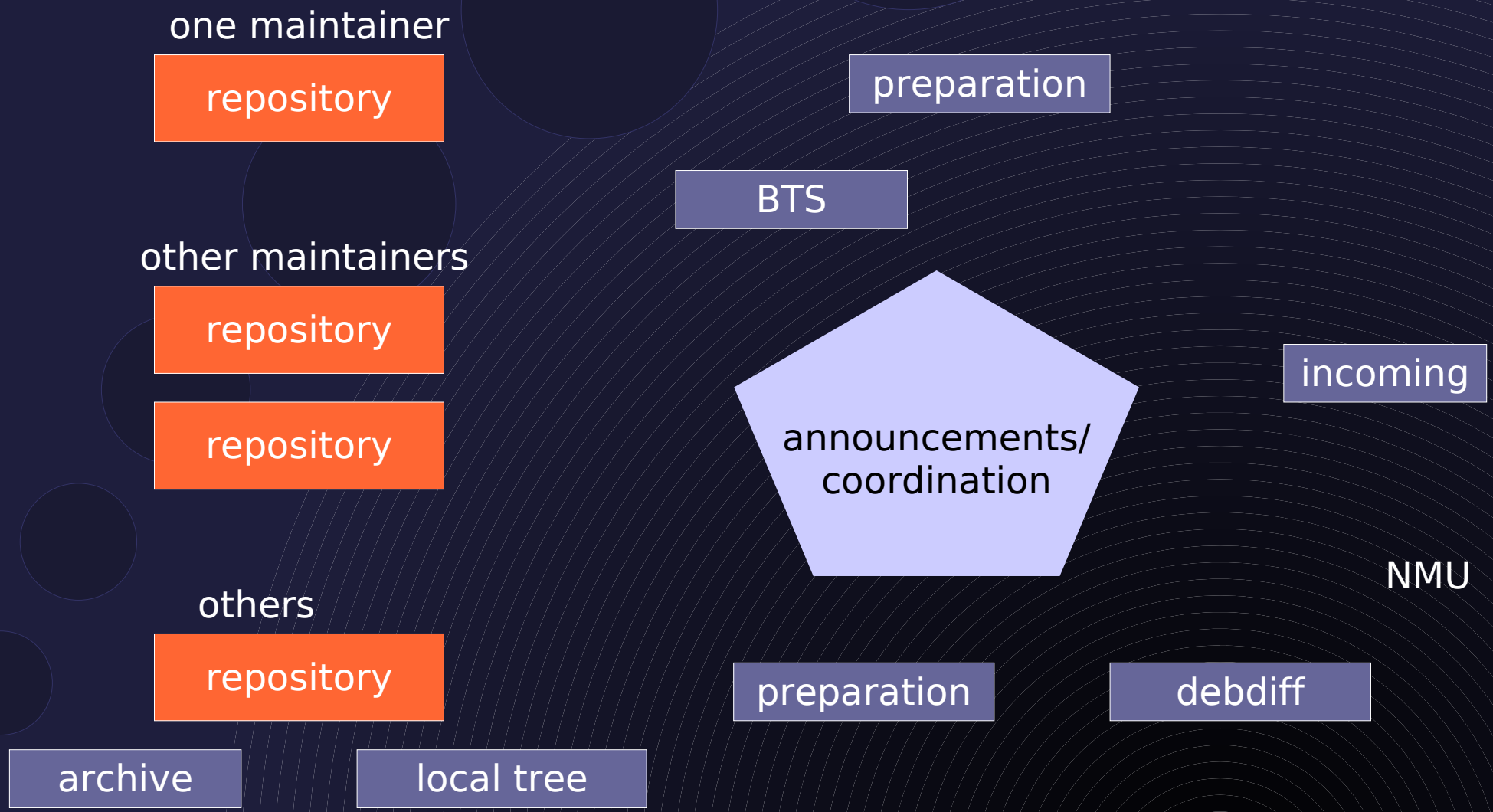
Package management: Centralised repository



Package management: Centralised repository

- Increasingly popular
- Examples: CVS, SVN
- Still, who uploads in the end? Bottleneck?
- Contributors must
  - know where to check out, or use outdated source from archive (makes merges harder)
  - learn the VCS and procedures
- Single point of failure w/o offline capability

Package management: decentralised repository



Package management: decentralised repository

- Examples: tla/baz, bzr, git, darcs
- Tight dependency on announcement/coordination medium (mailing list, patch queue manager)
- Who uploads in the end?
- Steep learning curve of decentralised VCS
- Must know where things are
- Difficult to keep track for outsiders

Package maintenance

- Approaches to track changes
- Three typical to local/no management
  - no patch separation whatsoever (single diff)
  - dpatch (dbs) / quilt
- Three are typical to VCS management
  - no patch separation within package
  - feature & integration branches
  - stgit

Package maintenance condensed

- Four patterns:
  - No tracking / single diff
  - dpatch
  - quilt / stgit
  - feature & integration branches
- Probably more, developers are creative

Package maintenance: No tracking / single diff

- Claim: by far most packages in the archive
- Procedure:
  1. Obtain latest source tree
  2. Make changes to the source
  3. (Maybe) document changes in changelog
  4. Build package
  5. Upload
- Does not scale (at all; try it on libc6)

Package maintenance: dpatch

- dpatch keeps patches separate
- Procedure:
  1. Obtain latest package source
  2. dpatch-edit-patch patch-name
  3. Make changes pertaining to one feature
  4. dpatch-edit-patch another-patch-name
  5. Make changes pertaining to another feature
  6. Build package
  7. Upload

Package maintenance: dpatch

- Keeps package tree clean
- Flat learning curve
- Patches can be scripts too
- May be difficult for contributors to identify a specific patch to manipulate
- Inter-patch dependencies are only linear
- Horrific conflict handling
- Slow

Brief intermission: Wig & Pen

- Evolutionary extension to dpkg
- Additional tarballs for subdirectories
  - ...s_1.2.orig-**xyz**.tar.gz --> ... s-1.2/**xyz**
- debian/ directory can be a tarball with separate patches to the source (dpatch)
- Support for other types of archives (bz2, zip, ...)

Package maintenance: quilt / stgit

- Multiple "stacks" of patches
- Can be used like dpatch
- Supports multiple stacks "one stack per feature"

Package maintenance: quilt / stgit

- Procedure:
  1. Obtain latest source package
  2. "Pop" your way to the patch you wish to change (or add) down the appropriate feature stack
  3. Make changes
  4. Refresh the stack(s)
  5. Build the package
  6. Upload

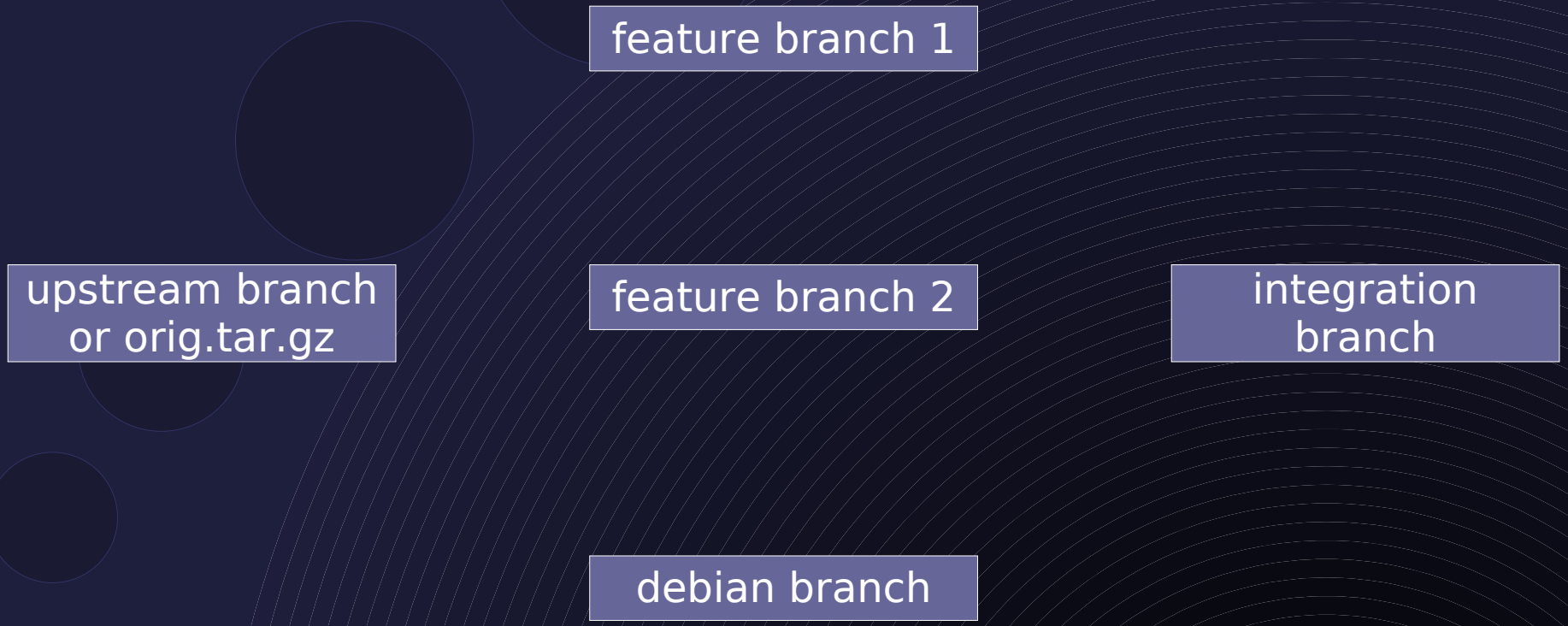
Package maintenance: quilt / stgit

- Quilt stores locally, stgit uses a git repository
- Clean separation of features
- Suboptimal merging / conflict resolution
- Functionally equivalent to branches but without the advantages of a VCS
- Linear dependencies only
- No inter-stack dependencies

Package maintenance: feature & integration branches

- Ideally with a VCS that has proper branch support: baz, bzr, git, SVN, darcs
- Keep upstream in a read-only branch (maybe possible to use orig.tar.gz instead)
- Branch anew for each feature ./debian/ is also a feature
- Merge all feature branches into a merge-only integration branch

Package maintenance: feature & integration branches



Package maintenance: feature & integration branches

- Procedure:
  1. Checkout source tree to create branch, or specific branch to work on
  2. Make and commit changes
  3. Integrate (merge) branch into integration branch
  4. Build package based on integration branch
  5. Upload

Package maintenance: feature & integration branches

- Scales very well
- Good merging / conflict resolution
- Clean separation of features
- Steep learning curve

Package maintenance: summary

- Every method has pros and cons (surprise!)
- dpatch is a very tangible approach which should be preferred over the single diff approach; suitable for simple packages with few maintainers
- branching is very flexible and might be the best choice for complex packages, or packages with many maintainers

Package management and maintenance combined

... can we somehow condense all pros into a new method and leave the cons behind?

My idea for package management: In-Deb VCS

- Store decentralised VCS data in the Debian source package
- Provide intuitive frontend tools (think dpkg-buildpackage)
- Provide a logical integration with the BTS
- Harness the VCS to facilitate and improve the upload process

In-Deb VCS and Launchpad/hct

- Canonical has Launchpad and hct, which have the same roots as In-Deb VCS
- Launchpad offers workflow components
- hct will become obligatory for Ubuntu
- Centralised, and not to be adopted by Debian
- But the intention is to coordinate and exchange with Canonical to keep the two approaches aligned

In-Deb VCS: differences from hct

- Uses mirror infrastructure, not centralised
- Integration with source package format
- Tighter integration with Debian methods:
  - tracking in BTS, not Launchpad
  - dak, not Launchpad
- In-Deb VCS is optional, compatible with other approaches

In-Deb VCS: introduction procedure

- Figure out how developers would like to work
  - Interviews, questionaires
  - Live sessions
- Make sure not to force its use: regular methods **must** continue to work
- Code, document, get it working
- Campaign and let it (not me) persuade people

Introducing Bazaar-NG

- A VCS developed by Canonical
- Decentralised approach
- Small, intuitive command set
- Python bindings
- Active, enthusiastic community, who are interested to support this use-case

In-Deb VCS: specifics

- .bzr/ directory stored in extra tarball and distributed across mirrors
- Consider integration with dpkg-source
- Address size concerns:
  - Can .bzr replace diff.gz and debian.tar.gz?
  - Size-efficient storage (weaves, knits)
    Investigate using .orig as weave base
  - Harness to-be-developed possibility to ghost revisions to external storage

In-Deb VCS: advantages

- Decentralised, mirrored repository
- Uses existing infrastructure, no changes needed beyond Wig & Pen possibilities
- All advantages of the VCS
- (bzr developers are mostly Debian people)

In-Deb VCS: challenges

- Slight size increase, if we get it right (mirrors)
- Dependency on bzr, Python
  bzr/Python may need to be added to build-essential
- ... probably many more

Upload requests

- Do not upload source + 1 binary package
- Instead, upload a request made up of
  - Location of integration branch
  - Certificates issued by lintian, piuparts, etc., signed by developer who ran them
  - One or more developer signatures signing off the upload
- Components required for a request can be set on a per-package level

Upload requests: advantages

- Formalised integration of quality checks
- Less payload to be transfered to ftp-master
- All architectures built by Debian buildds
- Smooth transition possible:
  - provide a server receiving upload requests
  - checks certificates and signatures
  - obtains package, builds it (probably i386)
  - uploads to ftp-master

Summary

- Jane Doe should have an easier time contributing
- How to get people to adopt new methods
- Ideas for improved workflow
  - Integrating stable and testing security
  - Combining the cream of package management and maintenance approaches
  - Upload requests instead of package uploads

Thank you for your attention!

Martin F. Krafft <madduck@debian.org>

http://martin-krafft.net/phd *(wip)*
http://people.debian.org/~madduck
http://debiansystem.info

Copyright & Licence