Patterns and anti-patterns in Samba Development

Presented by Andrew Bartlett
Samba Team - Catalyst // June 2018

catalyst
open source technologists

SAMBA TEAM

Samba is a member project of the
Software Freedom Conservancy

software freedom
conservancy

Andrew Bartlett and Catalyst's Samba Team

- Samba Developer since 2001
- Based in Wellington, New Zealand
- Team lead for the Catalyst Samba Team
- These views are mine alone
  - I'm a little passionate about this stuff

- **Garming Sam**
- **Douglas Bagnall**
- **Gary Lockyer**
- Tim Beale
- Joe Guo
- Aarron Haslett
- Jamie McClymont

A talk about what we do well

- Samba is a successful, multi-decade software project
- Successful pivot from ‘just a file server’ into also being a full AD DC
  - Successful merge of Samba 3.x with the Samba4 fork (to be blunt)
- International collaborative development
  - Cross-cultural

And about what we don't do well

- Contributor and contribution counts static
  - 2008-2011 was a boom time per openhub.net data
- Lost the student contributor class
- Very few hobbyist contributors
- Most new contributors have been employees of Vendors
  - Catalyst team has grown, thanks to our commercial clients

Successfully avoided becoming a ‘.com’ project

- Samba has remained independent
  - Never bought out
  - Never sold out
- Nobody made their fortune
  - But provided gainful employment for many
  - Launched many software engineering careers
- Passionately an independent Free Software project since 1992!

But we also have no roadmap

- The poorly updated wiki page doesn't count
- We meet here to speak about our work, but rarely our direction
- Samba's pattern is not to mention a proposed direction until we have already taken it
  - Great for not setting up false expectations
  - Or being asked to paint someone else's bikeshed
- Hard for collaborative planning
  - Our patches are small changes that do 'nothing'
  - But no clear overall design

Pretty good community behaviour

- We pattern good behaviour and generally see it on our lists

But rely on ‘know it when you see it’ for enforcement

- No written code of conduct
- No contact point for concerns
- Relies on Jeremy seeing it and saying that it crossed the line
- Could be difficult working out how to respond without a written plan

Comprehensive tests

- We have thousands of tests
- By convention:
  - All new features have a full testsuite
  - Bugs have a testsuite to prove they fixed the bug

But flapping tests

- Enough to drive anyone flapping mad!
- Specifically for me currently:
  - raw.notify
  - dgram.netlogon
- A genuine caution against marking tests as flapping (ignored)
  - Real bugs behind many flapping tests

CI: “Not rocket science rule of Software Engineering”

- *automatically maintain a repository of code that always passes all the tests*
- Attributed via marge-bot to Graydon Hoare
  - main author of Rust
- Well before Rust even started, Samba was doing that
- Pre-commit CI on the gate (autobuild)
  - No merges
  - Only rebase
  - Re-start tests on each new base

But not available outside the team

- Non-team members have no access to sn-devel
  - The only host that really counts
- And no easy access to a substitute
  - Make test takes 4-5 hours and is highly host-dependent
- Once reviewed, team members often reply with ‘sorry, failed autobuild’
  - And the cycle starts again

Clear coding style guidelines

- README.Coding clearly specifies how our code should look
- C99, mostly
- Linux style, mostly

But much of our code still pre-dates it

- So the pressure of consistency is against the rules!
- New and changed lines in old functions follow the rules
  - But old code is untouched
- Don't change formatting and code at the same time
  - Because it makes it hard to see what actually changed
- Fix formatting or make easier to review?
  - What about global search/replace?
- And even re-indent patches are avoided

Successfully implemented code review

- Two-engineer review has been a requirement for years now
- Code review regularly catches important issues
- Now an unquestioned and systemic part of our development practice

But the workload is quite un-even

- 1059 Andrew Bartlett
- 678 Jeremy Allison
- 484 Andreas Schneider
- 415 Garming Sam
- 386 Ralph Boehme
- 338 Amitay Isaacs
- 329 Martin Schwenke
- 303 Douglas Bagnall
- 283 Stefan Metzmacher
- 160 Volker Lendecke
- 71 Alexander Bokovoy
- 66 Ralph Böhme
- 54 Richard Sharpe
- 51 Gary Lockyer
- 34 David Disseldorp
- 26 Guenther Deschner
- 25 Christof Schmitt
- 18 Uri Simchoni
- 15 Björn Jacke

And the implementation is mixed

- Many Samba Developers give great, detailed reviews
  - I thank the reviewers
  - At Catalyst, Upstream code review is ‘on the clock’, so I thank the Directors
- However some contributors have a cloak of invisibility
- Some reviewers only review for white-space
- No early upfront design stage review
  - Reviewers often have to reverse engineer it from a massive patch set

Many reviews are quick and responsive

- Particularly when good trust is established between developers
- Many simple patches reviewed and accepted within hours

Some contributors have a hard time getting attention

- Let alone two reviews for external contributors
- Certainly not the reactive feedback they need if we want to win them as new contributors
- Particularly challenging for Python code
  - The easiest to contribute to but with the least willing reviewers
- For reviewers, hard to commit to a review and push of unknown code
  - A lot of time can be wasted on patches that don't pass test

Solving social problems with engineering solutions

- This is clearly what the team is best at
- Whitespace issues
  - Git commit hook
- Not running make test
  - Autobuild

Proposed new engineering solution: Gitlab

- This won't solve all our problems
  - But isn't GitHub, so that is a start
- Possibility to use merge requests to describe overall goals
  - E-mail integration allowing reply-by-mail
- GitLab is being adopted by Debian and Gnome

https://gitlab.com/catalyst-samba/samba

- Catalyst Staff repo on gitlab.com
  - Used for day to day development
  - Full CI run in the Catalyst Cloud and shared runners
  - master branch mirrored in
- Been in use for a couple of months as a prototype for broader Samba team use
- Merge requests only lightly tested
  - Git branches and CI links posted to samba-technical manually

https://gitlab.com/samba-team/samba

- I've set up an offical Samba Team mirror on gitlab.com
  - Fork this for a private development repo
- Also a collaborative development repo here:
  - https://gitlab.com/samba-team/devel/samba
  - Invited members can run the full CI here
    - (see next slide)
  - Official branches mirrored

Rackspace hosted CI

- Gitlab.com shared runners (CI) can't run our full testsuite
- A gitlab CI runner is hosted at Rackspace
  - Described by an ansible playbook
  - Dynamically deploys 4 CPU, 8GB ram machines
  - Runs the remaining not-yet-split-up tests (4h30m)
- First $2000 costs each month provided by Rackspace
  - Covers about 2000 CI runs

Could we become a GitLab project?

- Many advantages to being mailing list based
- However we may miss the ‘GitHub generation’
- We might sneer
  - Can the ‘GitHub kids’ really code to our level?
  - But new developers need to find us and make a first contribution
  - Joining a mailing list to make a first contribution was more reasonable in early 2000s
- GitLab and GitHub used in university courses now

What could it look like?

- We should keep sn-devel for now
  - But accept, autobuild and then close requests like we do GitLab pull requests
- Prefer not to do another notification bot
  - I would prefer all developers joined the project and got direct notifications
  - This allows reply-by-email
- Consider something like ‘marge-bot’ in the long term
  - Manages an autobuild queue automatically based on review tags and CI results

What issues would it address

- Merge requests could contain an overall design
  - A few paragraphs on the ‘what and why’
  - Stays with the patch set for life of the series
- Give non-team members access to CI
  - Contributors and reviewers find out they if have broken tests up front
  - Potentially pure more style checks into the CI
- Track outstanding patches and assigned reviewers
- Easier first contributions

On a lighter note

- Caution light trolling ahead

Above all: git patches as performance art

- The most important thing is solid, tested code

Above all: git patches as performance art

- The most important thing is clear code
  - The most important thing is solid, tested code

Above all: git patches as performance art

- The most important thing is following README.Coding
  - The most important thing is clear code
    - The most important thing is solid, tested code

Above all: git patches as performance art

- The most important thing is a ‘clear’ set of patches without unrelated changes
  - The most important thing is following README.Coding
    - The most important thing is clear code
      - The most important thing is solid, tested code

Above all: git patches as performance art

- The most important thing is the 80 column rule
  - The most important thing is a ‘clear’ set of patches without unrelated changes
    - The most important thing is following README.Coding
      - The most important thing is clear code
        - The most important thing is solid, tested code

Above all: git patches as performance art

- The most important thing is the order of patches
  - The most important thing is the 80 column rule
    - The most important thing is a ‘clear’ set of patches without unrelated changes
      - The most important thing is following README.Coding
        - The most important thing is clear code
          - The most important thing is solid, tested code

Above all: git patches as performance art

- The most important thing is whitespace
  - The most important thing is the order of the patches
    - The most important thing is the 80 column rule
      - The most important thing is a ‘clear’ set of patches without unrelated changes
        - The most important thing is following README.Coding
          - The most important thing is clear code
            - The most important thing is solid, tested code