Fostering New Contributors

Lessons in Community Management from SWT
Eric Williams

ericwill@redhat.com
@yyzericwill

- Red Hat Eclipse team member in the Toronto office
- Full time SWT developer and committer
- Intern at Red Hat from May 2015 - August 2016
- After my internship I was hired back as a full time software developer work on SWT
Alexander Kurtakov

akurtako@redhat.com
@akurtakov

- Red Hat Eclipse technical leader
- Eclipse and Tools Top level projects PMC member
- Platform, Linux Tools, CDT, DLTK .... committer
What is community management?

- Open source software projects are made up of code, contributed by a wide variety of contributors.

- Community management handles the logistics of getting the code.

- This means enabling development, and attracting new contributors.
How is this accomplished?

- Community management entails:
  - Marketing
  - Documentation
  - Community interaction (issue trackers, mailing lists, chat rooms, etc.)
  - Smart design decisions
  - Identifying and resolving bottlenecks in code contribution

- Ensures that code can flow freely from contributors into the project with minimal barriers
Why is this important?

- People often think about the content (i.e. the code)
- Not everyone thinks about “how to get the code” or where it comes from
- A strong community defines the pipeline of incoming code
- New contributors are paramount
  - No new contributors = the project slowly dies
Path required for code to be merged

Code

Contributor
No one will want to drive on your project’s “road” if it is full of obstacles

(especially if they are doing it for free)
Goals of community management

1. Users involved with bugs/mailing lists are likely potential contributors
   ○ Creating a welcoming project increases the chance that users will contribute

2. New contributors mean a smaller workload, as work is distributed amongst members of community

3. Everybody benefits
What lessons did SWT teach us?
Bugzilla
Triaging incoming tickets

- Triage new bug reports in a timely manner
  - 24 - 48 hours maximum

- Users who file bugs are likely already motivated to submit fixes

- Letting new tickets go unanswered does not encourage contribution
Triaging lessons learned

- Use keywords to group similar bugs

- Extract relevant information early on the in bug report to speed up time required for a fix, and establish parameters

- Being attentive to incoming tickets gives a good impression to the user, speeds up development, and keeps the project lively
Manage existing bugs

- These are tickets that have been filed within the last 2 years, and are likely still relevant

- Check in with users after major milestones to see if the issue is still present

- Offer to review patches for issues that are low priority to you
  - This might result in someone contributing a patch where they might not have before
Older tickets

- When I started as an intern in 2015, SWT-GTK had over 2500 open tickets
  - Many of these date from 2005 or earlier
  - Today the ticket count stands at less than 400

- These cause clutter and reduce the effectiveness of issue tracking

- Too many old tickets makes it difficult to navigate bugzilla, and prioritize development efforts
SWT-GTK bugs triaged: 2015 - present
SWT-GTK bugs fixed: 2015 - present
Cleaning up the backlog

- Dedicate time each week to close some tickets that are clearly no longer relevant
- Ping older tickets asking about relevance
- Keeping a clean, relevant bugzilla makes it easier for new contributors to find issues they experience, or things they want to fix
General tips

- Avoid use of overly complicated terminology, and unnecessary jargon
- This confuses users, and discourages potential contributions
- Always be polite
  - No condescension towards beginners
  - Light humour helps ease frustration
  - Prompt responses show a respect for time
Mailing lists
The front line

- Many lists are the frontlines of communication for developers, and potential contributors

- Avoid flame wars
  - Always be polite
  - Diffuse/avoid heated debates with frustrated users

- Avoid detailed technical discussions, move these to bugzilla
As an existing member of the community

- Avoid detailed technical discussions, move these to bugzilla

- Keep the number of lists to a minimum
  - Too many lists to choose from leads to confused contributors, and/or dead lists

- Keep track of relevant upstream mailing lists related to your product
Patch reviews
Timeline

- Do not let patches sit for more than 24 - 48 hours

- If you are particularly busy, ask another committer to review it instead

- Provide time feedback to the submitter
  - Even a comment such as “I’ll review this tomorrow” is better than silence
  - Inaction drives away new contributors
Reviewing the patch

- Do not be overly harsh
  - Keep firm code standards, but be polite
  - New contributors are driven away if their patch is torn to shreds upon first submission

- Forgo minor nitpicking (like grammar) if the code is good

- Have resources available
  - It is very frustrating to have code rejected due to seemingly arbitrary standards
After accepting a patch

• Say thank you!
  ○ Small encouragements build rapport, and encourage future contributions

• Reward high quality, repeat contributors with a committership if there is interest

• Offer to include frequent contributors in design meetings/decisions
Documentation
UP TO DATE DOCUMENTATION

NOW THERE IS SOMETHING I HAVEN'T SEEN IN A LONG TIME.
Wiki/Articles

- Keep it up to date!

- It does not reflect well on a project when the main documentation is still 10+ years old

- Keep articles concise, store them in a central location

- Have a beginners guide that new contributors can be pointed to
Code documentation

- Short, concise comments
  - Avoid excessively long Javadocs -- these can really clog up code

- Reference bug numbers in comments to add context

- Write descriptive commit messages, do not leave them empty
  - Summarize the issue, and explain how the patch fixes it, what was tested, etc.
Design Decisions
Prioritizing work

- When faced with a low number of contributors, work must be prioritized
  - Fixing GTK3 usability issues vs. GTK2 bug fixes
  - Supporting latest GTK3 versions instead of older, deprecated ones

- Consider which bugs need more attention
  - Some components might have blockers that completely break their workflow

- Consider other user’s needs when deciding which bugs to fix over others
High level decisions

- Long term design decisions affect contributor interest, and retention
- It will be harder to find contributors interested in supporting old legacy software
- Staying up to date with underlying dependencies/frameworks keeps the project competitive, thus retaining users
Removing Hurdles
Building and Testing

- Simplify any build/CI systems
  - Reduce waiting time for contributors to see results of builds/tests

- Maintain automated tests
  - Write meaningful yet concise test cases
  - Avoid duplication

- Use widely used/standard tools like maven
Dependency Management

- Keep dependency chain small, contributors should be able to test easily without needing to install lots of dependencies
  - This is especially true for tests: testing code should not require installing excessive amounts of libraries/frameworks

- Modernize dependencies
  - No one enjoys hunting for ancient X libraries
  - Consider porting any old scripts to modern languages like Python, instead of Perl or Bash
Examples from SWT

- 32 to 64 bit
  - Switched default to 64 bit to not burden contributors (64 bit) with extra conversions

- SWT tools - multiple iterations to make it as easily installable as possible

- Unit tests
  - Run with tycho
  - Reorganize
  - Modernize
Results

- 4 new contributors in the last 16 months, compared to none the years previous
  - Simeon Andreev - Linux/GTK
  - Nikita Nemkin - Windows
  - Rolf Theunissen - GTK3 on Windows
  - Conrad Groth - Windows

- Many of these contributors contributed new code on the condition that certain things be modernized
  - For example, Nikita pushed for Windows XP removal to simplify the code base
THANK YOU

plus.google.com/+RedHat
linkedin.com/company/red-hat
youtube.com/user/RedHatVideos
facebook.com/redhatinc
twitter.com/redhat
twitter.com/eclipsejavaide