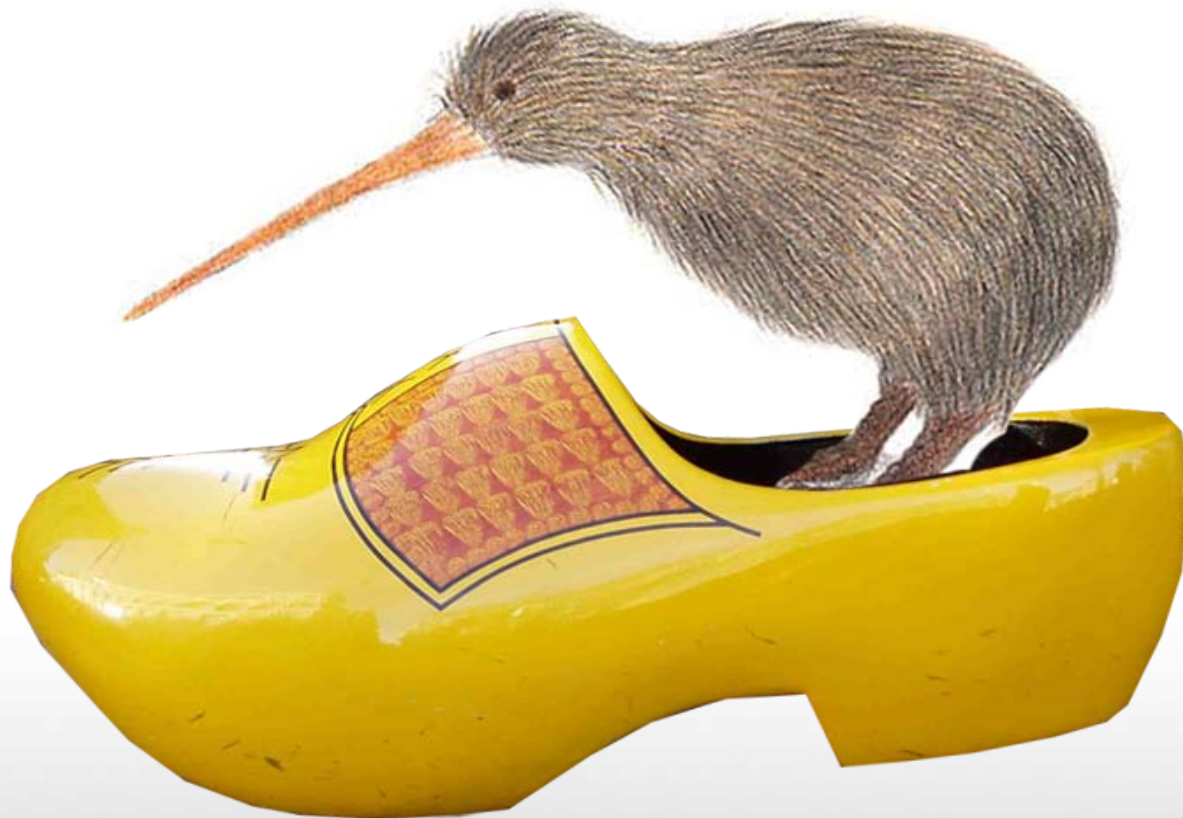


How to open source internal software to benefit your
organisation

Martijn Verburg (@karianna, @java7developer)

Introduction – Speaker



Introduction – Speaker



- I'm a Java and open source software advocate.
- <http://www.ikasan.org> (Ikasan EIP) and <http://pcgen.sf.net> (PCGen).
- Who thinks that the power of technical/user communities are awesome.
 - Like this one! [Coderanch](#), JUGs etc...
- Who is a little bit daft.
 - Writing “[The Well-Grounded Java Developer](#)” with Ben Evans.
- Who loves questions and feedback (especially the honest kind).
 - Email: martijnverburg@gmail.com
 - Blog: <http://martijnverburg.blogspot.com> - Are we there yet?
 - Twitter: [@karianna](#), [@java7developer](#)



Introduction – There be Dragons



We are not going to get through all of this!



<http://bit.ly/hzmRHu>





This talk usually
produces more questions
than answers

Sorry about that





So, what's your open source software story?

Waddya wanna know?



Introduction – Areas to cover



1. Getting Started Part I
 - Benefits of open source and picking a project.
2. Convincing the Stakeholders
3. Getting Started Part II
 - The important bits of that long list below
4. Technical Infrastructure
5. Social and Political infrastructure
6. Money
7. Communications
8. Packaging, Releasing and Daily Development
9. Managing Volunteers
10. Licensing, Patents and Copyright



Getting Started Part I - Benefits of open source



- **Business**
 - Lower costs
 - Increased resources
 - Public relations
- **Technical**
 - ‘10,000 eyes’ on the technical content
 - Explaining/Documenting for outsiders improves quality
 - More users == more rigorous testing!
- **Social**
 - Participation in a vibrant community
 - Working with new cultures/ideas



Getting Started Part I – Choosing an appropriate project

- Key indicators that can help you identify the right project:
 - Plumbing
 - Not a core competency
 - No intrinsic IP value
- Ikasan EIP – The Middleware platform of Mizuho International Bank
 - Middleware is plumbing
 - Software development is not a core competency of the bank
 - Software had no IP value (that the bank wanted to trade on)



Convincing the Stakeholders – Who are they?



<http://bit.ly/dWf4KU>



Convincing the Stakeholders – Legal



- Legal: Questions
 1. What license to use?
 2. What liability do we have?
 3. Uh oh, we don't even have a policy on using open source, what now?

- Legal: Answers
 1. That depends! There are plenty of business/corporate friendly licenses.
 - There is legal precedence in this area
 2. That depends on the 'law of the land'
 - Typically none, unless it's overruled by common law
 3. Then you are vulnerable to lawsuits
 - Get a policy in place immediately!



Convincing the Stakeholders – Information Security



- Information Security: Questions
 1. Will our data be in the public domain?
 2. Are any of our systems made vulnerable?
 3. How is authentication/authorisation applied?
 4. Open source must surely mean less secure?
- Information Security: Answers
 1. Not unless your staff upload sensitive information (same as now)
 2. Not if you host the project externally as recommended
 3. According to project policy
 - Typically everything is reviewed before being committed
 4. Fallacy
 - Think Linux vs. MS Windows
 - “Security through obscurity is not security at all”



Convincing the Stakeholders – Infrastructure



- Infrastructure: Questions
 1. What access do internal staff require?
- Infrastructure: Answers
 1. Typically they need secure access to external hosting providers, e.g.:
 - HTTPS
 - SCP
 - SSH
 - Plus HTTP access to some social/communication sites



Convincing the Stakeholders – IT Management



- Head of IT: Questions
 1. Does it affect productivity?
 2. Does this mean free help?
 3. What's the cost?
 4. Quality Control?

- Head of IT: Answers
 1. Initially yes! But then community takes over
 2. Yes! But you get out what you put in
 3. Legal is often the biggest up front cost
 - On the IT side, it can be set-up for <math><£1000 + X \text{ person hours}</math>
 4. Is typically tighter due to public scrutiny
 - If project is run in recommended way



Convincing the Stakeholders – The business



- Business: Questions
 1. What's in it for us?
 2. Won't we lose control of what we need?
- Business: Answers
 1. Faster delivery at a higher quality
 - Due to more users
 - Due to more technical eyes looking for issues
 2. Product can still be easily controlled by bank staff
 - Can be forked if necessary



Convincing the Stakeholders – The development team

- Development Team: Questions
 1. But outsiders wouldn't understand our code?
 2. We're worried about our existing quality?
 3. How do we run all of this?
- Development Team: Answers
 1. Then you need to improve the code/documentation
 2. You're no better or worse than the other 200,000+ open source projects, don't worry
 3. That's what the other 40+ slides will cover!
 - But seriously, read Karl Fogel's seminal work on this topic producingoss.com



Getting Started Part II – Project name and Mission Statement



- Project name should be:
 - Catchy and/or relative to the project
 - Easy to remember
 - Doesn't impose on trademarks / is unique
 - Has major domains available (.com, .net, .org)
- Mission Statement should:
 - Define the major goal of your project
 - Be concrete, limiting and short!



Getting Started Part II – 60 seconds to reel them In



<http://bit.ly/esPcYZ>



Getting Started Part II – 60 seconds to reel them In

- **Declare FOSS Status**
 - Make it abundantly clear that your project is Free Software and/or Open Source
 - Choose your license carefully!
- **Features and Requirements Listing**
 - Clarifies your mission statement scope
 - Allow users to understand immediately what they need to run your software
- **Development Status**
 - Future Roadmap and Milestones
 - What stage is your project at (Alpha, Beta, Mature)?



Getting Started Part II – Make it available



- Downloads
 - Both Source and binary downloads should be available
- Version Control
 - Vital to have source control in place early on
 - Anonymous access a must
- Issue Tracker
 - A place to record issues is a mandatory requirement
- Canned Hosting
 - SourceForge
 - Google Code



Getting Started Part II – Make it accessible



<http://bit.ly/dWjTj7>



Getting Started Part II – Make it accessible



- Public Communications Channel
 - A Mailing List/Forum should be opened immediately
- Developer Guidelines
 - How to communicate with other developers
 - How to provide patches
 - How development is done (Benevolent Dictatorship, democracy etc)
 - Practice conspicuous code reviews and make them public



Getting Started Part II – Documentation



- Biggest complaint about most open source projects is poor documentation
- Clear “How To” for installation, otherwise people will give up
- Have a FAQ and “Common Task” tutorials
- Label the areas where the documentation is known to be incomplete
- Should be available online and offline
- Screenshots and sample output, a picture tells a thousand words



Getting Started Part II – Setting the tone



- Open source projects are very social, failures tend to occur due to social rather than technical reasons
- Avoid private discussions! Make it a project policy
- Nip rudeness in the bud
- Kathy Sierra's "[Creating Passionate Users](#)"
 - Javaranch's "Be Nice" policy
- Be sensitive if opening a formerly closed project



Technical Infrastructure – Mailing Lists/Forums



- Choose one that the community prefers
- Mailing list manager (Mailman, Ezmlm, Y! groups, Google groups etc)
- To avoid SPAM:
 - Only allow posts from subscribers and moderate their first message
 - Run messages through a SPAM scrubber
- The great “reply all” debate
 - <http://www.unicom.com/pw/reply-to-harmful.html>
 - <http://www.metasystema.net/essays/reply-to.mhtml>
- Archive!



Technical Infrastructure – Version Control



- Vital for a distributed group
- Choose a popular/easy to use one
- Version everything
- Make it publicly browsable
- Enforce commit comments, preferably with a issue number
- Don't be afraid of branches, don't be afraid to cut them off either



Technical Infrastructure – Issue Tracking



- Far, far more important than most people realise
- Choose one that is easy for users to use, encourage good reporting
- Choose a lifecycle and have the meta data clearly organised
- Good systems can automatically show milestones and roadmaps
- Triage!
- Be polite with user reports



Technical Infrastructure – IRC and social networking



- IRC
 - Great for instant communication
 - Start with one channel
 - Discuss archiving policy
 - Encourage posting of large data to 'paste' sites, e.g. <http://pastebin.ca>
- Social Networking (Facebook, Twitter, RSS etc)
 - Can be a good idea to set these up
 - Don't over use them / flood them with messages
 - Best for one way communication



Technical Infrastructure – Website and Wiki



- Website
 - Gateway for your project
 - Make it professional!
- Wiki
 - Very important for community generated knowledge
 - Set editorial standards early, especially navigation and tagging
 - Use common wiki software, not something obscure
 - Design discussions and architecture documents can be centralised here



Social And Political Infrastructure – Structure



- Benevolent Dictator
 - Really more of a 'community-approved arbitrator' or 'judge'
- Council/Board of Directors
 - As projects mature and grow, boards or councils tend to form
 - Most decisions are consensus (polls), with formal voting if required
 - Decision needs to be made on who votes
 - Sometimes allow vetoes



Money – Types of involvement



- Sharing costs, e.g. <http://koha.org>
- Software is used to support hardware sales
- Software is used to undermine a competitor
- Dual licensing, e.g. Oracle's MySQL
- Donations



Money – Don't be a faceless company



- Hire team members for the long term
- Appear as individuals, not a single corporate entity
- Be open about your motivations
- Money can't buy you love, treat project members as equal



Money – Funding non-programming activities



- Quality Assurance (i.e. Professional Testing)
- Legal Advice and Protection
- Documentation and Usability enhancements
- Providing Hosting/Bandwidth
- Marketing and PR
 - Remember that you are being watched
 - Don't bash competing products



Communications – You are what you write



- You are judged on your communication
 - Good communicators achieve more than good programmers
- Structure And Formatting
 - Come up with a convention for the project
 - Plain text, 80 character wide emails are preferred
 - Spelling and Grammar are important, don't neglect them!



Communications – Content and tone



- Remember that there are usually many more readers than writers
- Avoid hyperbole
- Over time terseness will creep in, this is OK
 - As long as it remains polite!
- Edit before you send
- Don't assume English is their first language



Communications – Recognising rudeness



- When you respond, try to respond properly and with full effort
- Remember there are no visual cues
- Try to use real names
- Trim replies and disclaimers
- Rude people waste other people's valuable time
 - Be wary of the vocal minority



Communications – Soft topics last longer



- It's always the non-technical discussions that last the longest
- Don't waste time on the "Bike Shed"
- Avoid Holy wars, especially over:
 - Programming languages
 - Licenses
 - Reply-to munging



Communications – Handling growth



- Number of Inexperienced users rises rapidly
- Number of experienced users rise much more slowly
- Good publicly available documentation is vital
- Produce specialised forums/lists ([Javaranch](#))
- Make archives available
- Keep conversations out of the issue tracker



Communications – Publicity



- You Website front page is seen by more people than any other part of the project
 - Important news should be posted there
- Also have a "News" or "Press Releases" area of the web site
- If your project has an RSS feed utilise that as well
- If the announcement is about a new release of the software, then update your project's entry on <http://www.freshmeat.net/>



Packaging, Releasing and Daily Development – Release Numbering



- Serves as a guide to end users whether to take a new release
- major.minor.trivial designation often used
 - Have a clear definition of what bumps a major, minor, trivial release
 - Major breaks backwards compatibility
- Use Alpha, Beta and RC as appropriate
 - Google's idea of Beta should not be followed!



Packaging, Releasing and Daily Development – Release Branches



- A Branch should be created for each Major/Minor release
 - e.g. 1.0.x and 1.1.x
- Different strategies on how to release:
 - Release Owner - Trusted team member empowered to make the call
 - Vote changes in - Can get endless
- Generally best to have a rough roadmap first
- Normal to fix vital bugs in several lines & release simultaneously
 - Decide when you want to 'End Of Life' a line



Packaging, Releasing and Daily Development – Packaging



- Standards are `tar/zip` for Linux/UNIX and `zip` for windows
- Should include Files:
 - README
 - INSTALL
 - LICENSE
 - CHANGES
 - MEMBERS



Packaging, Releasing and Daily Development – Packaging



- Follow consistent way for users to compile and install the software
 - e.g. (Java) `mvn clean install`
 - e.g. (C++)
`$./configure`
`$ make`
`# make install`
- Make std binaries e.g. `msi/exe` for windows and appropriate packages for various Linux/UNIX distros
- Use MD5/SHA signing so people know that packages have not been tampered with
- Make Betas, RCs readily available and make it easy to test side by side with a stable release



Packaging, Releasing and Daily Development – Daily Development



- Encourage discrete check-ins, so they can be easily rolled back
 - e.g. Don't mix several items in one check-in
- Enforce commit comments
 - Try to enforce traceability to an issue number
- Enforce Unit Tests/Integration Tests with commits
 - It's important to avoid regressions
 - Continuous Integration is vital



Managing Volunteers – Getting the most out of volunteers



- Volunteers often start with the project due to a technical reason
 - e.g. Fixing a minor bug
- Will stay for many, many different reasons
- You need to ascertain what makes each individual tick
 - Helps you pick up on disruptive members early



Managing Volunteers – Delegation



- Is a public declaration of trust
- Draws people further into the project
- Need to give people a graceful out
- Differentiate asking someone to investigate something vs. asking them to take ownership
 - Don't blindly assign!
- Follow up after you delegate!
- Notice which people are interested in a particular area



Managing Volunteers – Praise and Criticism



- Two sides of the same 'attention' coin
- Praise is often the only payment volunteers get
 - Use it wisely, don't undervalue it
- Criticism must be delivered dispassionately with detail
 - "It was sloppy."
 - vs.
 - "Could you review that and apply std project guideline X please?"



Managing Volunteers – Prevent Territoriality



- Single point of failure.
- Diminishes community.
- Common to avoid author tags in code, preference to use "X Team".



Managing Volunteers – Every User Is A Potential Volunteer



- Always maximise contact with a new user who has reported something.
- Try to get user involved in the fix.
- Remember to be patient with new volunteers, educate them!



Managing Volunteers – Share management tasks as well as technical tasks



- Often management tasks are as time consuming as technical ones
 - e.g. Board of Directors on PCGen consist of management 'Team Leads'.
- Common roles are:
 - Patch Manager
 - Release Manager
 - Translation Manager
 - Documentation/FAQ Manager
 - Issue Manager



Managing Volunteers – Transitions



- Volunteers won't stay in same role forever
 - RL == 'Real Life' often gets in the way!
- Time management is a vital skill
 - Encourage new volunteers to start with a small commitment
 - Monitor signs of tiredness
 - Don't expect unpaid volunteers to “Do it right now”
- Very occasionally a volunteer is inappropriate for a position
 - Get private project consensus before privately discussing a transition to a new role



Managing Volunteers – Committers



- Committers are seen as the core of a project team
 - They typically form quality control
 - They typically have a 'vote'
 - Choose them on their judgement more than anything else!
- Can use partial commit access to entice people in
- Avoid mystery committers
 - They should be accessible to the rest of the team



Managing Volunteers – Forks



- Try to avoid forks where possible
 - Forks aren't necessarily a bad thing
- You should act dispassionately and not engage in blocking behaviour
- Almost all forks fail and/or merge back in with the original project



Licenses, Copyrights and Patents – Terminology

- “Free Software” (<http://www.fsf.org/>) - Tends to be used by those who have the philosophical belief
- “Open Source” (<http://www.opensource.org/>) - Tends to be used by those without
- List of licenses maintained at <http://www.gnu.org/licenses/license-list.html>



Licenses, Copyrights and Patents – Aspects of licenses



- Compatibility with proprietary licenses
 - Apache, MIT & Others
- Compatibility with other free licenses
 - Several licenses automatically are compatible with each other
 - GPL can be tricky on this aspect
- Enforcement of crediting
 - Whether you must provide notice or not
- Protection of trademark
 - Must have written permission to derive from original



Licenses, Copyrights and Patents – License choice



- MIT/X Window style license is recommended for a “Do with this what you want” approach
- GPL style license is recommended for a “Do with this what you want except put on restrictions“ approach
 - The LGPL loosens some of those restrictions
- Apache license is seen as business friendly
- Apply license text/snippet to your source code, most licenses have a template for this



Licenses, Copyrights and Patents – Copyright assignment and ownership



- Dealing with contributors copyright and ownership:
 - Do nothing (Not recommended)
 - Contributor License Agreements - Electronic form is sufficient
 - Transfer of Copyright - Total transfer of copyright
 - <http://www.openrightsgroup.org>
- Dual Licensing
 - Gaining popularity with corporations
 - Typically allows 'Free' license to non profits, academics and other open source users
 - Can be difficult to manage community with dual licensing



Licenses, Copyrights and Patents – Patents



- Very harmful to software development
- Often owned by corporations with deep pockets
 - Open Source project is forced to stop
- You can defensively patent to make sure that your project doesn't get blocked



Shameless Plug (java7developer.com)

