Freie Universität Berlin

Course "Softwareprozesse"

Open Source Software Development

Lutz Prechelt
Freie Universität Berlin, Institut für Informatik
http://www.inf.fu-berlin.de/inst/ag-se/

- Definition
- Process characteristics, strengths
- Economical incentives
- Project types, leadership and decision-making
  - The OSS career
- OSS license types
- OSS tools
- Innovation management in OSS

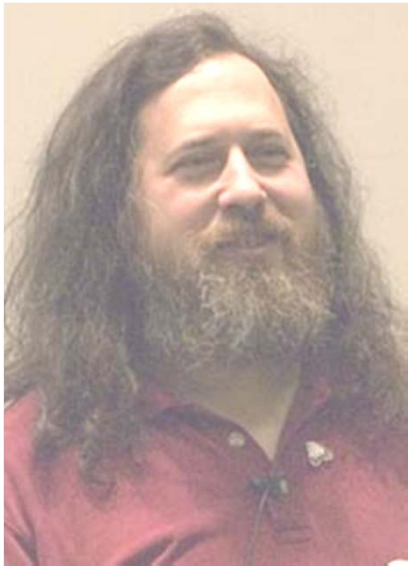
Learning objectives

- Understand the definition of Free/Open Source SW
- Understand typical process characteristics:
  - participants, process, productivity, quality
  - leadership and decision-making, participant career
  - economical incentives for participation
  - tools used
- Understand the various OSS licenses
- Understand the notion of innovation management and how it applies to OSS

Definition "Free Software"

Freie Universität Berlin

Richard Stallman, Free Software Foundation (FSF):
http://www.gnu.org/philosophy/free-sw.html



- The freedom to run the program, for any purpose (freedom 0)
- The freedom to study how the program works, and adapt it to your needs (freedom 1).
  - This requires access to the source code.
- The freedom to redistribute copies so you can help your neighbor (freedom 2).
- The freedom to modify the program, and release your improvements to the public, so that the whole community benefits (freedom 3).

On Richard Stallman, see

- http://www.faifzilla.org/ and http://www.catb.org/~esr/writings/rms-bio.html

Definition "Free and Open Source Software" (2)

- Stallman calls such software "Free Software"
  - he promotes it actively since 1985
  - http://www.fsf.org/
- Today, the more common term is "Open Source Software"
  - abbreviated OSS
  - This move was initiated in 1998 by Eric Raymond:
    - because the previous term made "a lot of corporate types nervous"
    - http://www.catb.org/~esr/open-source.html
- Sometimes also termed "Free/Libre Open Source Software"
  - abbreviated FLOSS or shortened to FOSS or F/OSS
- Free SW now has two "home organizations": FSF and OSI, the Open Source Initiative
  - http://opensource.org/

Contrasts: proprietary, shared source, closed source

Freie Universität Berlin

- Most company software is *proprietary* ("eigen", "geschützt"): The copyright holder reserves the right to use the software
  - either to himself (custom SW)
  - or to people who accept restrictions regarding the use of the SW and usually pay a license fee (commercial SW products)
- Usually (but not always) proprietary SW is *closed-source*
  - meaning even the allowed users only get to see a binary version
- If not, it is often called "shared source"
  - Different models possible:
    - You may view source code or
    - You may also modify it or
    - You may also give modified versions to people having a license for the original version
  - Instances exist from Microsoft, Hewlett-Packard, etc.
    - e.g. http://microsoft.com/sharedsource

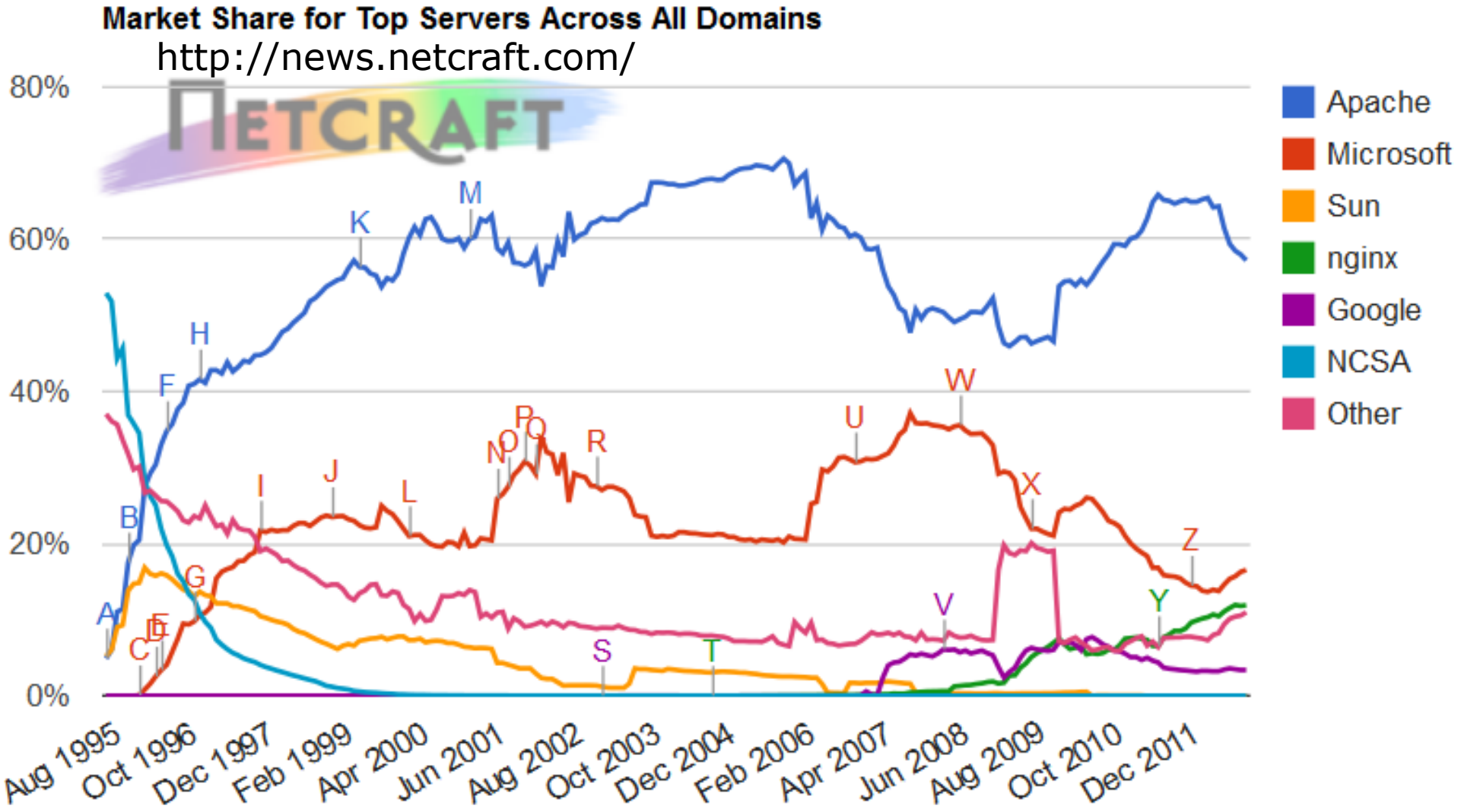
OSS case study: Apache httpd

Freie Universität Berlin

- History:
  - The NCSA (at University of Illinois) developed the web's first widely used web server software, httpd, in the early 1990s
  - When the primary author left NCSA, several httpd users started sending around bug fixes ("patches") and improvements by email
  - When this process became too tiresome, it evolved into a proper project for developing what would be Apache httpd
- Status today:

  - A majority of all websites use Apache httpd
  - Highly stable and function-rich Web Server
  - Plug-in architecture with about 90 default extensions ("modules")
    - and over 400 more released by people outside the Apache project
  - Core team size about 60 people, democratic process
  - Founding project of the Apache SW foundation
    - http://www.apache.org now 179 different projects (2012-11)

Case study: Apache httpd
Market share



Case study: Apache httpd
List of Apache modules

Freie Universität Berlin

- List of plugins delivered with httpd 2.2.2:
  - core, mpm common, beos, event, mpm netware, mpmt os2, prefork, mpm winnt, worker, mod actions, mod alias, mod asis, mod auth basic, mod auth digest, mod authn alias, mod authn anon, mod authn dbd, mod authn dbm, mod authn default, mod authn file, mod authnz ldap, mod authz dbm, mod authz default, mod authz groupfile, mod authz host, mod authz owner, mod authz user, mod autoindex, mod cache, mod cern meta, mod cgi, mod cgid, mod charset lite, mod dav, mod dav fs, mod dav lock, mod dbd, mod deflate, mod dir, mod disk cache, mod dumpio, mod echo, mod env, mod example, mod expires, mod ext filter, mod file cache, mod filter, mod headers, mod ident, mod imagemap, mod include, mod info, mod isapi, mod ldap, mod log config, mod log forensic, mod logio, mod mem cache, mod mime, mod mime magic, mod negotiation, mod nw ssl, mod proxy, mod proxy ajp, mod proxy balancer, mod proxy connect, mod proxy ftp, mod proxy http, mod rewrite, mod setenvif, mod so, mod speling, mod ssl, mod status, mod suexec, mod unique id, mod userdir, mod usertrack, mod version, mod vhost alias

Case study: Apache httpd
Process characteristics

Freie Universität Berlin

- Audris Mockus, Roy Fielding, James Herbsleb:
  "A Case Study of Open Source Software Development: The Apache Server",
  Intl. Conf. on Software Engineering, ACM press, 2000
  - Analyzes the apache project during 1996 to 1998/99

- 50,000 emails sent over mailing list during this timeframe
  - essentially all communication goes over the list
  - phone and personal email are uncommon in many OSS projects
  - A voting system with quorum is used for decision-making
- Common code base is managed in a CVS system
  - a central file server, storing and coordinating subsequent versions of each file
- Change requests are managed in a problem reporting database (BUGDB)

Case study: Apache httpd
Process characteristics (2)

Freie Universität Berlin

- Members of the Apache Group (AG) have write access to CVS
  - A person can become a member after about 6 months of contributing to the project
  - Current members vote on acceptance of new members
  - There were 8/12/12/25 members in 1995/1996/1998/2000
    - 66 members in 2012-11
- "Core developers" are the active AG members plus a few soon-to-become-members contributors (typically 2-3)
- People tend to work on those parts of the product they are most familiar with
  - Often equivalent to implicit code ownership: The opinion of a creator of code area X has greater weight in discussions about X
  - Therefore, new developers tend to either develop new features or take over code of a developer that is leaving

Case study: Apache httpd
Process characteristics (3)

Freie Universität Berlin

The size of the Apache development community:

- 182 people contributed to 695 changes related to problem reports (PRs)
- 249 people contributed to 6092 other changes or extensions
- Overall, almost 400 people contributed code
- 3060 people submitted the 3975 problem reports
  - 458 of them submitted the 591 that lead to one or more changes

Magnitude hypothesis for successful OSS projects:

  - if core developers := 1 then developers=10, bugreporters=100

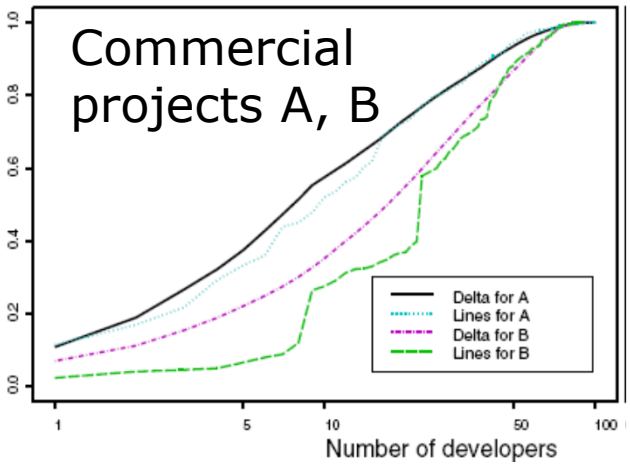
How widely was work distributed among people?

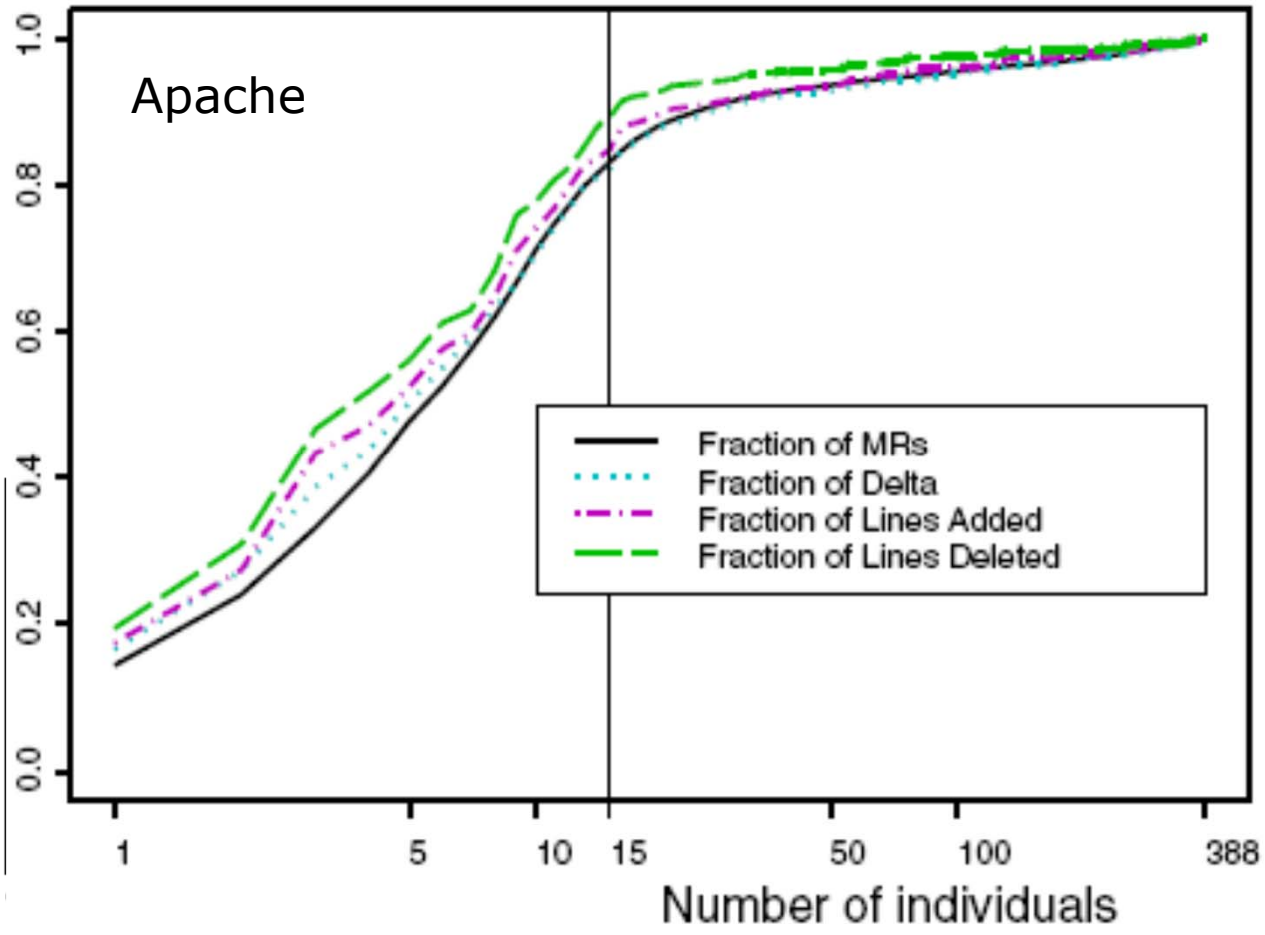
- The top 15 developers (out of 388!) contributed 83% of the change transactions, 88% of the added lines, and 91% of the deleted lines
  - (see graph on next slide)
  - i.e. by far most people make few and small changes only

Case study: Apache httpd
Process characteristics (4)

Freie Universität Berlin

- Distribution of number and size of contributions over people
  - most pronounced for new code: there are 4 developers per 100 non-PR changes, but 26 per 100 PR changes
    - PR: problem report





Case study: Apache httpd
Process characteristics (5)

Freie Universität Berlin

- **MRs:** number of changes (**m**odific. **r**equest)
- **Delta:** number of files changed

Table 1. *Statistics on Apache and five commercial projects.*

	MRs (K)	Delta (K)	Lines added (K)	Years	Developers
Apache	6	18	220	3	388
A	3.3	129	5,000	3	101
B	2.5	18	1,000	1.5	91
C	1.1	2.8	81	1.3	17
D	0.2	0.7	21	1.7	8
E	0.7	2.4	90	1.5	16

domain. Project A is code for a wireless base station, project B is a port of legacy code for an optical network element, and projects C, D, and E represent various applications for operations, administration, and

Case study: Apache httpd
Process characteristics (6)

Freie Universität Berlin

Measure	Apache	A	C	D	E
Post-release Defects/KLOCA	2.64	0.11	0.1	0.7	0.1
Post-release Defects/KDelta	40.8	4.3	14	28	10
Post-feature test Defects/KLOCA	2.64	*	5.7	6.0	6.9
Post-feature test Defects/KDelta	40.8	*	164	196	256

No system-testing is common in OSS

Avoids favoring bloated code

- Note that Apache is much higher-used than A, C, D, E
  - so the numbers will represent a higher fraction of all defects

Sketch: Typical Open Source Software process

Freie Universität Berlin

- Kim Johnsson: *"A descriptive process model for OSS dev."*, M.Sc. thesis, Univ. of Calgary, 2001
  - Lists a number of characteristics found in most OSS projects

Driving forces:

- Prototyping is closed
  - Most projects start as closed-source by an individual or group
- User-driven requirements, developers are often users

Organization view:

- Collaboration is decentralized
  - not much hierarchical communication
- Participation is tiered
  - OSS "career": the onion model
- Leadership is trust-based (meritocracy)
- Planning is informal

Sketch (2): Typical Open Source Software process

Freie Universität Berlin

Development style:

- Enhancement is iterative and incremental
  - OSS tends to have far more releases than commercial SW
- Communication is asynchronous
  - Almost exclusively via email
- Architectures are designed for modularity
  - To minimize coupling and hence communication effort
    - see the modules in Apache, plugins in Eclipse etc. etc.
- Tool support is ubiquitous
  - Issue tracking, version management, automated build and test
- Information space is shared
  - Central repositories for source code, documentation, issue database, releases etc.

OSS conventional-view success factors: Cathedral & Bazaar

Source:

- Eric Raymond: *"The Cathedral and the Bazaar"*
  - http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/ (Version 3.0 (rev. 1.57), September 2000)

Describes two styles of software development:

- **Cathedral style**: (=most of the commercial world)
  - closely integrated groups of highly skilled individuals plan thoroughly and implement with much care and no haste
  - *"built like cathedrals, carefully crafted by individual wizards or small bands of mages working in splendid isolation, with no beta to be released before its time"*
- **Bazaar style**: (=most of the open source world)
  - open for participation by everyone, hardly any central planning, no competence guarantee, quickly evolving
  - *"resemble a great babbling bazaar of differing agendas and approaches "*

OSS conventional-view success factors: Feedback, co-testing

Freie Universität Berlin

- Using Linux and fetchmail as case studies, the essay formulates several success factors for SW development
  - All of which can be realized by the bazaar style

- *"6. Treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging"*
  - Note that works only for sufficiently technical users
- *"7. Release early. Release often. And listen to your customers."*
- *"8. Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone."*
  - *"Or, less formally, 'Given enough eyeballs, all bugs are shallow.' I dub this: 'Linus's Law'*".
    - The Linux kernel is indeed proof that this principle can work.
  - This can work well if combined with 7.

OSS conventional-view success factors: Better defect reports

Freie Universität Berlin

Why finding and fixing defects is easier with OSS:

- In closed source cases, users and developers use different mental models of the system
  - users: surface phenomena
  - developers: code structure, program state and control flow
- But defect reports stated in terms of surface phenomena are often useless
  - because the failure can often not be reproduced
    - e.g. because the user did not report some important condition
- In contrast, Open Source gives users the chance to report defects directly in terms of problematic program elements
  - For difficult-to-locate defects with multiple symptoms or multiple different paths from symptom to defect, it is useful if many people attempt to find a path:
    One will stumble over a simple path even if most will fail.

OSS conventional-view success factors: Collective design

Freie Universität Berlin

- *"It is not only debugging that is parallelizable; development and (to a perhaps surprising extent) exploration of design space is, too."*

Preconditions for founding a successful OSS project:

- *"A certain base level of design and coding skill is required"*
- *"It is absolutely critical that the coordinator be able to recognize good design ideas from others"*
  - But you need not have those ideas yourself (Linux is an example)
- *"A bazaar project coordinator or leader must have good people and communications skills."*

- *"19. Provided the development coordinator has a communications medium at least as good as the Internet, and knows how to lead without coercion, many heads are inevitably better than one."*

OSS conventional-view success factors: Summary

Freie Universität Berlin

Summing up:

- The biggest advantage of OSS development over closed source is the large number of contributors it makes possible
- This one factor helps in many dimensions:
  - Development speed (time-to-maturation)
  - Requirements and Usefulness
  - Correctness
  - Design quality
- A second important factor is developer self-selection combined with meritocratic developer selection
  - developers are motivated; only competent ones will be accepted
- A third is release planning without deadlines
  - or alternatively sometimes planning with variable feature sets

Note: OSS culture

Freie Universität Berlin

- How and why these advantages can be realized has a lot to do with
  - the culture in the OSS domain in general
  - and the factors motivating the participating individuals

- There is another essay by Eric Raymond that covers this topic: *"Homesteading the Noosphere"*,
  - http://www.catb.org/~esr/writings/homesteading/
- Main points:
  - The OSS community has a "gift culture":
    You are respected for making valuable gifts to the community
    - The culture has varying degrees of underlying OSS zealotism and anti-commercialism
  - Individual participants are motivated by striving for reputation
  - Hence many people tend to work ("homestead") where they expect the most reputation to be gained

Note: OSS culture (2)

Freie Universität Berlin

- Somewhat different empirical findings are described in
  - Rüdiger Glott, Bernhard Krieger, Gregorio Robles: "FLOSS - Part 4: Survey of Developers", 2002, http://www.infonomics.nl/FLOSS/report/FLOSS_Final4.pdf
- Main reasons for joining or staying in OSS community:
  - 1. develop new skills (~75%),
  - 2. share knowledge and skills (~60%),
  - 3. improve OSS products (~35%),
  - 3. participate in a new form of cooperation (~35%),
  - 3. think that SW should not be proprietary (~35%),
  - 6. solve a problem (~30%)
- and several others
  - including getting a reputation (~10%), making money (~10%)

OSS economical-view success factors: Magic Cauldron

Freie Universität Berlin

- Eric S. Raymond: *"The Magic Cauldron"*, http://www.catb.org/~esr/writings/cathedral-bazaar/magic-cauldron/

- Discusses economical mechanisms underlying the OSS movement
  - because indeed not all participants are freaky zealots working in their spare time (hobbyists). Rather, other kinds are for instance:
    - Developers working to solve some company's problem (*)
    - Users wanting their own problem to be solved
    - Free-lance developers working to increase their professional reputation
  - Describes cases when it makes commercial sense to go open source

(*) FLOSS report: These were about 30% of all in 2002!

OSS economical-view success factors: Free Riding

Freie Universität Berlin

- A physical good, when available for free, will be overused and hence damaged or destroyed ("free riding")
  - "Tragedy of the Commons" (William F. Lloyd, Garrett Hardin)
- However, an intellectual good such as software may even gain from being available for free. Reasons:
  - Software/ideas are not damaged when used by more people
  - Market share increases and thus quality improvements become more likely: It is sufficient that any one person sees making an improvement as rewarding (for himself/herself)
  - That all others get the same improvement for free is irrelevant
    - It is impossible to realize the potential market value of small improvements (if that exists at all)
    - Rivals are unlikely to profit from the revealing
    - There are things that the free-rider cannot get without participating (fun of writing code, community, learning)
  - To not openly submit might actually cost something: work for re-integrating the improvement with future versions

OSS economical-view success factors: Sales value vs. use value

Freie Universität Berlin

- Economic reasons for closing source:
  - Protecting sales value
  - Denying others the knowledge embedded in the SW
- ➔ Candidates for opening source:
  - All SW that has no sale value (for you!) and that does not contain crucial knowledge
- Ego-centric benefits from opening source
  - Get free help from others for maintaining the SW
  - Possibly get improvements to the SW you would never make yourself
  - Reputation
- Opening source reduces sale value, but increases use value
  - There are multiple situations when this is economically advisable:

OSS economical-view success factors: Cost sharing scenario

Freie Universität Berlin

Commercial OSS justification 1:
Cost sharing (The Apache case)

- Assume you need a flexible, reliable, high-performance web server with certain specific features
  - You have three choices:
    1. Buy one (and have vendor risk),
    2. Build your own (and spend a lot of money) or
    3. Join the Apache Group

- Investing into Apache development is actually your cheapest route and hence economically sensible

OSS economical-view success factors: Maintenance risk reduction scenario

Commercial OSS justification 2:
Risk reduction (The Cisco Print Spooler case)

- Assume you have created some useful in-house solution for a problem that is not business-critical
  - e.g. Cisco built a modification of the Unix print spooling service that could re-route "low on toner" print jobs to nearby printers in a global company network, notify administrators, etc.
- You would like to assure you can maintain the solution even if its (few) developers leave your company
  - 2 in Cisco's case
- Your best route is opening source and getting other companies to start using the same solution
  - You may even get further improvements for free

OSS economical-view success factors: Market positioning scenario

Freie Universität Berlin

Commercial OSS justification 3:
Loss Leader/Market Positioner (The Mozilla case)

- Opening source does not only deny you sale value, but also your competitors (for similar products)
  - This can also help keeping a competitor from achieving quasi-monopoly status
  - or from entering a market in the first place
- When Netscape opened source of their Mozilla browser it was to deny Microsoft a monopoly with Internet Explorer
  - which would have cut into Netscape's Web Server business via the de-facto control of HTML and HTTP by Microsoft
- Similar arguments apply to Star Office / OpenOffice and a number of other cases

Loss leader = Lockvogelangebot

OSS economical-view success factors: Widget frosting scenario

Freie Universität Berlin

Commercial OSS justification 4:
Widget frosting (The Darwin case)

- If you are building hardware, you need accompanying software but that software does not have sales value itself
  - e.g. device drivers for network/graphics/sound cards, printers
- Opening source brings you the benefits of free help at no loss
  - It is usually impossible anyway to deny your competitors access to any valuable secret in the code
- Example: In 2000, Apple Computers opened the Darwin operating system kernel (the heart of Mac OS X)
  - (Note that OpenDarwin was not successful; later shut down)

widget = Dingsbums, frosting = Zuckerguss

OSS economical-view success factors: Company reputation scenario

Freie Universität Berlin

Commercial OSS justification 5:
Give away a recipe, thus advertise your restaurant
(The Zope case)

- Service companies can immensely increase their name recognition and reputation by opening source on internal products
- Examples:
  - Cygnus Solutions — support for GNU tools (1989!)
  - Red Hat, SuSe — Linux support and services
  - Zope Inc. (formerly Digital Creations) — web development
- Raymond also describes less important models he calls
  - "Accessorizing" (O'Reilly),
  - "Free the future, sell the present" (Aladdin),
  - "Free the software, sell the brand" (Sun/Java),
  - "Free the software, sell the content" (could be AOL)

OSS economical-view success factors: User risk reduction effects

Freie Universität Berlin

Furthermore, there are added benefits for the users (that reflect back on the supplier):

- Reduced vendor lock-in
- Reduced risk of vendor going out of business
- Improved transparency of product
- Improved visibility of future developments

So high-payoff situations for OSS are the following:

- Reliability/stability/scalability are critical
- The software is a business-critical factor (e.g. Apache)
- Software that establishes or enables a common computing infrastructure (→ highest use of network effects)
- Its key methods are part of common engineering knowledge (→ less reason for going closed source)

End of part 1

OSS leadership and decision-making

Freie Universität Berlin

- By and large, OSS projects tend to have a meritocratic leadership model
  - Influence is won by making valuable contributions to the project
  - and by exhibiting technical and judgmental competence
- (exceptions possible when corporate sponsoring is present)

This statement raises two questions:

1. What is the process (in terms of milestones) of gaining influence for an individual?
   - Put differently: Are there clearly different degrees of influence that can be easily observed? (An "OSS career")
2. How does actual decision-making work?
   - Given that influence cannot easily be quantified

The OSS career

The typical career of an active OSS project participant:

1. Knows product
   - User
2. Knows process/project
   - Mailing list member: 2.1. Follows and 2.2. participates in the discussions in the project
3. Contributes suggestions to product
   - 3.1. Sends in defect reports or helps clarifying issues
   - 3.2. Sends in defect corrections ("bug fixes", "patches") to be checked and accepted by the developers
4. Has write-access to product
   - Developer status: can modify the source code version archive
5. Has meta-write-access to product
   - Can grant others write-access. Called differently in different projects (core developer, maintainer, leader)

Perhaps more stages here

The OSS career (2)

Freie Universität Berlin

- In small projects there is often a single person with meta-write access who makes the decision at his/her own discretion
- Some large projects define various roles and behavior explicitly and may have formalized decision-making rules and decision-making bodies for granting write-access (*join-scripts*)
  - http://httpd.apache.org/dev/guidelines.html
  - http://docs.python.org/devguide/
  - https://developer.mozilla.org/en-US/docs/Developer_Guide
  - http://www.openoffice.org/dev_docs/guidelines.html
- Some large projects also discriminate many different kinds of contributions (and corresponding roles) more clearly

  - e.g. OpenOffice (http://incubator.apache.org/openofficeorg/orientation/index.html): Development, QA, Localization, Marketing, Website Dev.

OSS decision-making (1)

The leadership structure (formation of opinion) of OSS projects is spread over a spectrum with the following poles:

- Egalitarian:
  - In any issue, the influence of an individual depends mostly on convincing argumentation.

- Leadership group:
  - The influence depends mostly on the individual's general reputation
    - which may be formalized or not

- Note: A leadership group without merits could not persist or would lead to forking. Thus, the difference between the poles is <u>not</u> huge.

A quote

"Does corporate development have communist roots,
with its strong belief in central planning and
the interchangeability of production programming staff units?"

Dave Thomas and Andy Hunt

OSS leadership types: Case studies

Freie Universität Berlin

In terms of leadership and decision-making structure, most small OSS projects fall into the categories 'hobbyist' or 'research project'.

Most larger ones fall into one of the following categories:

1. Democratic model
   - Example: Apache foundation
2. Benevolent dictator model
   - Examples: Linux, Python
3. Industry-based
   - Examples: Mozilla, OpenOffice, JBoss
4. OSS foundation projects
   - Apache, GNU, Debian
5. Industry-driven
   - Eclipse

see subsequent slides

OSS leadership types 1: Democratic model

Freie Universität Berlin

- A group of people use explicit democratic decision processes and drive the project like a society drives a democratic state
- Example: **Apache** software foundation
  - Quotes from http://www.apache.org/foundation/how-it-works.html#management (as of 2012-11)
  - "Projects are normally auto governing and driven by the people who volunteer for the job. […] ***"do-ocracy"*** -- power of those who do. This functions well for most cases.
  - When coordination is required, decisions are taken with a **lazy consensus approach**: a few positive votes with no negative vote is enough to get going. […]
  - […] a negative vote includes an **alternative proposal** or a detailed explanation of the reasons for the negative vote.
  - […] In the great majority of cases, the concerns leading to the negative vote can be addressed.
  - This process is called ***"consensus gathering"*** and we consider it a very important indication of a healthy community."

OSS leadership types 2: Benevolent dictator model

Freie Universität Berlin

- A single highly respected person makes all important decisions
- Examples: **Linux**, Python

Linus Torvalds

- In 1991, the finnish student Linus Torvalds started writing an operating system kernel
  - His message on comp.os.minix in August 1991: http://groups.google.com/group/comp.os.minix/msg/b813d52cbc5a044b
  - *"I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. […] It is NOT portable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks"*
- Today, the Linux kernel consists of at least 4 MLOC
- Yet Torvalds' few deputies still have to accept every change to this code before something is official

OSS leadership types 2: Benevolent dictator model (2)

Freie Universität Berlin

- Guido van Rossum started developing the programming language **Python** in 1990
  - In 1996, he wrote (in the introduction of Mark Lutz' book "Programming Python"): "[…] in December 1989, I was looking for a 'hobby' programming project that would keep me occupied during the week around Christmas."
- Today, Python is one of the most popular scripting languages and teaching languages
- The Python development community calls van Rossum the "Benevolent Dictator For Life" (BDFL)
  - for an example of this status, see the next slide

Guido van Rossum

OSS leadership types 2: Benevolent dictator model (3)

Freie Universität Berlin

- Until Version 2.4, Python does not have a C-like conditional expression operator (condition ? valueIf : valueElse)
- There was much discussion about the proper syntax for adding one, because Python style requires that it be much more self-explanatory than the C format
- After long discussion, van Rossum made the final decision alone and wrote
  - http://mail.python.org/pipermail/python-dev/2005-September/056846.html :
  - "**After a long discussion I've decided to add a shortcut conditional expression to Python 2.5. The syntax will be**

    A if C else B

    […some explanation of details and consequences…]

    Flames, pleas to reconsider, etc., to /dev/null. Congratulations gracefully accepted. It's still my language! :-) "

OSS leadership types 3: Industry-based

Freie Universität Berlin

- Most project members come from one industrial employer
  - they often work full-time for the project
  - and are being paid for it
- Examples: Mozilla, MySQL, JBoss

Where does the money come from?

- **Mozilla**: Mozilla Foundation (mostly from Google)
  - formerly part of Netscape
- **MySQL**: Professional services of Oracle
  - formerly MySQL AB (sold for US$ 800 mio), then Sun Microsystems
- **JBoss**: Professional services of Red Hat Inc.
  - formerly JBoss Inc. (sold for US$ 420 mio after 7 years)

OSS leadership types 4: OSS foundation projects

Freie Universität Berlin

- A formal organization (often called a foundation) is build in order to host a significant group of related projects that have something important in common
  - such as technology, leadership/governance principles, or philosophical principles
  - May or may not have a protagonist or main sponsor

Examples:

- Apache Software Foundation
  - is a non-profit corporation with 501(c)(3) U.S. charity status
    - members are individuals, new ones accepted by current member vote
  - Goals: Support OSS projects , create a reputable receiver for donations , provide legal shelter to project participants , protect the "Apache" brand
  - Runs several highly regarded projects
  - Runs an "incubator" for systematically integrating further projects into the foundation

Apache Incubator

Freie Universität Berlin

- As of Nov 2012, has 39 candidates
- Has a detailed formalized process for how a project can become an ASF project
  - http://incubator.apache.org/incubation/Incubation_Policy.html
- To become a candidate, a project must write a proposal
- and must have the support of
  - a Champion: An ASF member
    - http://www.apache.org/foundation/members.html
    - as of November 2012, 404 individuals are members
  - a Sponsor: Either the ASF Board or an Apache Top-Level Project or the Incubator Project Management Committee
- To become an ASF project, the candidate must
  - put all code under Apache license, resolve trademark issues
  - work in "the Apache way" (large community, voting, meritocracy, conflict handling, release planning, etc.)
  - create synergy with other Apache projects

OSS leadership types 4: OSS foundation projects (2)

Freie Universität Berlin

FREE SOFTWARE FOUNDATION

- The Free Software Foundation (home of GNU)
  - Original goal was a completely free Unix OS
  - Main Principle is that of Free Software as formulated by the GNU General Public License
  - Protagonist: Richard Stallman
  - GNU lists several hundred "GNU project entries"
    - http://directory.fsf.org/GNU/
    - But their association with the FSF is much looser than for ASF, often only via the GPL license
- Much of the backbone of Linux actually comes from GNU:
  - system utilities, shell, etc.
  - compilers, linker, C library etc.
- The FSF as a circumscribed organization is now mostly concerned with rallying free software (not with developing it)

OSS leadership types 5: Industry-driven: Eclipse

Freie Universität Berlin

- This model is a hybrid of the foundation type and the industry-based type

The most important representative is Eclipse:

- Eclipse was initially a proprietary development effort at IBM
  - building the successor for VisualAge for Java 4.0
- Then IBM formed a consortium with a number of companies
  - Borland, MERANT, QNX, Rational, Red Hat, SuSE, TogetherSoft, Webgain
  - for developing an IDE development platform
  - The initial format was a project for developing an IDE, but with special focus on UI infrastructure
  - The later goal: a universal platform from which to construct the user interfaces of all of IBM's interactive software products
  - This development was open source from the beginning
- Eclipse has now been transformed into a foundation,
  - still highly sponsored by IBM,
  - but with broader involvement of others in the decision-making

OSS licenses: Overview

Freie Universität Berlin

- There is a rather large number of OSS licenses that define the rights of the public with respect to the software
  - e.g. (as of 2006-10) Academic Free License ● Adaptive Public License ● Apache Software License ● Apache License, 2.0 ● Apple Public Source License ● Artistic license ● Attribution Assurance Licenses ● New BSD license ● Computer Associates Trusted Open Source License 1.1 ● Common Development and Distribution License ● Common Public License 1.0 ● CUA Office Public License Version 1.0 ● EU DataGrid Software License ● Eclipse Public License ● Educational Community License ● Eiffel Forum License ● Eiffel Forum License V2.0 ● Entessa Public License ● Fair License ● Frameworx License ● GNU General Public License (GPL) ● GNU Library or "Lesser" General Public License (LGPL) ● Historical Permission Notice and Disclaimer ● IBM Public License ● Intel Open Source License ● Jabber Open Source License ● Lucent Public License (Plan9) ● Lucent Public License Version 1.02 ● MIT license ● MITRE Collaborative Virtual Workspace License (CVW License) ● Motosoto License ● Mozilla Public License 1.0 (MPL) ● Mozilla Public License 1.1 (MPL) ● NASA Open Source Agreement 1.3 ● Naumen Public License ● Nethack General Public License ● Nokia Open Source License ● OCLC Research Public License 2.0 ● Open Group Test Suite License ● Open Software License ● PHP License ● Python license (CNRI Python License) ● Python Software Foundation License ● Qt Public License (QPL) ● RealNetworks Public Source License V1.0 ● Reciprocal Public License ● Ricoh Source Code Public License ● Sleepycat License ● Sun Industry Standards Source License (SISSL) ● Sun Public License ● Sybase Open Watcom Public License 1.0 ● University of Illinois/NCSA Open Source License ● Vovida Software License v. 1.0 ● W3C License ● wxWindows Library License ● X.Net License ● Zope Public License ● zlib/libpng license
  - for details see http://www.opensource.org/licenses/
- but they all derive from only 2 basic types:

OSS licenses: Essentials

Freie Universität Berlin

- We have seen Stallman's definition of Free Software
  - which appears somewhat vague, at least untechnical
- According to the OpenSource Initiative (opensource.org), the defining characteristics are the following:
  - (see http://www.opensource.org/docs/definition.php)

1. Right of free redistribution
2. Source code availability
3. Derived works (and their distribution) are allowed
   - but may require distribution as "unmodified sources + patches"
4. Undue restrictions must not be present:
   - no discrimination against persons or groups (e.g. "valid for IBM employees only"),
   - no discrimination against fields of endeavor (e.g. "no military use"),
   - no further steps required (e.g. signing a non-disclosure agreement, making a registration), etc.

OSS licenses: 2 basic types (GPL, BSD)

Freie Universität Berlin

The most crucial difference between licenses is their requirements for derived works:

- The "viral" ("copyleft") licenses require that all derived works, if distributed, are also distributed under the same license
  - Prototype representatives: *GNU General Public License* (GPL), *GNU Lesser General Public License* (LGPL)
    - They differ in their definition of what is considered a derived work (strong/weak copyleft)
  - Note that private (undistributed) derived works are allowed
- The liberal licenses allow that derived works can be published under a different license
  - perhaps including closed source licenses
  - Prototype representative is the *modified BSD license*

OSS licenses: Other types (MPL, variants)

Freie Universität Berlin

- The other licenses can be considered "in between" GPL and BSD license

- Sort of a middle ground is defined by the *Mozilla Public License* (MPL):
  - it discriminates deriving from existing parts (which must keep their previous license) from deriving by adding new parts (for which one can choose a license freely)
  - This means that, say, a company can still build proprietary extensions of a work, but has to publish changes in existing parts back to the community.
- Most licenses differ from these 3 types only by minor additional restrictions or permissions
  - and sometimes only in their wording of what are essentially the same rules

OSS licenses: Consequences

Freie Universität Berlin

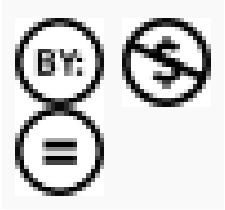
- When creating derived works from multiple OSS products at once, make sure the respective licenses are compatible
  - You will sometimes need a lawyer to answer that question
  - Example problem: The original Apache Software License was not compatible with the GPL since it contains a patent retaliation clause. (Apache Version 2 has resolved that)
    - http://en.wikipedia.org/wiki/List_of_FSF_approved_software_licenses

- The biggest obstacle is usually the "viral" nature of the GPL:
  - If you combine existing software A and B with your own C
  - and B is licensed under GPL
  - the GPL requires you to publish the resulting whole under GPL
  - which the license of A (say, MPL) may not allow.

  - E.g. to make Mozilla compatible with the GPL it contains code that is tri-licensed (GPL/LGPL/MPL).

Creative Commons

- OSS licenses mean that authors give away much of their copyright privileges, because they believe that is a good idea
- The same can apply to non-software works, too
- Such licenses are defined by the Creative Commons project
  - to be used for text, pictures, music, audio, video, multimedia, etc.
  - http://www.creativecommons.org
    - Founded by Lawrence Lessig
- Creative Commons has a modular license system, in which an author can turn the following options on or off in a license
  - require that the author be named
  - allow commercial use
  - allow derived works
  - allow redistribution under the same terms ("share-alike")
  - (all variants allow public use of the work)



Licenses: Commercial Implications

Freie Universität Berlin

- Most copyleft licenses don't forbid selling the software
  - But you still have to provide the source code
  - Basically you are allowed to charge a fee for the physical act of transporting the software.
- BSD license is the best choice for commercial applications.
- If the copyright is held by a single entity, a common move is:
  - Offer the software dual-licensed.
  - Companies can either use the free version and have to share their development or they pay and can derive proprietary products.
  - Examples (at some time): MySQL, QT, Asterisk, Berkeley DB

OSS tool support

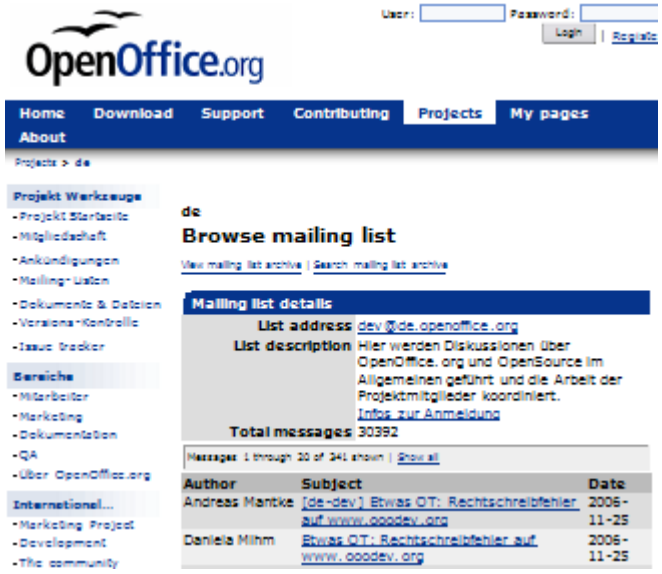
Typical elements of software tool support in OSS projects:

- All of these tools are in some way concerned with coordination

- A central versioning repository
  - Almost always either CVS or Subversion (svn)
  - A notable exception is Linux, which has its own tools
- One or several mailing lists for communication
  - among developers, between users and developers, perhaps between users (user groups)
- Often a central Wiki server
- A central issue-tracking system
  - e.g. Bugzilla, GNATS, Mantis, Trac
- Automated daily builds (or even continuous builds)
  - A variety of tools and frameworks exist to support this (e.g. Ant, Maven, SCons, Tinderbox, CruiseControl, Gump)

One or several mailing lists for communication

- Some projects (especially the smaller ones) have one list for everything and everybody
- Others split lists by topics and target groups
- E.g. OpenOffice mailing lists (Oct 2006)
  - All @openoffice.org: users@, announce@, discuss@, social@, webmasters@, dev@, allfeatures@, allcvs@, allbugs@, interface-announce@, interface-discuss@, releases@
  - Low-volume announcements lists and higher-volume discussion lists

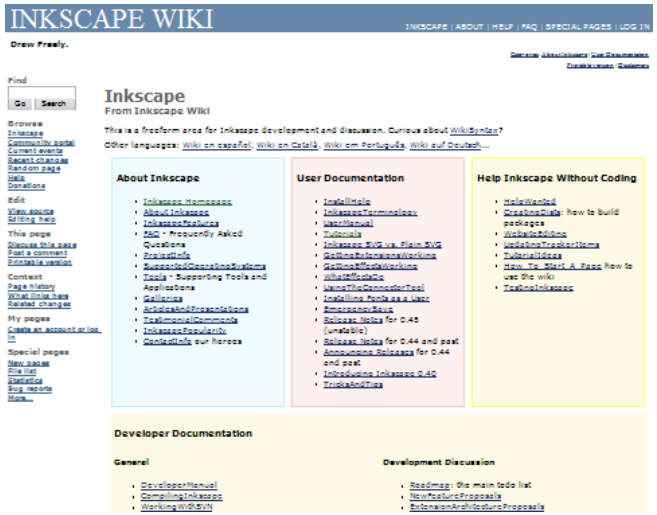


- Some projects suggest that all project-related communication should happen over the mailing list
  - i.e. no private email between participants on project matters

A central Wiki server

- Some projects use Wikis (webpages that can readily be edited by anyone) for centrally maintaining useful, but less critical information
  - Makes for a very low entry barrier for new would-be members
  - Can be used by people without programming skills to still contribute to the project
  - Especially useful for information that are frequently asked for, since readers can adapt them to new developments.
  - A good structure for a wiki is difficult to create and maintain
    - guidelines needed
- Example: http://wiki.inkscape.org

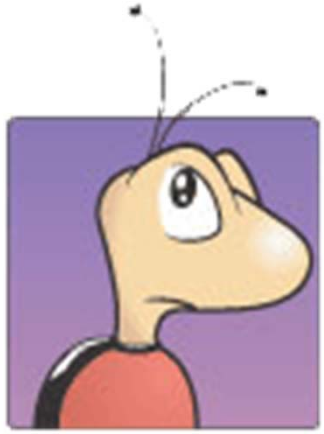
  (a vector drawing program) 4 wiki areas:
  - About Inkscape
  - User Docs
  - Developer Docs
  - Help Inkscape without Coding



A central issue tracking system: Bugzilla

Freie Universität Berlin

- Collects attributes of issues/change requests
- Helps manage workflow
  - person filters entry (if duplicate) or assigns entry to somebody
  - system tracks status: new, open, in work, fixed, tested, closed etc.



Bug #	Dupe Count	Change in last 30 day (s)	Component	Severity	Op Sys	Target Milestone	Summary
319196	40	15	Startup and Profile System	critical	Windows XP	---	customized toolbar always reset to default on restart, bookmarks and search engines lost, unable to add search engines (localstore.rdf corruption on upgrade or crash)
~~348227~~	9	8	RSS Discovery and Preview	normal	Mac OS X 10.3	Firefox 2 beta2	Live Bookmarks should take metadata from the Feed, not the page
349716	9	6	Keyboard Navigation	normal	Windows XP	Firefox 3 alpha1	Content accesskeys now require Alt+Shift
~~344422~~	6	5	Find Toolbar / FastFind	normal	Windows XP	---	New Quick search doesn't have "Find next", "Find previous" and other

Build tools

Freie Universität Berlin

- A build system is required to manage platform-independent compilation/testing/packaging of the software
  - Examples: Make, Ant, Maven, SCons

- Typically a build tool manages targets and dependencies between them.
  - Example: To run a java program we first need to run the target javac to compile the source files and then package a jar-archive.

```
<project default="compile">
  <target name="compile">
    <javac srcdir="." />
  </target>
  <target name="jar" depends="compile">
    <jar destfile="hello.jar" basedir="." includes="**/*.class"/>
  </target>
  <target name="run" depends="jar">
    <java classname="hello" classpath="hello.jar" fork="true"/>
  </target>
</project>
```

Continuous Integration

Freie Universität Berlin

- Since many developers work in parallel on the source code it is often hard to ensure that after committing the project still compiles.
- Also building packages for a large number of platforms is tedious.
- One solution: Continuous Integration
  - An automatic script watches the version repository and keeps a current version locally. It then builds the project for all platforms and runs regression tests.
  - If builds or tests fail, it "blames" the authors of last commits.
  - e.g. CruiseControl, Tinderbox, Jenkins, Gump

Build Time	Guilty	Linux argo-vm Dep Nightly	Linux balsa-trunk Dep GTK1 (gcc 3.4)	Linux bl-bldlnx01 Dep argo-vm test perf	MacOSX Darwin 8.7.2 xserve06 Dep Universal Nightly
Click time to see changes since then	Click name to see what they did				
2006/11/09 00:31:00	cvshook	L Z:14.52MB mZ:2.472MB			

Process Improvements and OSS

Freie Universität Berlin

The remainder of this slide set is concerned
with research performed by AG Software Engineering

- Assume we would like to perform process improvement
  - say, a la CMMI
- We know that this requires a lot of effort and time
- In a conventional SW organization, somebody high in the hierarchy makes the decision to do it and then commands it

- How does the equivalent process work in an OSS context?
  - Nobody has authority over project members → decisions are more complicated
  - Members are distributed → asynchronous discussion
  - Some improvements that are useful conventionally may not be useful here

Definition "innovation"

- Definition:
  Innovation means that a group adopts a new practice

Rogers

- Conforms to the usage by important authors,
  e.g. Everett Rogers, Peter Drucker, Harold Evans

- This definition is operational:
  - observable, executable

Drucker

- "Practice" refers to
  - habits, routines, and other forms of embodied recurrent actions
  - that are chosen and performed without conscious thought.

- In this sense, software process improvement is innovation

Innovation vs. invention

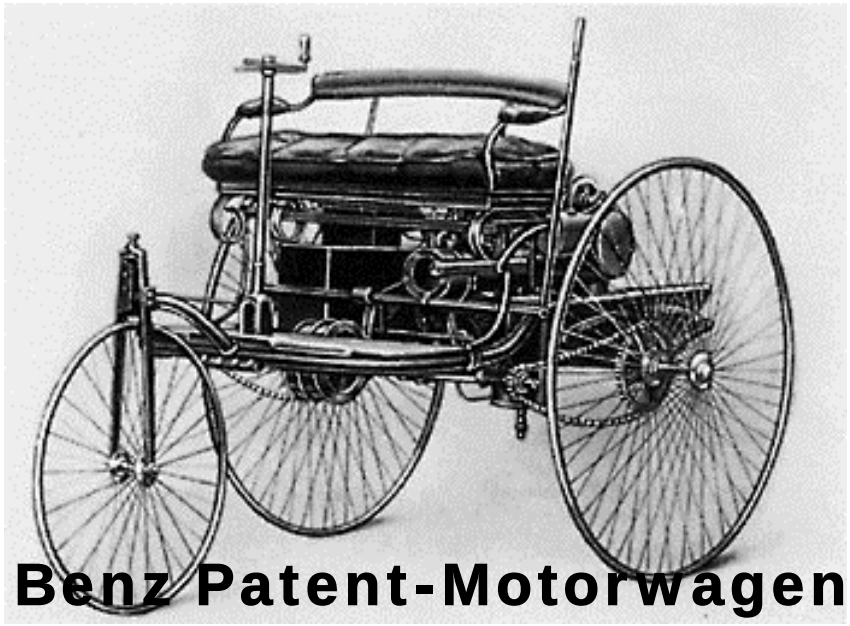
Freie Universität Berlin

- Invention is different from innovation.
  - Invention means to create something new,
  - but does not require that anyone accept or adopt it.

- Most inventions never become (or lead to) innovations
- Many innovations are not brought about by the inventor
- The same invention can lead to many innovations
  - one per group adopting it
- Innovation need not be unusual, widespread, or radical
  - and can happen slow or fast

Invention vs. innovation

- Carl Benz's first car was an invention
- but only Henry Ford's Model T brought the innovation
  - it was sufficiently cheap, reliable, available



Benz Patent-Motorwagen
1886

Ford Model T 1908

Invention vs. innovation

- Johann Philipp Reis invented the telephone 1860
  - others followed soon: Elisha Gray, Antonio Meucci
- Alexander Graham Bell did it again 1876,
  but then founded the Bell Telephone Company

Innovation as an active social process

Freie Universität Berlin

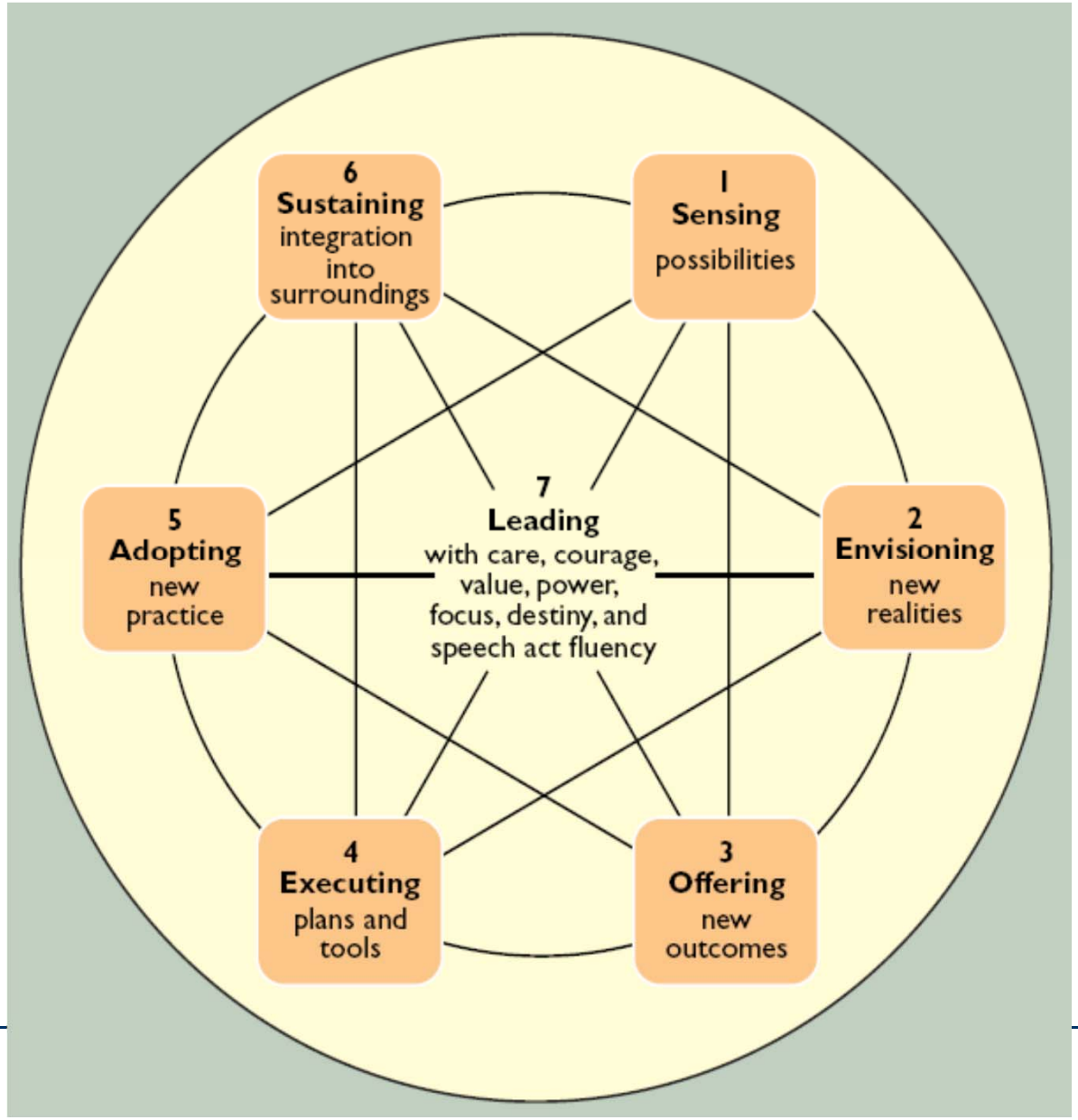
- Successful innovation is performed by following certain practices
- These practices can be trained and learned
  - presented in the form of a generative framework
- Technical capabilities are not at the heart of these practices
- Source: Peter J. Denning, Robert Dunham: *"Innovation as language action"*, Communications of the ACM 48(5), May 2006, pp.47-52, www.computer.org

Peter Denning

The generative framework: "Personal Foundational Practices"

Freie Universität Berlin

- 1 to 2: invention
  - 3,4: transition
- 5 to 6: adoption
- Not sequential steps!
  - more like parallel processes
- Each practice has both verbal and non-verbal aspects



Application to OSS process improvements

Freie Universität Berlin

AG Software Engineering investigates OSS process improvements:

- How do such innovations proceed?
  - And what can we learn from that? In particular:
- What does a would-be innovator need to do in order to maximize the chance of successful adoption?
  - How to identify candidate pairs of invention and project?
  - How to identify key people in the project?
  - How to communicate with the project?

Case study: The Moderator role

Freie Universität Berlin

- Communication and coordination are particularly difficult in OSS projects
- We **1_sensed** that it might be helpful to actively and explicitly promote coordination-related information in such projects
- We **2_envisioned** a new role in OSS projects, the Moderator, whose task is information management:
  - explicitly collecting and organizing information that speeds up information access for many participants (in particular new ones) and avoids redundant questions or searches
- We **3_offered** this "invention" to a project (GNU classpath)
  - We offered to set up a Wiki
  - There we could collect and structure information regarding e.g. design decisions

Case study: The Moderator role (2)

- The offer was accepted. We **4_executed**
  - by actually setting up the Wiki
  - by actually compiling initial information found in the mailing list archive and putting it there regarding (a) design decisions, (b) newbie instructions, (c) current development topics
- We continued maintaining this information, adding more from time to time and announcing it via email, thus triggering **5_adopting** the new practice
  - After some time, a few other project members started using the platform, too
  - Also for new purposes, such as arranging physical meetings
- Specific actions for **6_sustaining** the practice did not appear to be necessary
  - The Moderator role has apparently been distributed and filled since

Case study: The Moderator role (3)

- The details of our **7_leading** that made the effort successful still need to be understood
  - analyzing who did what when why
  - or not
- In order to understand the causation in the process, we need more examples of it

Lutz Prechelt, prechelt@inf.fu-berlin.de

Process Improvements and OSS (2)

Freie Universität Berlin

- Unfortunately, such participant observation research is far too time-consuming
  - one could not see enough innovation episodes that way
- Therefore, we switched to searching for innovation episodes on project mailing lists
  - chose medium-sized projects (10 to 50 members)
  - scanned the mailing lists of several hundred projects
    - and picked 12 projects for analysis
  - scanned thousands of email messages for innovation episodes
  - extracted the messages for about 100 such episodes
  - analyzed them in detail using the Grounded Theory Method
- Innovation episodes:
  - variable size (#messages, #participants, #days)
  - very different topics, some types of them recurring
  - often unsuccessful

Process innovation pattern: Partial migration

Freie Universität Berlin

- Context:
  - A process change was proposed
  - Many find it reasonable
- Forces
  - The change involves a lot of work for one person
    - and some work for everybody
  - It is risky or some members do not like it yet (are change-averse)
- Example:
  - Switch the version mgmt. from CVS to Subversion or from Subversion to a decentral system (e.g. git)
- Solution:
  - The change is made only for a fraction of the project at first
    - e.g. new repository created for one subsystem only
  - then tried out and adapted gradually
    - in order to distribute the workload and allow members to adapt slowly

Process innovation pattern: Adapter innovations

Freie Universität Berlin

- Context: A sensible process change was proposed
- Forces: Some members cannot or do not want to accomodate the future situation.
  - ➔ Resistance.
- Example: Change of version management software
- Solution: Create an adapter that allows those members to more or less stay in the previous mode
  - at least for a while

Process innovation pattern: Reduce enactment scope

Freie Universität Berlin

- Context: A sensible process change is proposed
- Forces: It involves a lot work work compared to its importance (or at least many members perceive it that way), or the benefits are unclear
- Example: Clean up bug tracker database after a release.
- Solution: Frame the suggestion as a one-time activity only. Wait and see how it worked out. Only then introduce it as a process change

OSS and CMMI

Freie Universität Berlin

- Level 2: Managed
  - Requirements Mgmt REQM
  - Project Planning PP
  - Project Monitoring&Control PMC
  - Supplier Agreement Mgmt SAM
  - Measurement and Analysis MA
  - Process and Product Quality Assurance PPQA
  - **Configuration Mgmt CM**
- Level 3: Defined
  - Req's. Development REQD
  - **Technical Solution TS**
  - Product Integration PI
  - Verification VER
  - **Validation VAL**
  - Organizational Process Focus OPF
  - Organ'l Process Definition OPD
  - Organizational Training OT
  - Integrated Project Mgmt. IPM
  - Risk Management RSKM
  - Decision Analysis and Resolution DAR
- Level 4: Quantitatively Manag'd
  - Organizational Process Performance OPP
  - Quantitative Project Mgmt QPM
- Level 5: Optimizing
  - Organizational Performance Management OPM
  - Causal Analysis and Resolution CAR

- may or may not be present
- **usually present to a large degree**
- almost never present

Further literature

Freie Universität Berlin

- http://www.infonomics.nl/FLOSS/report/
  - Empirical research project on aspects of OSS: company use of OSS, company participation in OSS projects, markets and business models, developer motives
- International Conference on Open Source Systems
  - e.g. http://oss2012.org/
- http://conferences.oreillynet.com/oscon/
  - O'Reilly Open Source Convention
- Michi Henning: "The rise and fall of CORBA", ACM Queue 4(5), June 2006, www.acm.org
  - A story how design-by-committee can fail that teaches how and why OSS proceses are often successful. Interesting!

Summary

Freie Universität Berlin

- OSS projects can produce high-quality software
  - and are at least as productive as conventional SW organizations.
  - Most use the Bazaar style of development
- There are sound economic reasons why companies contribute to OSS projects
- OSS projects follow a meritocratic leadership style
  - There are power structures, too, but based on merits and with strong democratic elements in the decision-making
- There are different types of OSS licenses
  - which can be mutually incompatible
- Most OSS projects use a typical set of development and coordination tools
  - Processes tend to be more lightweight than suggested by CMMI
  - Introducing new tools or processes is an interesting case of innovation management, because of the special group structure

Thank you!