Open Source Engineering Processes

Prof. Dr. Dirk Riehle

Friedrich-Alexander University Erlangen-Nürnberg

FOSS B05

Open Source Innovation

1. Legal innovation
2. **Process innovation**
3. Software tool innovation
4. Business model innovation

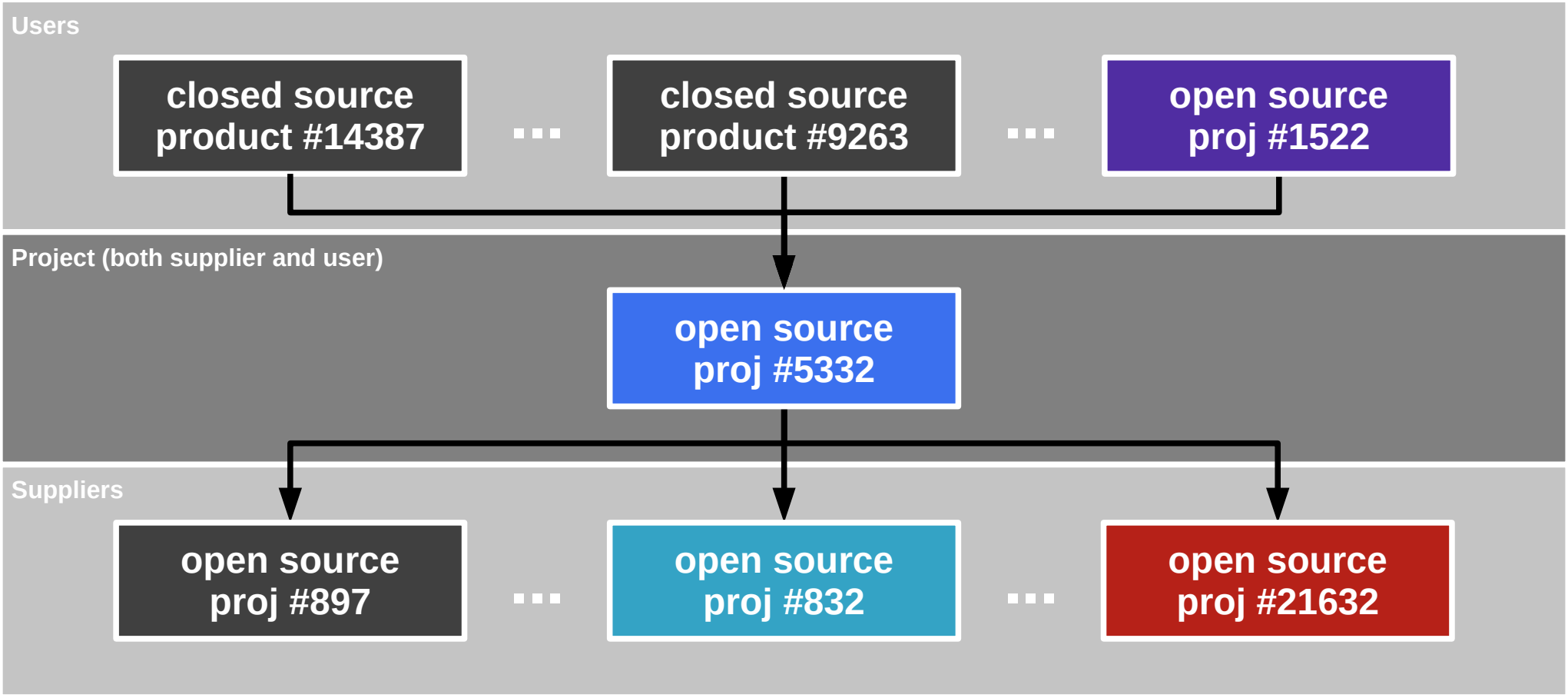
Open Source Process Innovation

- Community processes
- **Engineering processes**

Open Source Projects (Recap)

- Project community
  - People and companies engaged with the software
    - Includes using, developing, and marketing the software
- Software (components)
  - Not a product but a component (even if an application)
    - Users are therefore best advised to view project as a supplier
- Software users
  - Natural people scratch their own itch
  - Companies embedded software in products

The Software Supply Chain (Recap)

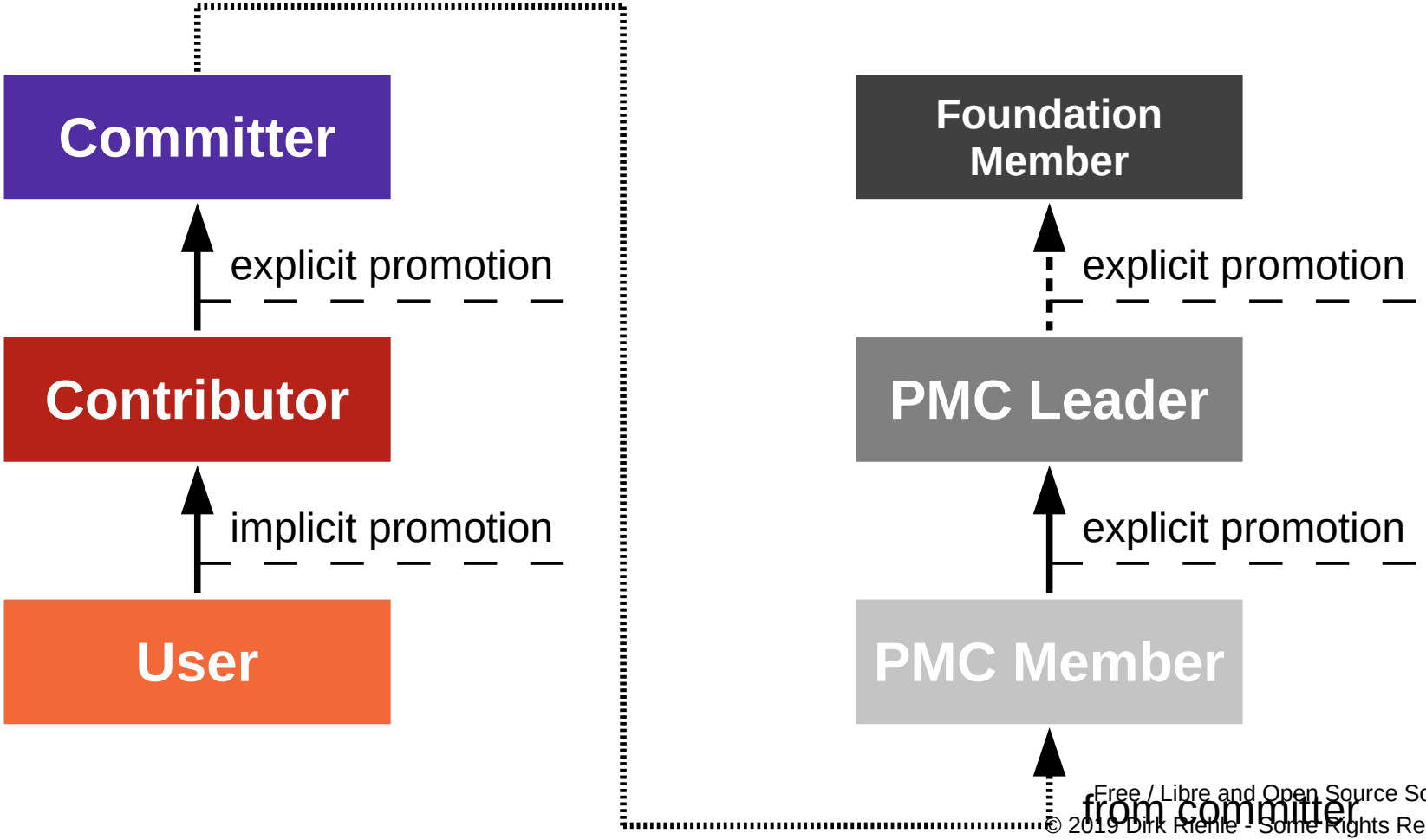


Key Roles in Software Engineering

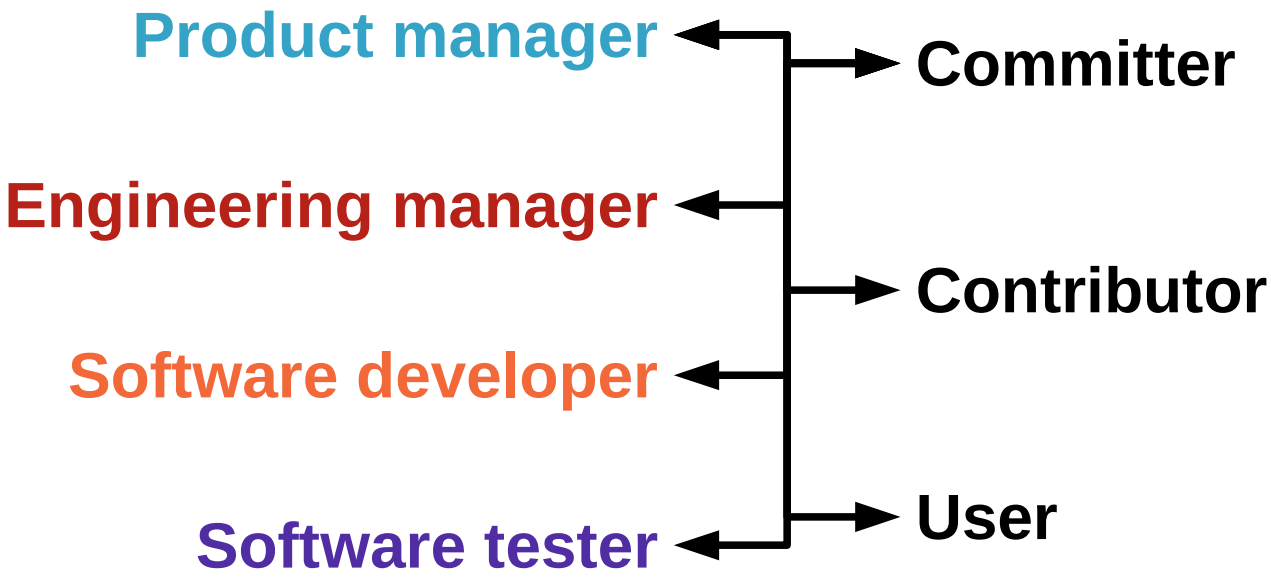
- Client-specific projects
  - Business analyst
  - Project manager
  - Software developer
  - Software tester / QA engineer
- **Products for a market**
  - **Product manager**
  - **Engineering manager**
  - **Software developer**
  - **Software tester / QA engineer**

Open source projects develop components for a market

Roles in Open Source Software Projects



Closed to Open Source Role Mapping



Product Management in Open Source

- Strategic product management
  - Does not take place in open source, is performed outside
- Technical product management
  - Product roadmapping
    - Some is performed but often is ad-hoc
  - Product specifications
    - Barely exist as documents (wikis, to-do lists, other)
  - Progress tracking
    - Managed by time, it is done when it is done

Engineering Management in Open Source

- Release planning
  - See product management: Some is performed but often is ad-hoc
- Resource allocation
  - Committers can prod contributors but that’s about it
  - Usually contributors pick up what they like to work on
- Process improvement
  - Ad-hoc, if any

Software Development in Open Source

- Programming
  - Like in closed source, but in general with less visibility as to completion

Quality Assurance in Open Source

- Code review
  - Is the core ingrained best practice followed by open source projects
- Automated testing
  - Like in closed source, though perhaps a bit more ad-hoc in general
- Manual testing
  - A lot of user testing, significantly more than in closed source
- Release management
  - Like in closed source, committers play release manager

Two Perspectives on the Discrepancy

- It is embarrassing for **open source**, because
  - It often does not know or apply the most basic best practices of product and engineering management
- It is embarrassing for **closed source**, because
  - All those best practices don’t seem to matter much: People figuring it out as they go along are equally or more effective
- The truth is probably somewhere in between
  - As so often, people matter more than processes and tools
  - Still, open source projects could learn a lot

Open Source Engineering Innovation

- There is no single process, only the one a community adopts
  - We can only talk about the 3-4-5 model of open communities
- Open source has a strong source code contribution process
  - A.k.a. peer review, patch submission, pull requests
- Distributed collaboration and version control
  - Developed and made popular by open source

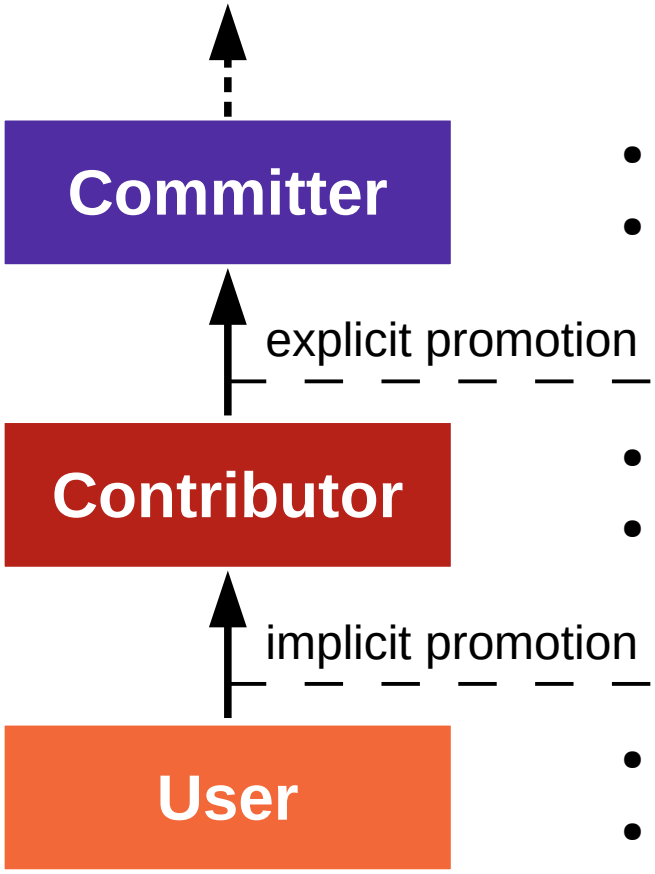
The 3-4-5 Model of Open Projects (Recap)

- Three principles of open collaboration
  - Egalitarian: Everyone may join and contribute
  - Meritocratic: Decisions are based on the merits of the argument
  - Self-organizing: Projects choose their own process
- Four practices of open communication
  - Communication in open source projects is open, that is, is public, written, complete and archived
- Five stages of project volunteering
  - Engagement proceeds through the finding, understanding, engaging, contributing, and leading stages

Contributions to Open Source Projects

- **Source code contributions**
- Bug reports using issue trackers
- Forum posts and answers
- Conference presentations, etc.

Roles in Open Source Projects (Recap)



- Has commit (write) rights
- Performs bulk of the work, patch review

- Provides small features, bug fixes
- Submits patches (no commit right)

- Knows and uses software
- If so, helps with comments, feedback

Source Code Contribution Process

1. Planning of contribution by contributor
2. Creation of original source code by contributor
3. Submission for inclusion by contributor
4. Review with possible back-and-forth by committer
5. Commit to code repository by committer

1. Planning of Contribution by Contributor

- A developer in the contributor role suggests a feature
  - Sometimes, committers will prod contributors for features
- The smart contributor lays out what needs to be done
  - Includes angle of attack, design issues and solutions
  - Perhaps provides time-line, need for coordination
  - Seeks, accepts, and works with project feedback

2. Creation of Contribution by Contributor

- The contributor develops the code

3. Submission of Contribution by Contributor

- The contributor submits the code for consideration
  - There are two common forms of submitting code:
    - Patch submission by email (the original method, still widely used)
    - Pull request using distributed version control systems

4. Review of Contribution by Committer

- A developer in the committer role reviews the code
  - If the contribution lacking, a back-and-forth ensues
    - If unanswered, the contribution might be dropped

5. Commit of Contribution by Committer

- The committer commits the source code

Patch Submission using `diff` and `patch`

- Contributor “diffs” their working copy with latest relevant version
- Submits diff output to committer by email or through bug tracker
- Committer uses “patch” to apply contribution to working copy
- Committer reviews working copy for committing to project
- Committer commits patched version of their working copy

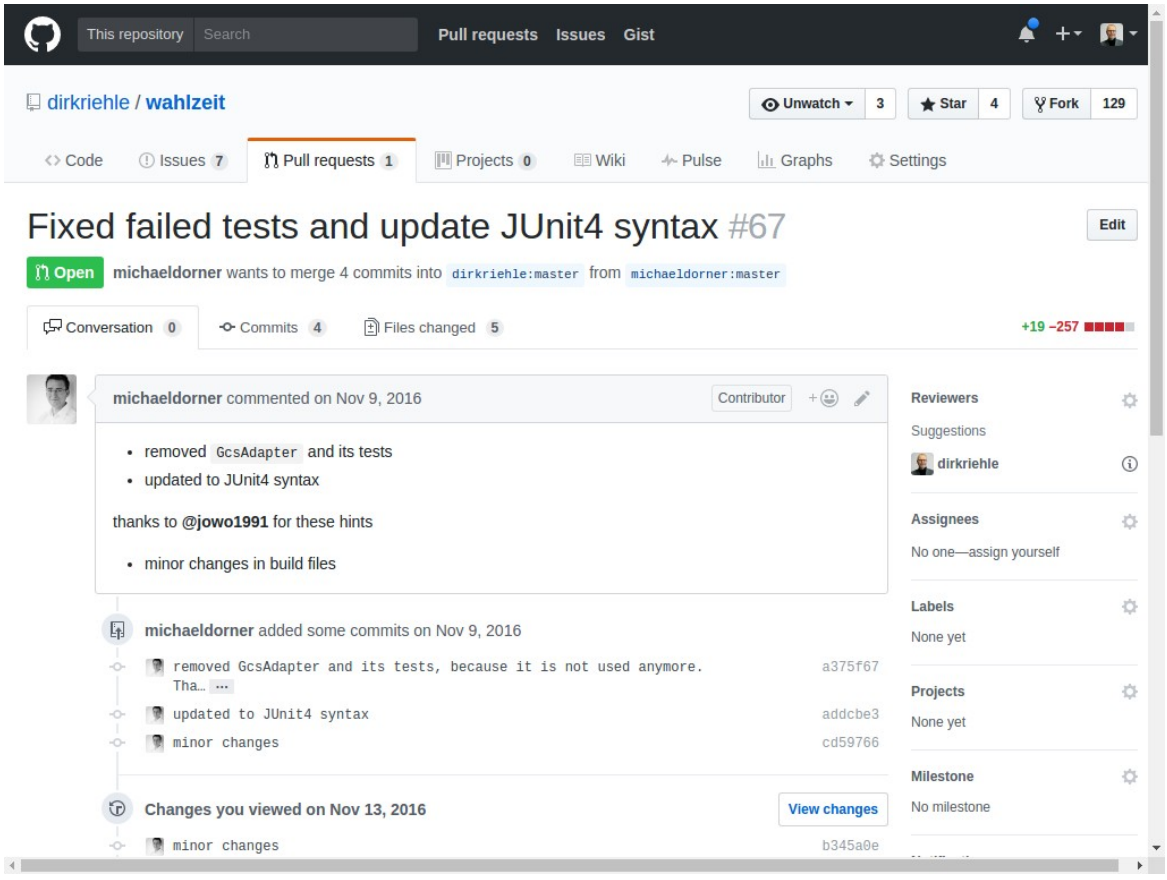
A Simple Diff / Patch Example

```
dirk@menlopark:~/workspace/jvalue/src/org/jvalue/names/values$
diff AbstractName.java ~/release/.../AbstractName.java > patchfile
```

```
219,225d218
<  protected void assertIsValidIndex(int i) throws Exception {
<     assertIsValidIndex(i, getNoComponents());
<  }
<
<  /**
<   * @MethodType assertion
<   */
```

```
katja@paloalto:~/myprojects/jvalue/src/org/jvalue/names/values$
patch AbstractName.java < patchfile
```

Sending Merge Requests (Pull Requests)



Successful Contribution is Difficult

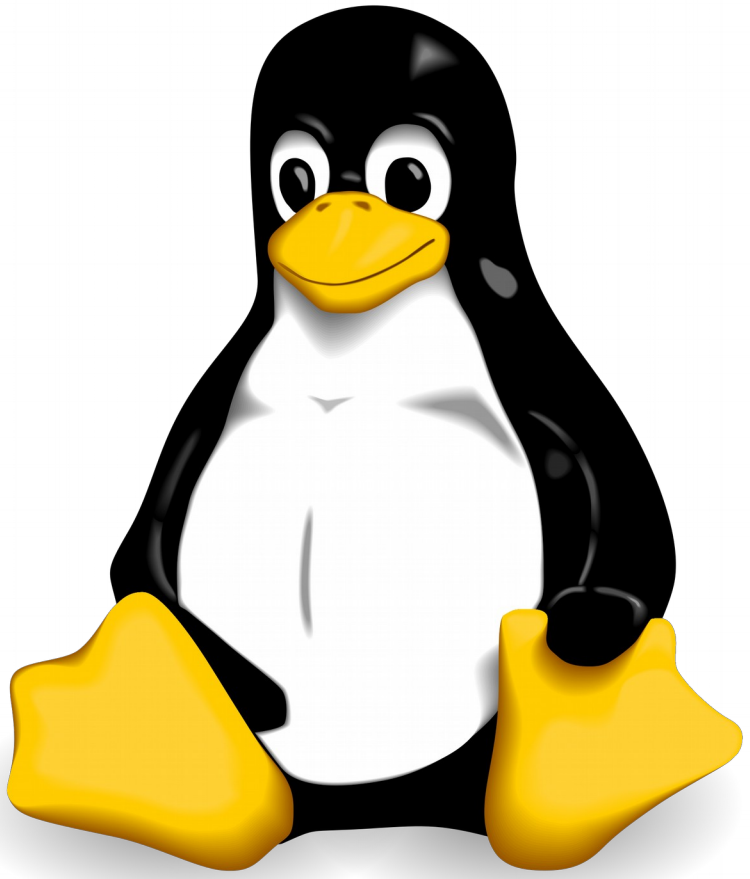
- Submissions by contributors often don’t make it
  - The not-so-smart contributor ...
    - Did not coordinate with the project in advance
    - Does not understand and comply with project practices
    - Misses the right time window
  - The committer might reject the submission because ...
    - The contributor does not follow-up properly upon a critique
    - The submission is too difficult to review (e.g. too large [W+08])
  - No committer has time to review the submission

Three (Very Different) Process Examples

- The Linux kernel
  - An operating system kernel
  - Hierarchical: “Linus and his trusted lieutenants”
- The PostgreSQL RDBMS
  - A relational database system
  - Core group of committers and evangelists
- The Tiki Wiki CMS Groupware
  - A web application platform
  - Anarchic: (Almost) everything goes

The Linux Kernel

- Is a Unix-derived / like operating system kernel
  - A kernel provides the core functions of an operating system
- Is the core of the GNU / Linux operating system
  - The operating system adds drivers, tools, etc. for a complete system
- Is broadly applicable, and widely used
  - As an embedded, mobile, desktop, and server operating system
- Is a unique free software project
  - Has more than 5 million lines of code and more than 1000 active contributors
  - Is GPLv2-licensed with caveats



Motivation for Company Engagement

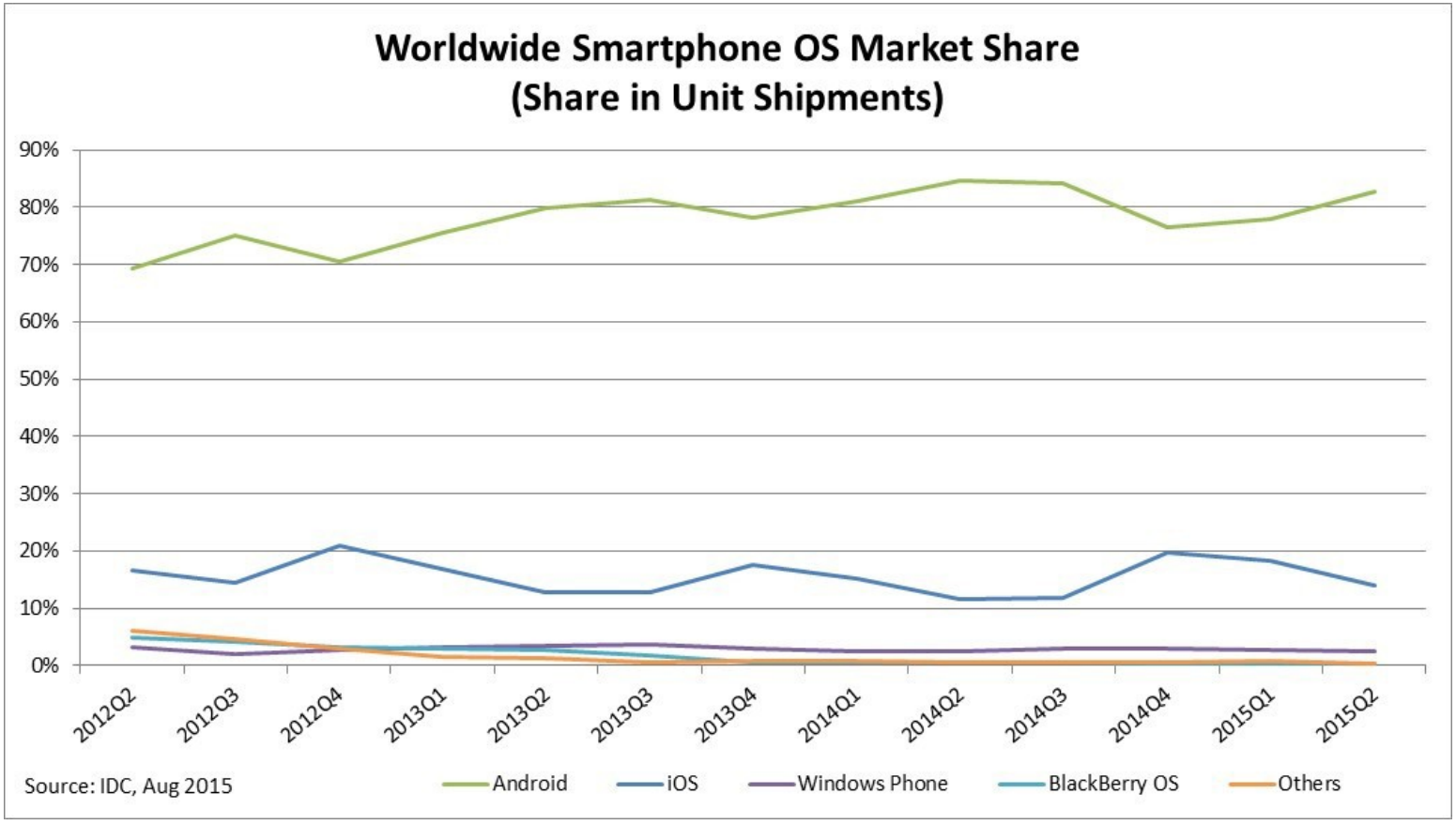
- Motivation to use or build on Linux and the Linux kernel
  - Keep proprietary operating systems at bay (cf. business models)
  - Be able to deliver products to a large market
- Motivation to contribute to Linux and the Linux kernel
  - Benefit from the open process, user feedback, improvements, etc.
  - Avoid maintenance costs for “out-of-tree” development
  - Gain credibility and position to influence future development

Development of the Linux Kernel [C09]

- Is supported and guided by the Linux Foundation (LF)
  - The LF employs Linus Torvalds, the founder and leader the Linux kernel
- Is solidly commercial (i.e. paid employees develop Linux)
- Is unique so that generalization is difficult / not appropriate
- The Linux kernel process calls committers “maintainers”

“At least 65% of the code which went into 2.6.20 was created by people working for companies.” [C07]

Growth of Linux (by way of Android) [1]



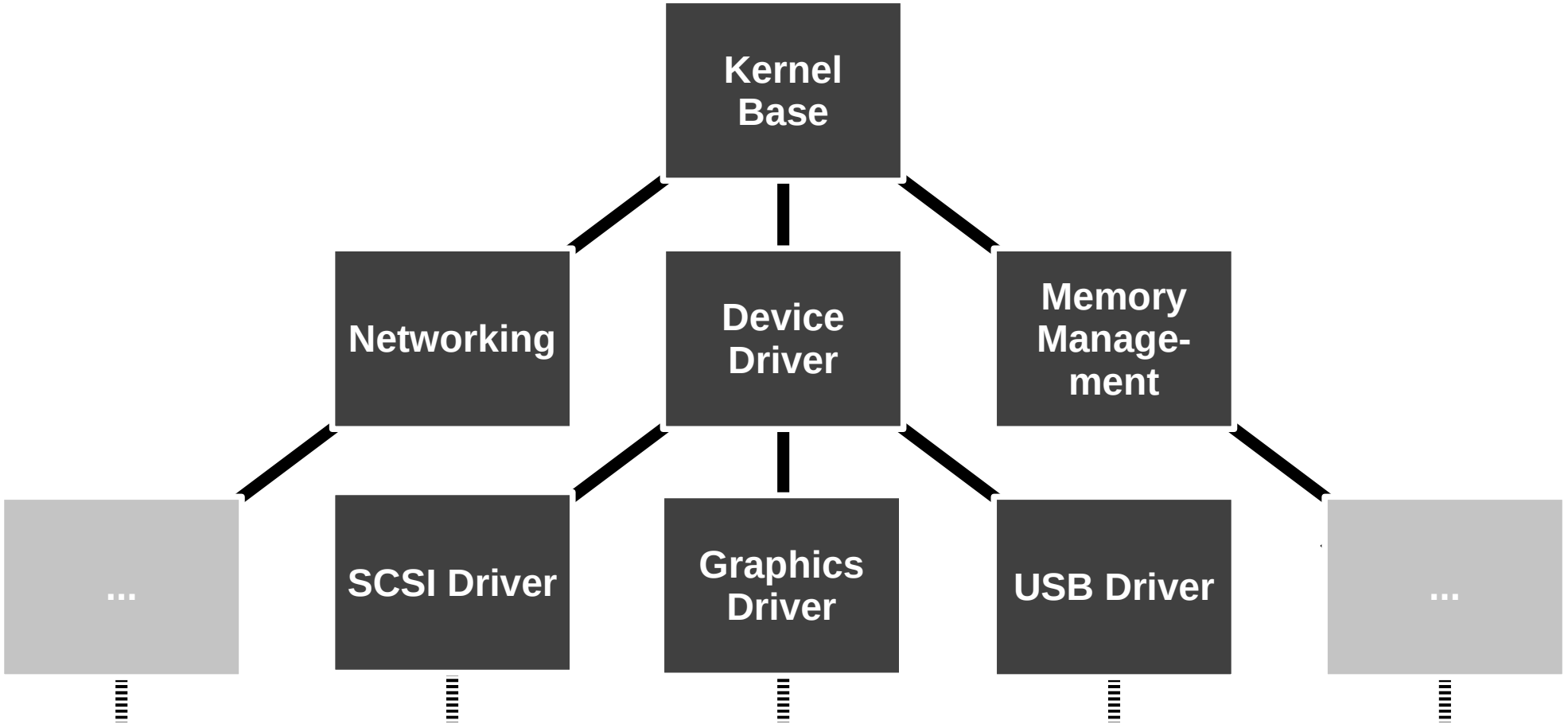
Linux Kernel: Product Management

- There is no centralized planning or product management
  - Individuals and companies decide on what to work on, for example,
    - Companies integrate device drivers
    - Individuals “scratch their own itch”
  - Corbet [C09] doesn’t even talk about it
- The Kernel bug tracker can serve as a starting point

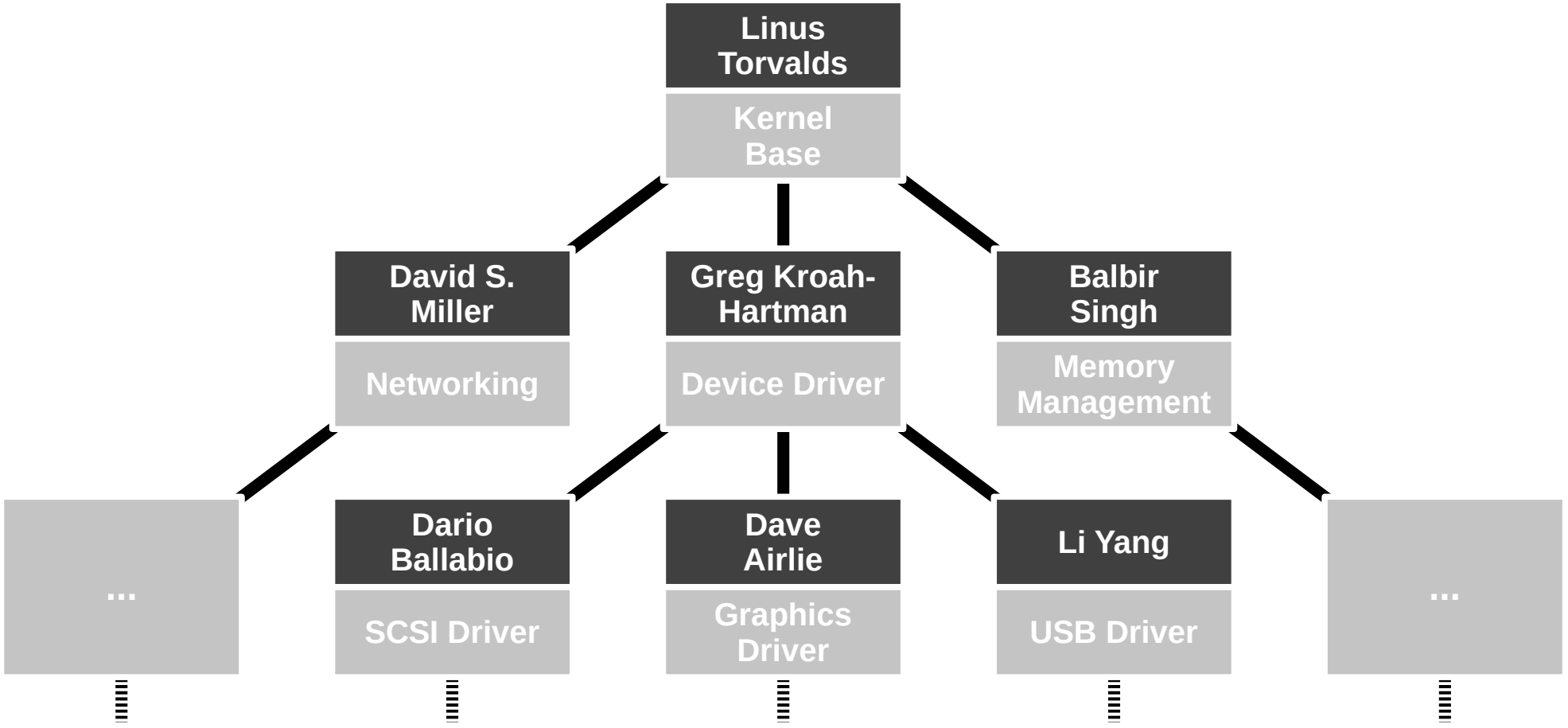
Linux Kernel: Engineering Management

- The Kernel development time-line
  - Is determined and managed by Linus Torvalds (with community)
- The Kernel has a major release every two – three months
  - Weeks 1 + 2 (merge window)
    - Stable code is merged into the mainline
    - First release candidate is declared
  - Weeks 3 – end
    - Fixes to problems with the release candidate are accepted
    - New features are rejected / postponed to next merge window
  - Final “stable release” is cut
- Code (“patches”) is stabilized outside the mainline
  - Any contribution (feature, bug fix, etc.) is called a patch

Linux Kernel: Code Architecture Illustration



Linux Kernel: Conway's Law Applied



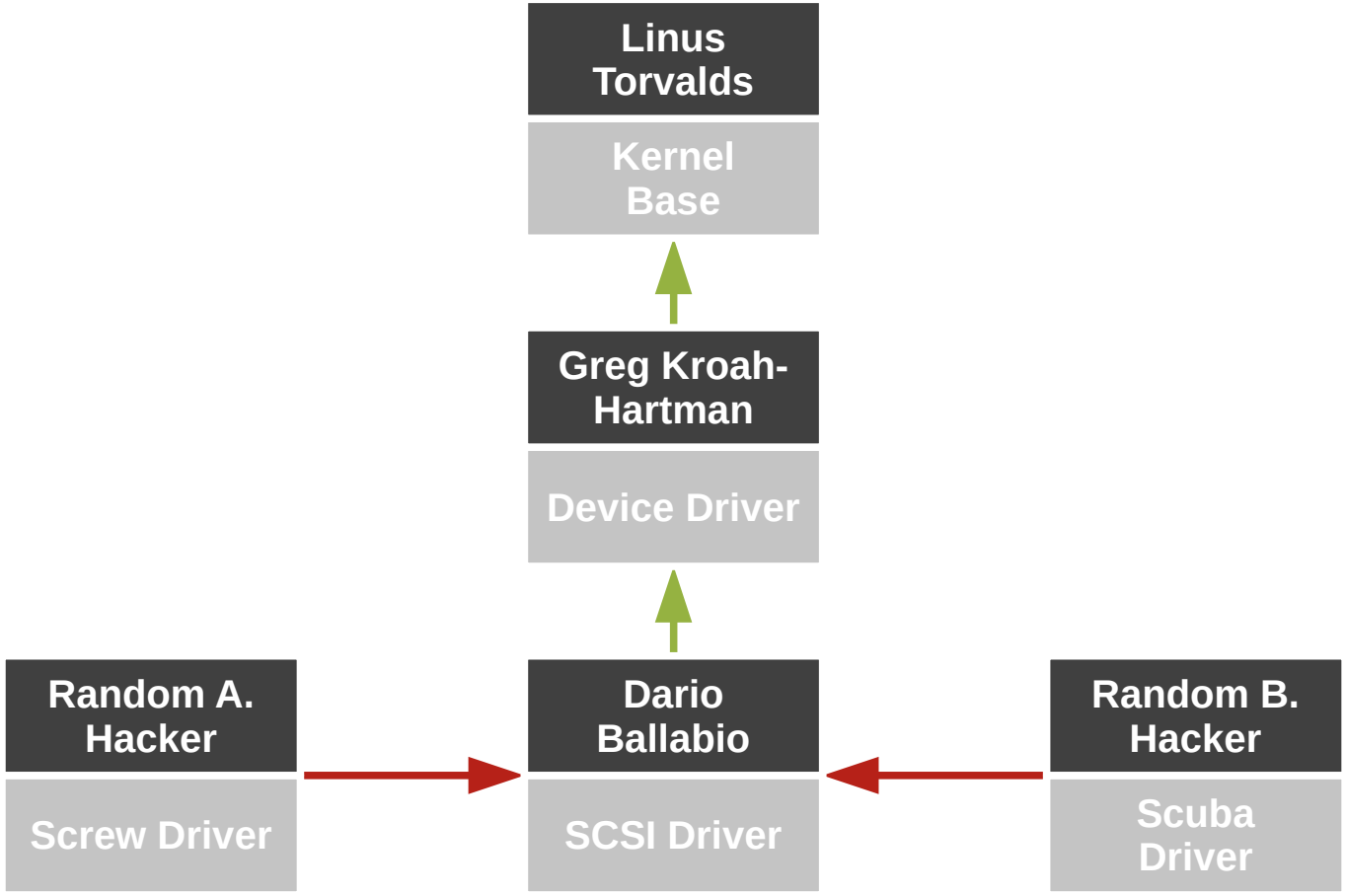
Linux Kernel: Patch-Flow 1 / 3

- Preparation of patches outside the mainline for integration
  - Design
    - Can be done in private, should better be done in public
  - Early review
    - Patch is posted to (relevant) mailing list, receives feedback
    - Feedback ideally unearths any serious problems with patch
    - Patch gets reworked until it gets close for mainline inclusion
  - Wider review
    - Patch is received by relevant subsystem maintainer
    - Maintainer applies patch to their (git) staging area
      - Exposes patch to even wider audience, leads to more feedback
      - Patch may not move on into the mainline during merge window

Linux Kernel: Patch-Flow 2 / 3

- Integration of patches into mainline during merge window
  - Top-level maintainers ask Torvalds to (git) pull their changes
    - Top-level maintainers themselves pulled from lower level maintainers
- Final integration decision at tree root by Torvalds
  - In practical terms, he relies on his “trusted lieutenants”
    - This “chain of trust” can go a few levels deep
  - Also implies Torvalds is merging more than programming

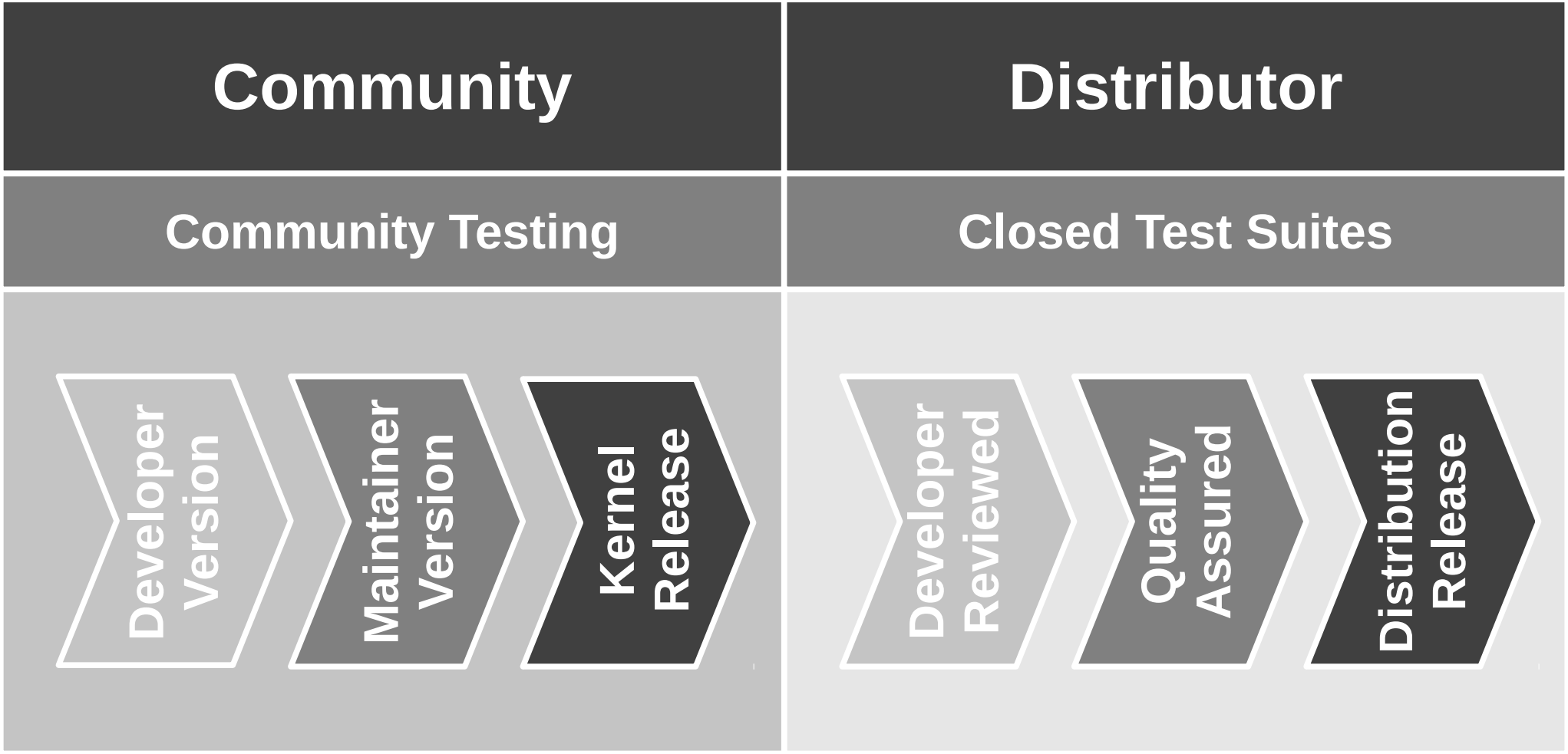
Linux Kernel: Patch-Flow 3 / 3



Linux Kernel: Software Development

- New code must be formatted according to coding guidelines
  - Reformatting old code to conform is considered “noise”
- Patch submission should follow several distinct rules
  - Break down major changes into a series of smaller patches
  - Make each patch semantically self-contained, reasonably small
  - Each patch must be able to stand on its now, not break the build
  - Describe each patch in the series following Kernel guidelines
- After patch submission, the review commences

Linux Kernel: Quality Assurance 1 / 2

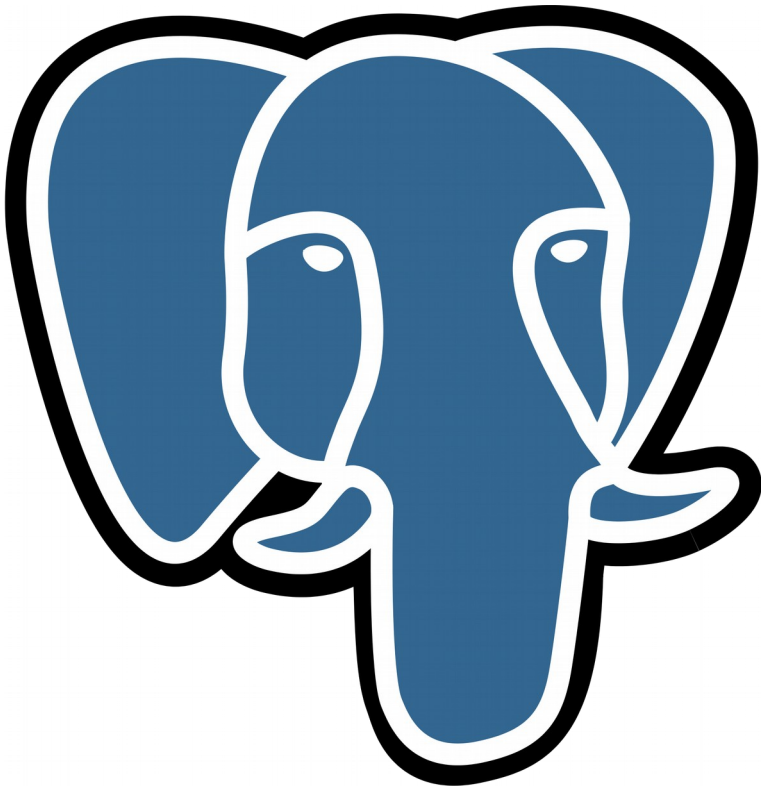


Linux Kernel: Quality Assurance 2 / 2

- Coding guidelines and good programming patterns
- Use of compiler warnings and code-checking tools
- Heuristics and advice on testing the kernel by hand

The PostgreSQL RDBMS

- Is “the most advanced” open source RDBMS
  - Standards-compliant
  - Enterprise-ready
  - Feature-rich
- Has 20 years of active development
  - Serves the needs of enterprises
  - Serves the needs of individuals
- Has broad community support
  - Companies contribute for commercial purposes
  - Individuals contribute to “scratch their itches”
- Is permissively licensed
  - PostgreSQL license, similar to BSD / MIT



PostgreSQL: Product Management [1]

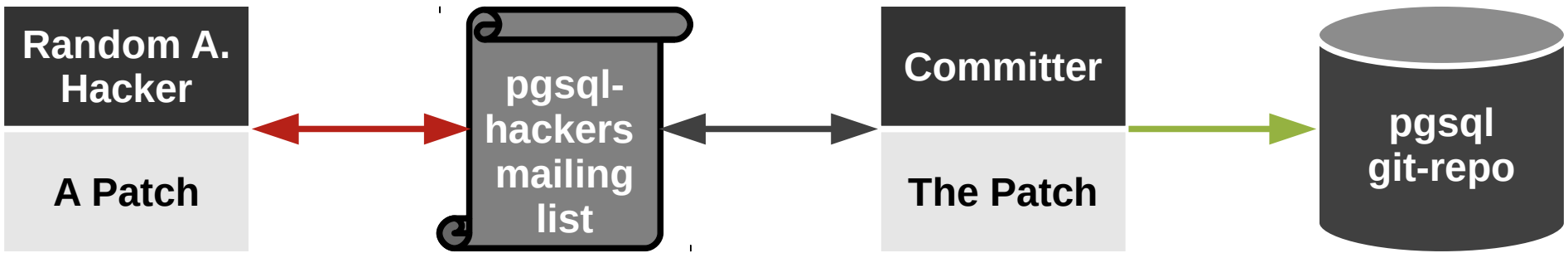
- Product roadmap is defined by core team [2]
- Product backlog is maintained as a to-do list [3]
- There is no prioritization: “Pick your feature”
- Community uses user polls for market research

PostgreSQL: Engineering Management

- Core team (long-term stable six members)
  - Coordinate release management activities
  - Facilitate consensus building process
  - Decide if community consensus fails
  - Administer major assets (website, repository)
- 20+ major contributors (committers)
  - Have commit rights, contribute to code base
- (Regular) contributors
  - Contribute to code base through committers
- “Hacker Emeritus” and past contributors

PostgreSQL: Patch-Flow

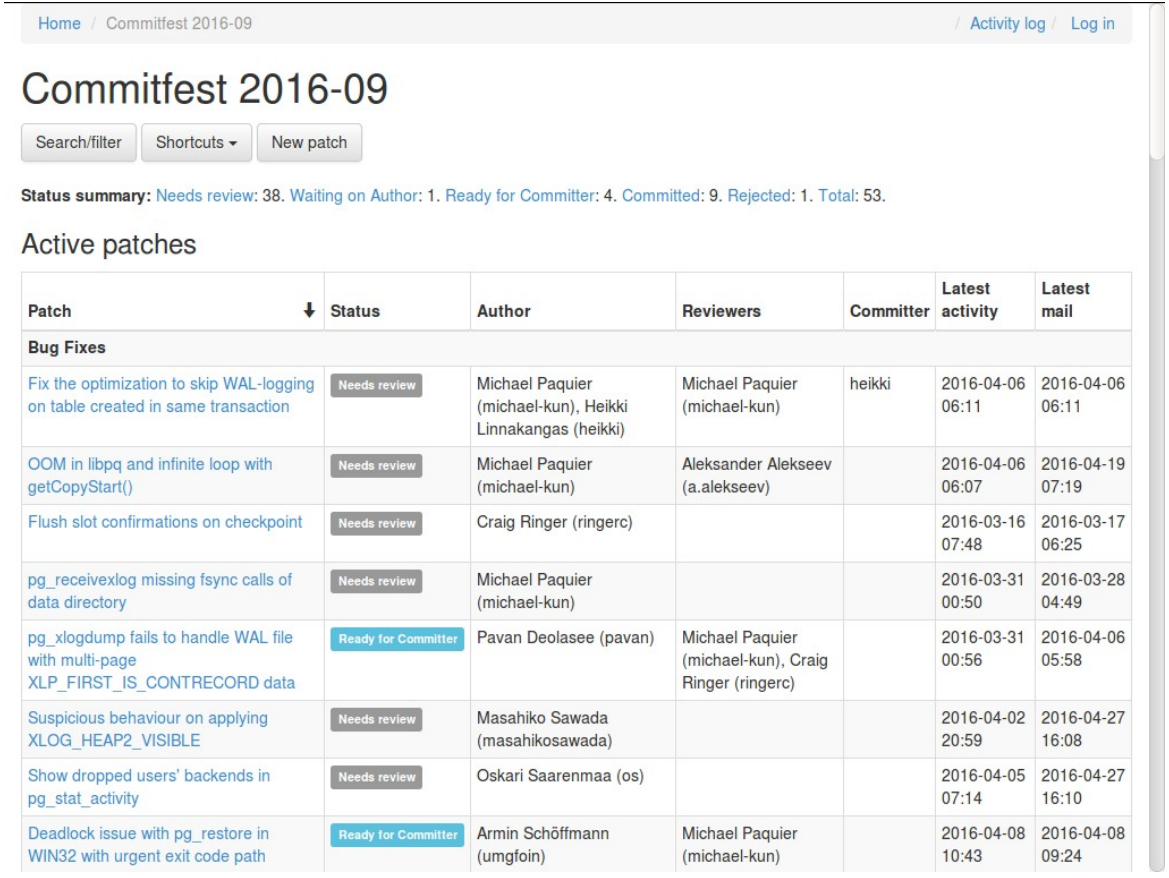
- Patch design
  - Can start in private, but is better done publicly
- Patch submission
  - Should always be submitted to the pgsql-hackers mailing list
- Patch review
  - Takes place publicly on mailing list
- Patch integration
  - A committer commits the patch to the git repository



PostgreSQL: Commitfest 1 / 2

- A commitfest is a period of intense patch review and integration
  - The purpose is to catch-up with the patch backlog
    - Old patches get stale, work is lost
  - Commitfests are held every other month and run for a month
    - Unless a pending release gets in between and postpones the commitfest
- Commitfest roles and activities
  - The commitfest manager selects patches for the patch queue
  - The commitfest manager assigns patches for review
  - The patches are discussed, until reviewers sign off
  - “Ready for committer” patches get picked-up, committed

PostgreSQL: Commitfest 2 / 2



Home / Commitfest 2016-09 / Activity log / Log in

Commitfest 2016-09

Search/filter | Shortcuts | New patch

Status summary: Needs review: 38. Waiting on Author: 1. Ready for Committer: 4. Committed: 9. Rejected: 1. Total: 53.

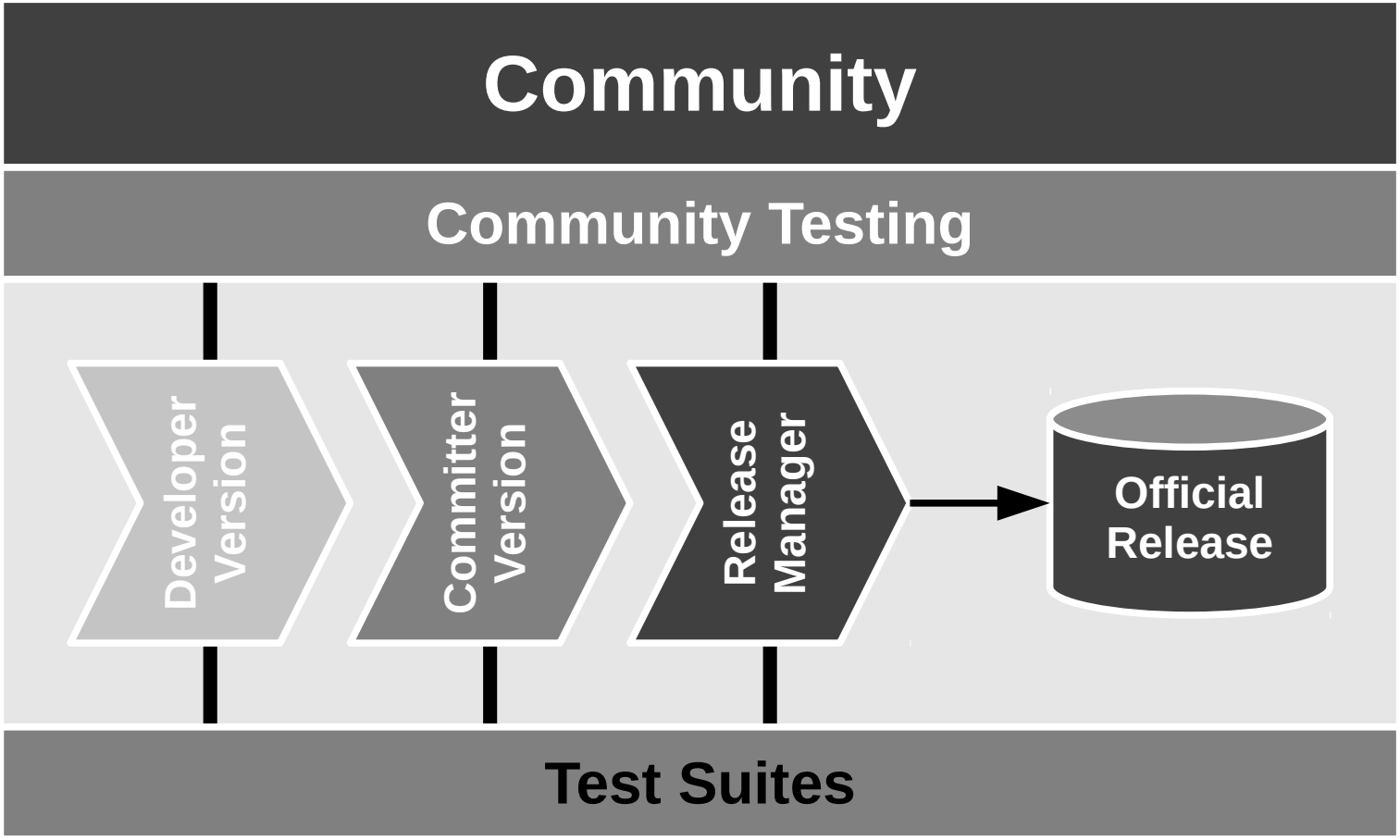
Active patches

Patch	Status	Author	Reviewers	Committer	Latest activity	Latest mail
Bug Fixes						
Fix the optimization to skip WAL-logging on table created in same transaction	Needs review	Michael Paquier (michael-kun), Heikki Linnakangas (heikki)	Michael Paquier (michael-kun)	heikki	2016-04-06 06:11	2016-04-06 06:11
OOM in libpq and infinite loop with getCopyStart()	Needs review	Michael Paquier (michael-kun)	Aleksander Alekseev (a.alekseev)		2016-04-06 06:07	2016-04-19 07:19
Flush slot confirmations on checkpoint	Needs review	Craig Ringer (ringerc)			2016-03-16 07:48	2016-03-17 06:25
pg_receivexlog missing fsync calls of data directory	Needs review	Michael Paquier (michael-kun)			2016-03-31 00:50	2016-03-28 04:49
pg_xlogdump fails to handle WAL file with multi-page XLP_FIRST_IS_CONTRECORD data	Ready for Committer	Pavan Deolasee (pavan)	Michael Paquier (michael-kun), Craig Ringer (ringerc)		2016-03-31 00:56	2016-04-06 05:58
Suspicious behaviour on applying XLOG_HEAP2_VISIBLE	Needs review	Masahiko Sawada (masahikosawada)			2016-04-02 20:59	2016-04-27 16:08
Show dropped users' backends in pg_stat_activity	Needs review	Oskari Saarenmaa (os)			2016-04-05 07:14	2016-04-27 16:10
Deadlock issue with pg_restore in WIN32 with urgent exit code path	Ready for Committer	Armin Schöffmann (umgfoin)	Michael Paquier (michael-kun)		2016-04-08 10:43	2016-04-08 09:24

PostgreSQL: Software Development

- Coding guidelines and good programming patterns
- Use of compiler warnings and code-checking tools

PostgreSQL: Quality Assurance 1 / 2



PostgreSQL: Quality Assurance 2 / 2

- Defined test suites
  - Use of regression test suites
  - Performance test suites using industry benchmarks
  - Use of closed or open source applications
- Provides testing tools
  - Isolation tests

PostgreSQL: Release Management

1. Commitfest
2. Alpha release
3. Beta release
4. Release

The Tiki Wiki CMS Groupware System

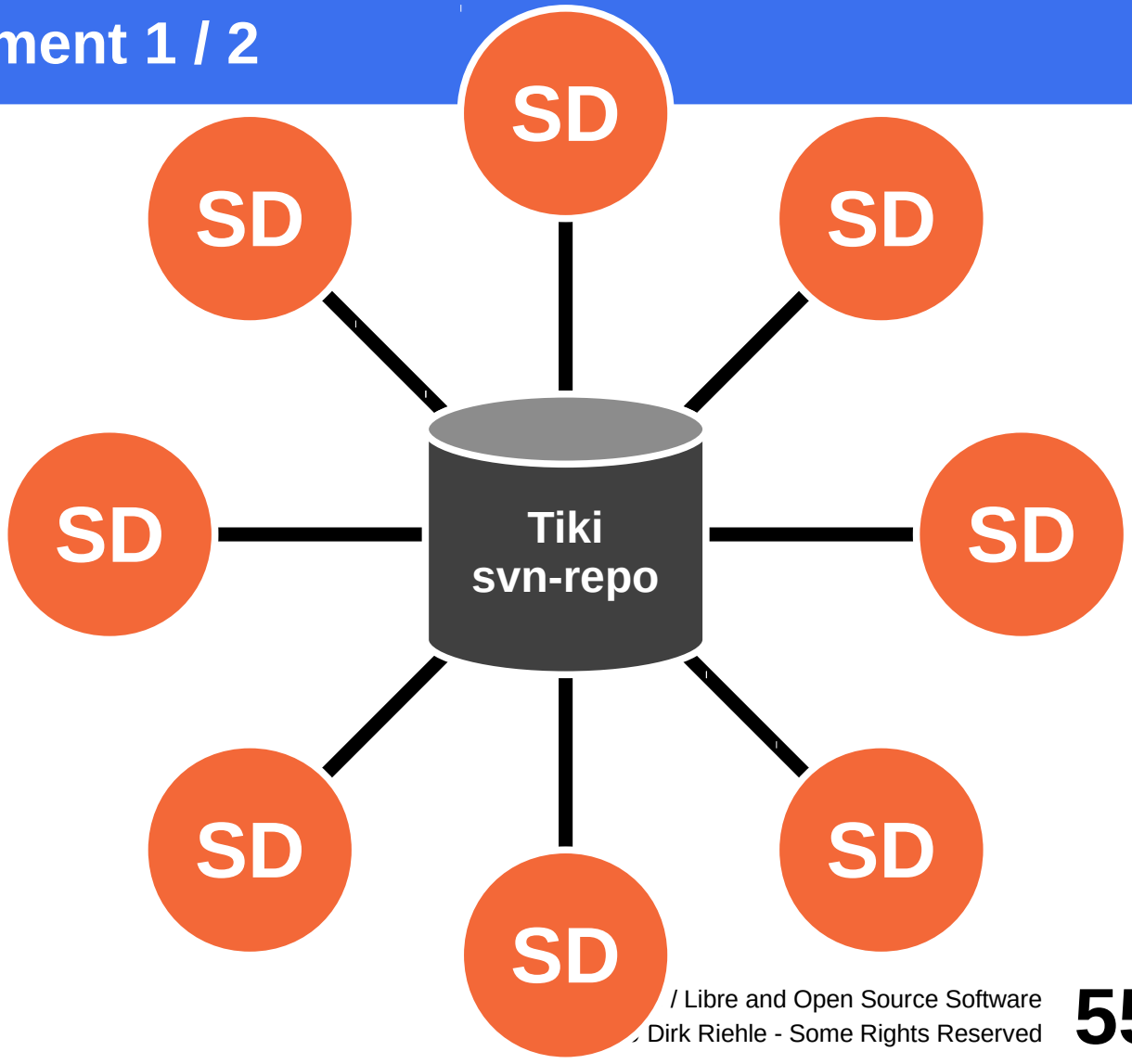
- A wiki, a CMS, a groupware system
  - Web application platform with the most built-in features
  - “Everything but the kitchen sink”
- In active development since 2002
  - With more than 250 contributors

Tiki: Product Management [E09]

- No central product management, but shared to-do list
  - Anyone can add a feature request
  - Anyone can pick a feature to work on

Tiki: Engineering Management 1 / 2

- Anyone who asks and seems like a reasonable person will get commit access to the main repository
- Contributor recruiting
  - “Recruit early, recruit often”
  - Smart, highly collaborative people
- Strong code ownership



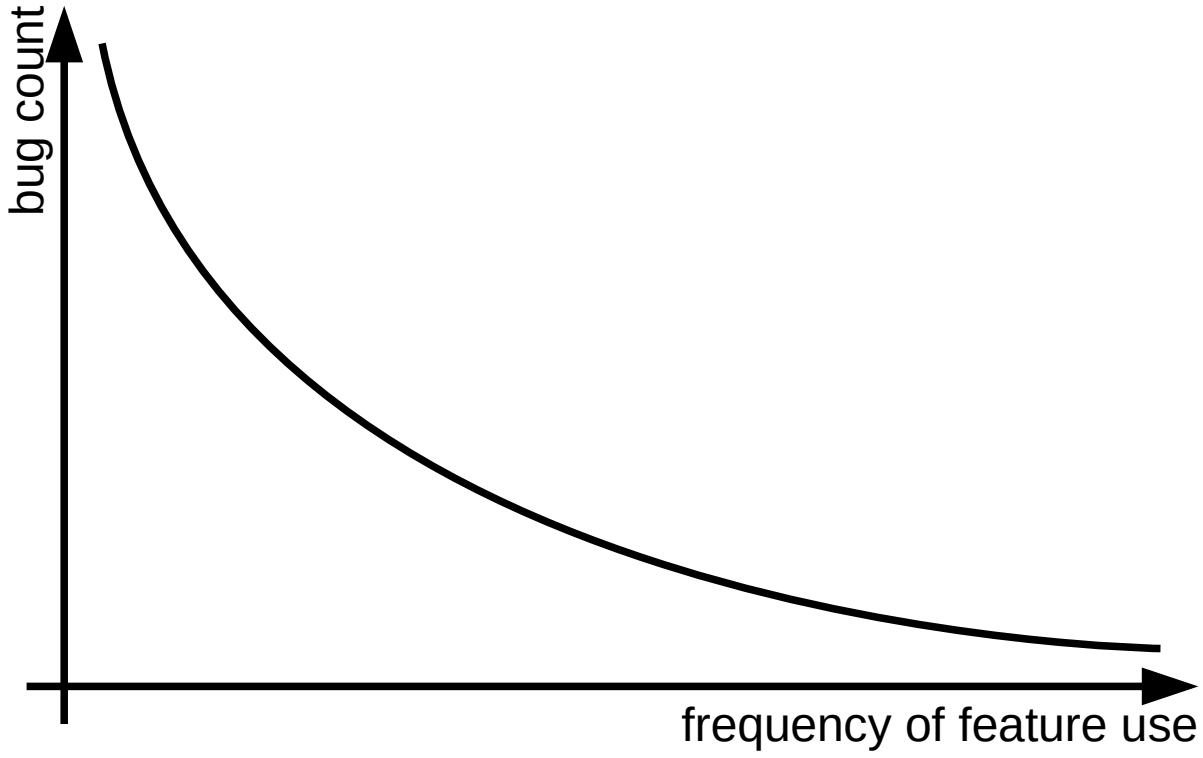
Tiki: Engineering Management 2 / 2

- Contributions have to follow these two rules
  - “Don’t break anything”
  - “Make it optional” (users can switch on/off features)
- “Commit early, commit often”

Tiki: Software Development

- Tiki has a monolithic software architecture
  - No complex plug-in architecture gets in the way
  - Contributors can directly change the code
  - Avoids “dependency hell”
- “If it ain’t broken, don’t fix it”

Tiki: Quality Assurance 1 / 2



Tiki: Quality Assurance 2 / 2

- Extensive community testing
  - “Given enough eyeballs, all bugs are shallow”

Review: Product Management

- No central control, no single responsible product manager
  - Companies contribute in line with their own interests
  - Individuals contribute to scratch their own itches
- If anything, a central document, jointly developed, but not binding
  - Open source is code-centric, bug tracker is important

Review: Engineering Management

- No general unifying process, only the 3-4-5 model applied

Review: Software Development

- Uses and enforces programming guidelines
- Has low tolerance for anything non-functional

Review: Quality Assurance

- Utilizes peer review for patch submission
- Relies strongly on community testing

Review / Summary of Session

- Position of open source projects
  - Are suppliers to their users
  - As such are products for a market
- Key roles and processes in open source
  - PM, EM, SD, and QA are quite different from closed development
  - A few unique innovations (patch review, distributed version control)
- Three example projects and their processes

Thank you! Questions?

dirk.riehle@fau.de – http://osr.cs.fau.de

dirk@riehle.org – http://dirkriehle.com – @dirkriehle

Credits and License

- Original version
  - © 2012-2019 Dirk Riehle, some rights reserved
  - Licensed under Creative Commons Attribution 4.0 International License
- Contributions
  - ...

Open Source Engineering Processes

Prof. Dr. Dirk Riehle

Friedrich-Alexander University Erlangen-Nürnberg

FOSS B05

It is Friedrich-Alexander University Erlangen-Nürnberg – FAU, in short.

Corporate identity wants us to say “Friedrich-Alexander University”.

Open Source Innovation

1. Legal innovation
2. **Process innovation**
3. Software tool innovation
4. Business model innovation

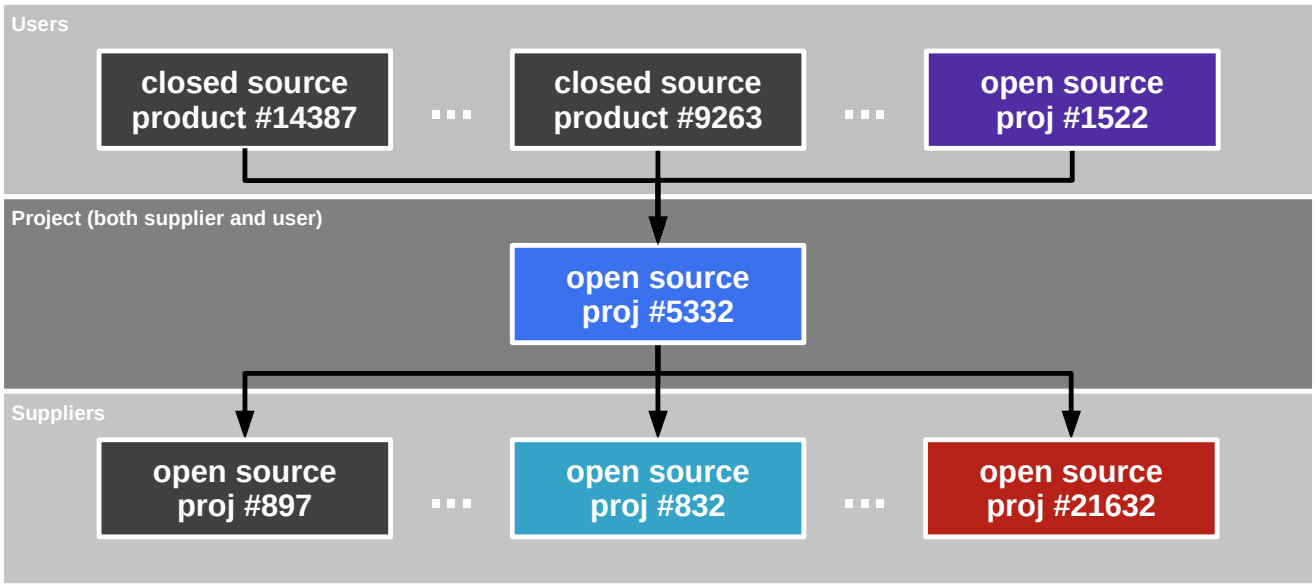
Open Source Process Innovation

- Community processes
- **Engineering processes**

Open Source Projects (Recap)

- Project community
  - People and companies engaged with the software
    - Includes using, developing, and marketing the software
- Software (components)
  - Not a product but a component (even if an application)
    - Users are therefore best advised to view project as a supplier
- Software users
  - Natural people scratch their own itch
  - Companies embedded software in products

The Software Supply Chain (Recap)

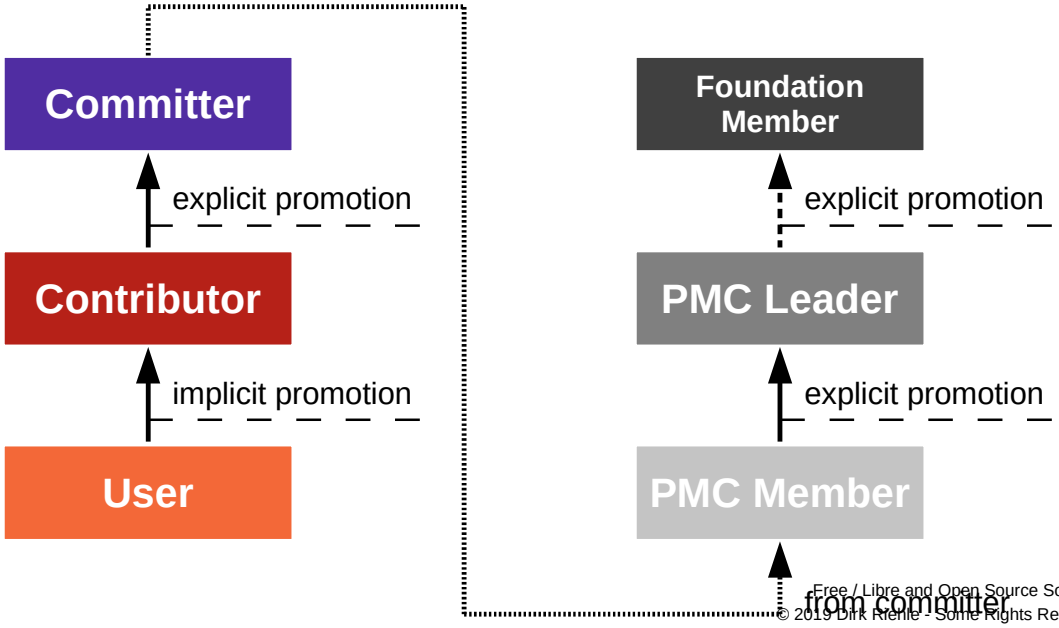


Key Roles in Software Engineering

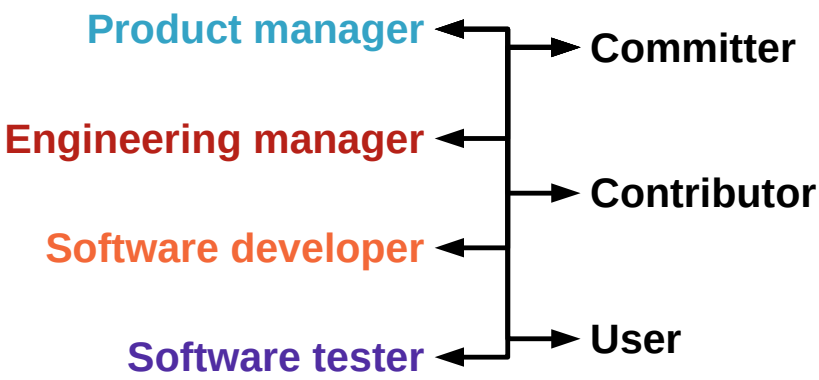
- Client-specific projects
  - Business analyst
  - Project manager
  - Software developer
  - Software tester / QA engineer
- **Products for a market**
  - **Product manager**
  - **Engineering manager**
  - **Software developer**
  - **Software tester / QA engineer**

Open source projects develop components for a market

Roles in Open Source Software Projects



Closed to Open Source Role Mapping



Product Management in Open Source

- Strategic product management
  - Does not take place in open source, is performed outside
- Technical product management
  - Product roadmapping
    - Some is performed but often is ad-hoc
  - Product specifications
    - Barely exist as documents (wikis, to-do lists, other)
  - Progress tracking
    - Managed by time, it is done when it is done

Engineering Management in Open Source

- Release planning
  - See product management: Some is performed but often is ad-hoc
- Resource allocation
  - Committers can prod contributors but that’s about it
  - Usually contributors pick up what they like to work on
- Process improvement
  - Ad-hoc, if any

Software Development in Open Source

- Programming
  - Like in closed source, but in general with less visibility as to completion

Quality Assurance in Open Source

- Code review
  - Is the core ingrained best practice followed by open source projects
- Automated testing
  - Like in closed source, though perhaps a bit more ad-hoc in general
- Manual testing
  - A lot of user testing, significantly more than in closed source
- Release management
  - Like in closed source, committers play release manager

Two Perspectives on the Discrepancy

- It is embarrassing for **open source**, because
  - It often does not know or apply the most basic best practices of product and engineering management
- It is embarrassing for **closed source**, because
  - All those best practices don’t seem to matter much: People figuring it out as they go along are equally or more effective
- The truth is probably somewhere in between
  - As so often, people matter more than processes and tools
  - Still, open source projects could learn a lot

Open Source Engineering Innovation

- There is no single process, only the one a community adopts
  - We can only talk about the 3-4-5 model of open communities
- Open source has a strong source code contribution process
  - A.k.a. peer review, patch submission, pull requests
- Distributed collaboration and version control
  - Developed and made popular by open source

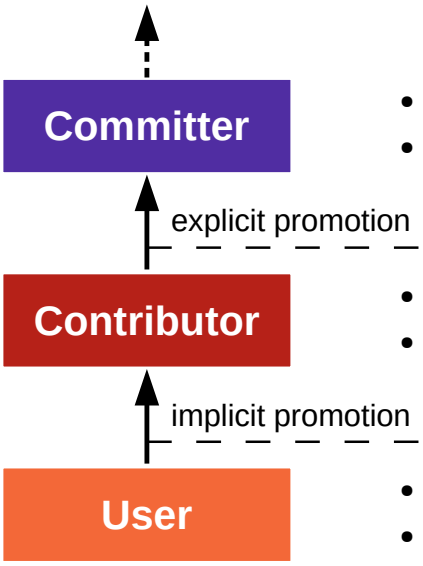
The 3-4-5 Model of Open Projects (Recap)

- Three principles of open collaboration
  - Egalitarian: Everyone may join and contribute
  - Meritocratic: Decisions are based on the merits of the argument
  - Self-organizing: Projects choose their own process
- Four practices of open communication
  - Communication in open source projects is open, that is, is public, written, complete and archived
- Five stages of project volunteering
  - Engagement proceeds through the finding, understanding, engaging, contributing, and leading stages

Contributions to Open Source Projects

- **Source code contributions**
- Bug reports using issue trackers
- Forum posts and answers
- Conference presentations, etc.

Roles in Open Source Projects (Recap)



- Has commit (write) rights
- Performs bulk of the work, patch review

- Provides small features, bug fixes
- Submits patches (no commit right)

- Knows and uses software
- If so, helps with comments, feedback

Source Code Contribution Process

1. Planning of contribution by contributor
2. Creation of original source code by contributor
3. Submission for inclusion by contributor
4. Review with possible back-and-forth by committer
5. Commit to code repository by committer

1. Planning of Contribution by Contributor

- A developer in the contributor role suggests a feature
  - Sometimes, committers will prod contributors for features
- The smart contributor lays out what needs to be done
  - Includes angle of attack, design issues and solutions
  - Perhaps provides time-line, need for coordination
  - Seeks, accepts, and works with project feedback

2. Creation of Contribution by Contributor

- The contributor develops the code

3. Submission of Contribution by Contributor

- The contributor submits the code for consideration
  - There are two common forms of submitting code:
    - Patch submission by email (the original method, still widely used)
    - Pull request using distributed version control systems

4. Review of Contribution by Committer

- A developer in the committer role reviews the code
  - If the contribution lacking, a back-and-forth ensues
    - If unanswered, the contribution might be dropped

5. Commit of Contribution by Committer

- The committer commits the source code

Patch Submission using `diff` and `patch`

- Contributor “diffs” their working copy with latest relevant version
- Submits diff output to committer by email or through bug tracker
- Committer uses “patch” to apply contribution to working copy
- Committer reviews working copy for committing to project
- Committer commits patched version of their working copy

A Simple Diff / Patch Example

```
dirk@menlopark:~/workspace/jvalue/src/org/jvalue/names/values$
diff AbstractName.java ~/release/.../AbstractName.java > patchfile
```

```
219,225d218
<  protected void assertIsValidIndex(int i) throws Exception {
<     assertIsValidIndex(i, getNoComponents());
<  }
<
<  /**
<   * @MethodType assertion
<   */
```

```
katja@paloalto:~/myprojects/jvalue/src/org/jvalue/names/values$
patch AbstractName.java < patchfile
```

Sending Merge Requests (Pull Requests)

Successful Contribution is Difficult

- Submissions by contributors often don’t make it
  - The not-so-smart contributor ...
    - Did not coordinate with the project in advance
    - Does not understand and comply with project practices
    - Misses the right time window
  - The committer might reject the submission because ...
    - The contributor does not follow-up properly upon a critique
    - The submission is too difficult to review (e.g. too large [W+08])
  - No committer has time to review the submission

Three (Very Different) Process Examples

- The Linux kernel
  - An operating system kernel
  - Hierarchical: “Linus and his trusted lieutenants”
- The PostgreSQL RDBMS
  - A relational database system
  - Core group of committers and evangelists
- The Tiki Wiki CMS Groupware
  - A web application platform
  - Anarchic: (Almost) everything goes

The Linux Kernel

- Is a Unix-derived / like operating system kernel
  - A kernel provides the core functions of an operating system
- Is the core of the GNU / Linux operating system
  - The operating system adds drivers, tools, etc. for a complete system
- Is broadly applicable, and widely used
  - As an embedded, mobile, desktop, and server operating system
- Is a unique free software project
  - Has more than 5 million lines of code and more than 1000 active contributors
  - Is GPLv2-licensed with caveats

Motivation for Company Engagement

- Motivation to use or build on Linux and the Linux kernel
  - Keep proprietary operating systems at bay (cf. business models)
  - Be able to deliver products to a large market
- Motivation to contribute to Linux and the Linux kernel
  - Benefit from the open process, user feedback, improvements, etc.
  - Avoid maintenance costs for “out-of-tree” development
  - Gain credibility and position to influence future development

Development of the Linux Kernel [C09]

- Is supported and guided by the Linux Foundation (LF)
  - The LF employs Linus Torvalds, the founder and leader the Linux kernel
- Is solidly commercial (i.e. paid employees develop Linux)
- Is unique so that generalization is difficult / not appropriate
- The Linux kernel process calls committers “maintainers”

“At least 65% of the code which went into 2.6.20 was created by people working for companies.” [C07]

Growth of Linux (by way of Android) [1]

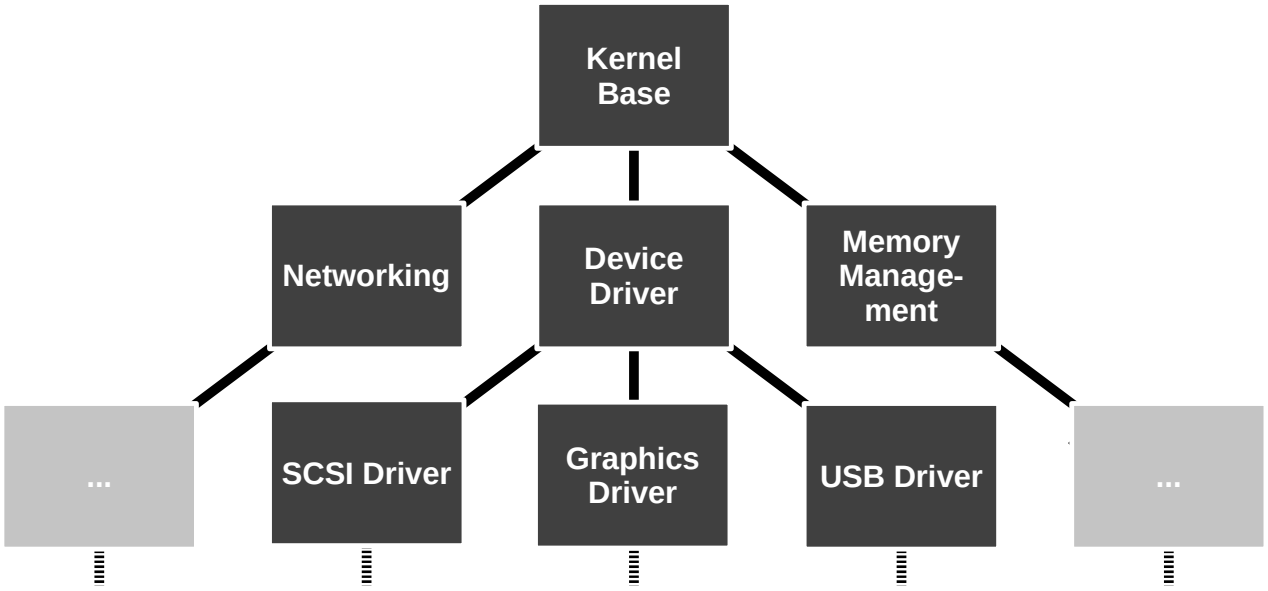
Linux Kernel: Product Management

- There is no centralized planning or product management
  - Individuals and companies decide on what to work on, for example,
    - Companies integrate device drivers
    - Individuals “scratch their own itch”
  - Corbet [C09] doesn’t even talk about it
- The Kernel bug tracker can serve as a starting point

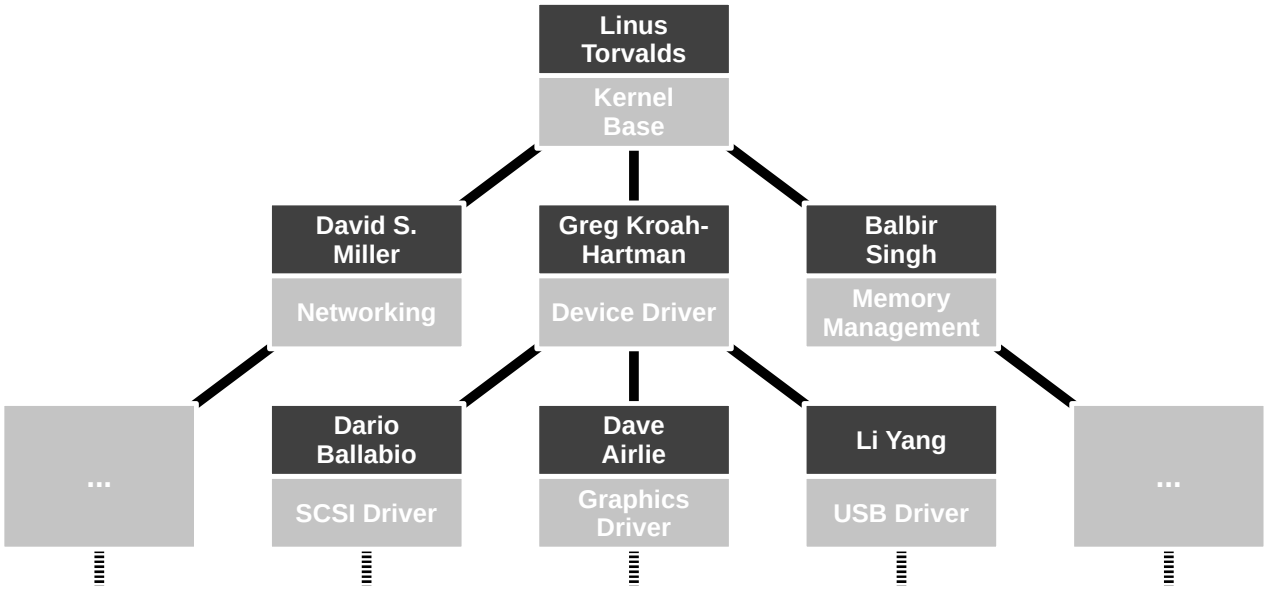
Linux Kernel: Engineering Management

- The Kernel development time-line
  - Is determined and managed by Linus Torvalds (with community)
- The Kernel has a major release every two – three months
  - Weeks 1 + 2 (merge window)
    - Stable code is merged into the mainline
    - First release candidate is declared
  - Weeks 3 – end
    - Fixes to problems with the release candidate are accepted
    - New features are rejected / postponed to next merge window
  - Final “stable release” is cut
- Code (“patches”) is stabilized outside the mainline
  - Any contribution (feature, bug fix, etc.) is called a patch

Linux Kernel: Code Architecture Illustration



Linux Kernel: Conway's Law Applied



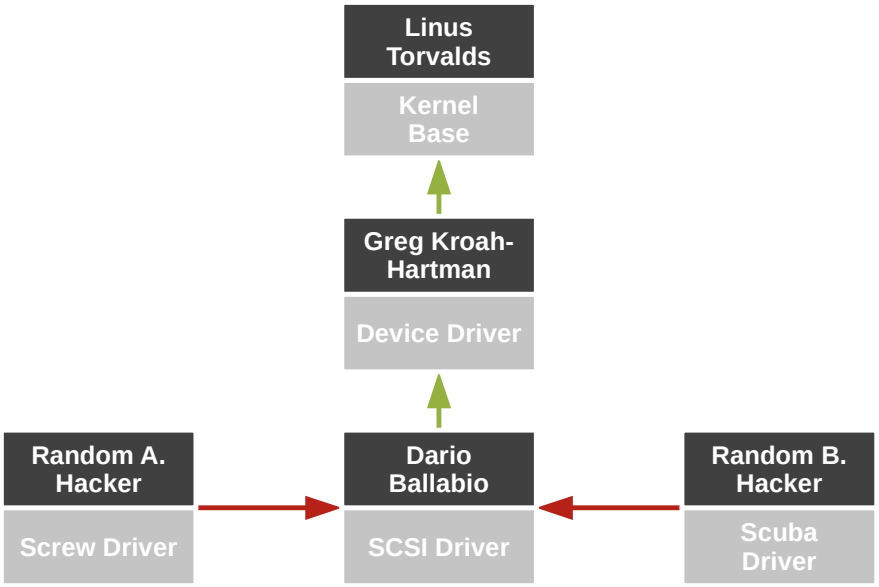
Linux Kernel: Patch-Flow 1 / 3

- Preparation of patches outside the mainline for integration
  - Design
    - Can be done in private, should better be done in public
  - Early review
    - Patch is posted to (relevant) mailing list, receives feedback
    - Feedback ideally unearths any serious problems with patch
    - Patch gets reworked until it gets close for mainline inclusion
  - Wider review
    - Patch is received by relevant subsystem maintainer
    - Maintainer applies patch to their (git) staging area
      - Exposes patch to even wider audience, leads to more feedback
      - Patch may not move on into the mainline during merge window

Linux Kernel: Patch-Flow 2 / 3

- Integration of patches into mainline during merge window
  - Top-level maintainers ask Torvalds to (git) pull their changes
    - Top-level maintainers themselves pulled from lower level maintainers
- Final integration decision at tree root by Torvalds
  - In practical terms, he relies on his “trusted lieutenants”
    - This “chain of trust” can go a few levels deep
  - Also implies Torvalds is merging more than programming

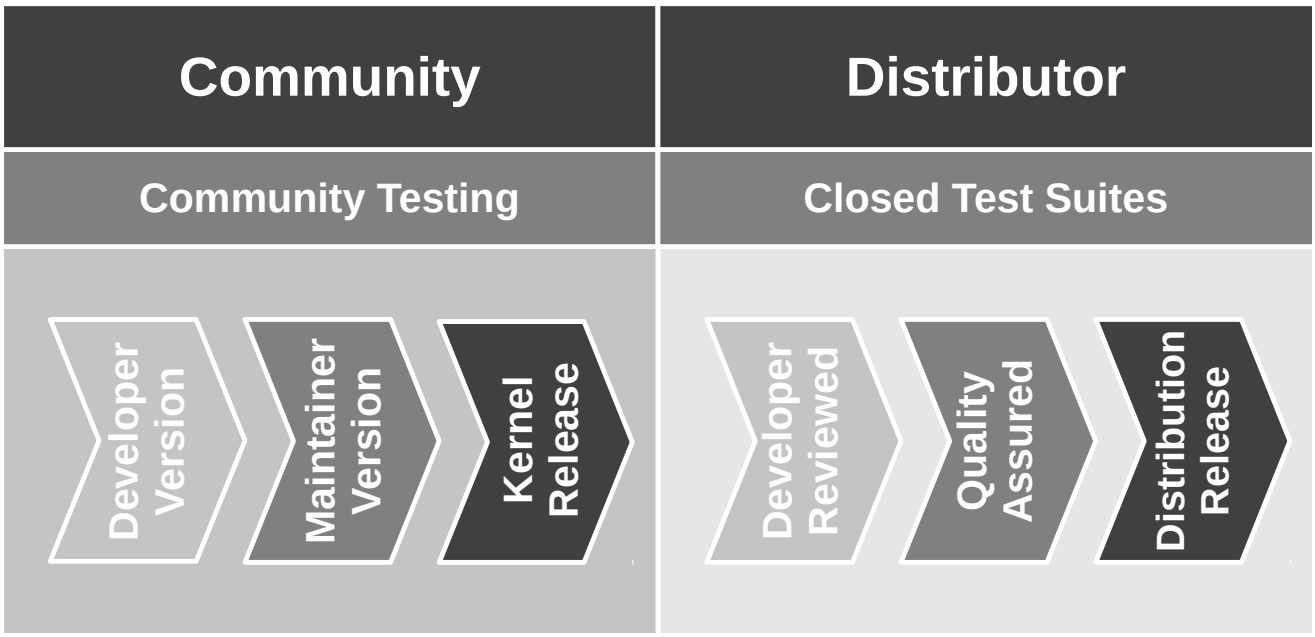
Linux Kernel: Patch-Flow 3 / 3



Linux Kernel: Software Development

- New code must be formatted according to coding guidelines
  - Reformatting old code to conform is considered “noise”
- Patch submission should follow several distinct rules
  - Break down major changes into a series of smaller patches
  - Make each patch semantically self-contained, reasonably small
  - Each patch must be able to stand on its now, not break the build
  - Describe each patch in the series following Kernel guidelines
- After patch submission, the review commences

Linux Kernel: Quality Assurance 1 / 2



Linux Kernel: Quality Assurance 2 / 2

- Coding guidelines and good programming patterns
- Use of compiler warnings and code-checking tools
- Heuristics and advice on testing the kernel by hand

The PostgreSQL RDBMS

- Is “the most advanced” open source RDBMS
  - Standards-compliant
  - Enterprise-ready
  - Feature-rich
- Has 20 years of active development
  - Serves the needs of enterprises
  - Serves the needs of individuals
- Has broad community support
  - Companies contribute for commercial purposes
  - Individuals contribute to “scratch their itches”
- Is permissively licensed
  - PostgreSQL license, similar to BSD / MIT

PostgreSQL: Product Management [1]

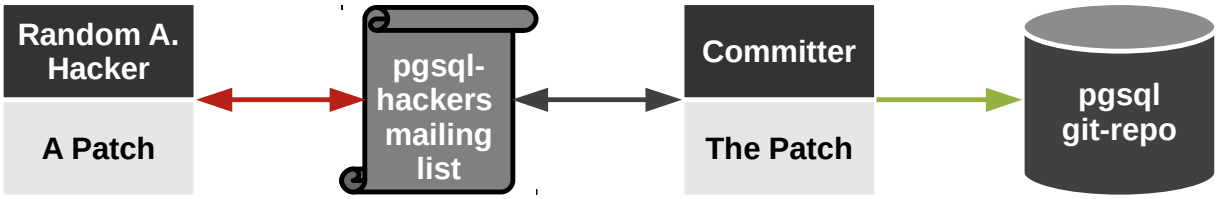
- Product roadmap is defined by core team [2]
- Product backlog is maintained as a to-do list [3]
- There is no prioritization: “Pick your feature”
- Community uses user polls for market research

PostgreSQL: Engineering Management

- Core team (long-term stable six members)
  - Coordinate release management activities
  - Facilitate consensus building process
  - Decide if community consensus fails
  - Administer major assets (website, repository)
- 20+ major contributors (committers)
  - Have commit rights, contribute to code base
- (Regular) contributors
  - Contribute to code base through committers
- “Hacker Emeritus” and past contributors

PostgreSQL: Patch-Flow

- Patch design
  - Can start in private, but is better done publicly
- Patch submission
  - Should always be submitted to the pgsql-hackers mailing list
- Patch review
  - Takes place publicly on mailing list
- Patch integration
  - A committer commits the patch to the git repository



PostgreSQL: Commitfest 1 / 2

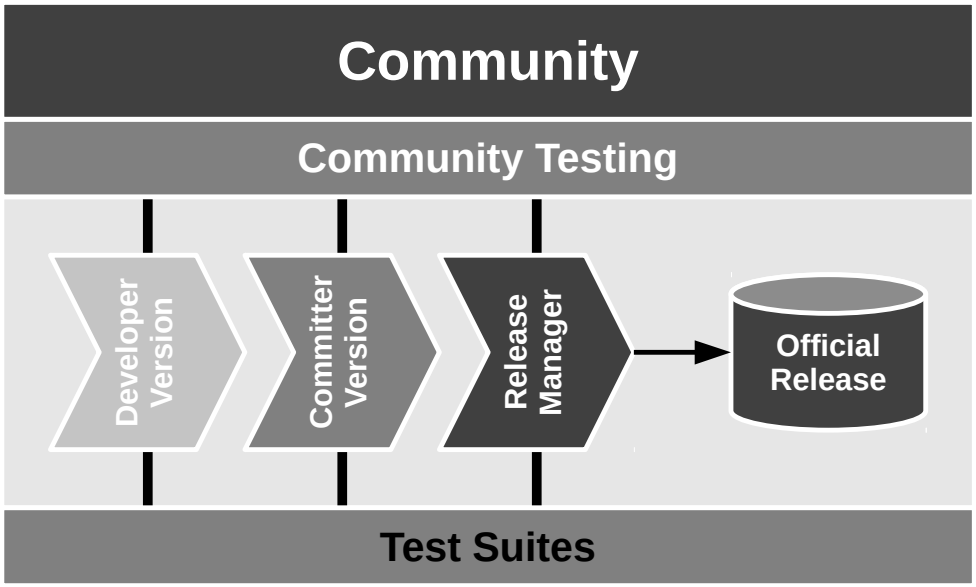
- A commitfest is a period of intense patch review and integration
  - The purpose is to catch-up with the patch backlog
    - Old patches get stale, work is lost
  - Commitfests are held every other month and run for a month
    - Unless a pending release gets in between and postpones the commitfest
- Commitfest roles and activities
  - The commitfest manager selects patches for the patch queue
  - The commitfest manager assigns patches for review
  - The patches are discussed, until reviewers sign off
  - “Ready for committer” patches get picked-up, committed

PostgreSQL: Commitfest 2 / 2

PostgreSQL: Software Development

- Coding guidelines and good programming patterns
- Use of compiler warnings and code-checking tools

PostgreSQL: Quality Assurance 1 / 2



PostgreSQL: Quality Assurance 2 / 2

- Defined test suites
  - Use of regression test suites
  - Performance test suites using industry benchmarks
  - Use of closed or open source applications
- Provides testing tools
  - Isolation tests

PostgreSQL: Release Management

1. Commitfest
2. Alpha release
3. Beta release
4. Release

The Tiki Wiki CMS Groupware System

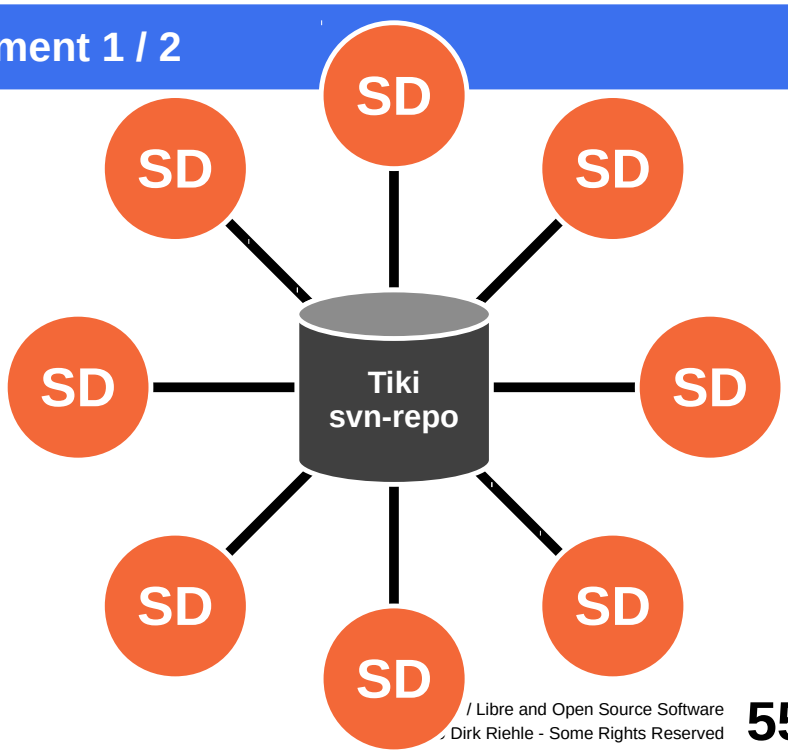
- A wiki, a CMS, a groupware system
  - Web application platform with the most built-in features
  - “Everything but the kitchen sink”
- In active development since 2002
  - With more than 250 contributors

Tiki: Product Management [E09]

- No central product management, but shared to-do list
  - Anyone can add a feature request
  - Anyone can pick a feature to work on

Tiki: Engineering Management 1 / 2

- Anyone who asks and seems like a reasonable person will get commit access to the main repository
- Contributor recruiting
  - “Recruit early, recruit often”
  - Smart, highly collaborative people
- Strong code ownership



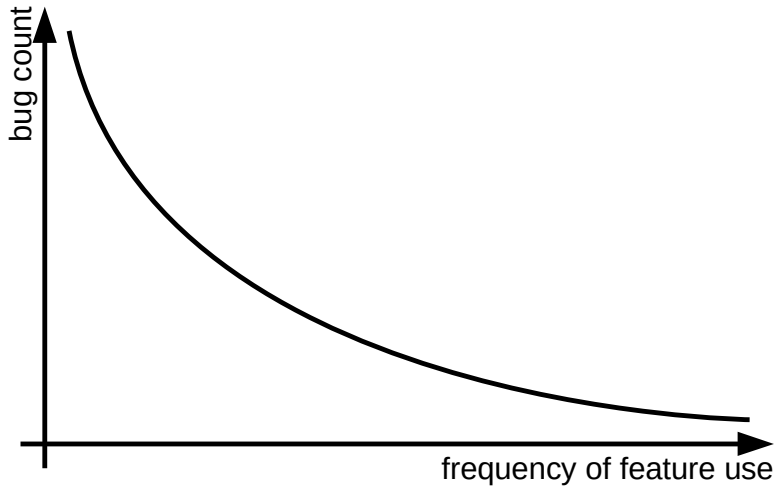
Tiki: Engineering Management 2 / 2

- Contributions have to follow these two rules
  - “Don’t break anything”
  - “Make it optional” (users can switch on/off features)
- “Commit early, commit often”

Tiki: Software Development

- Tiki has a monolithic software architecture
  - No complex plug-in architecture gets in the way
  - Contributors can directly change the code
  - Avoids “dependency hell”
- “If it ain’t broken, don’t fix it”

Tiki: Quality Assurance 1 / 2



Tiki: Quality Assurance 2 / 2

- Extensive community testing
  - “Given enough eyeballs, all bugs are shallow”

Review: Product Management

- No central control, no single responsible product manager
  - Companies contribute in line with their own interests
  - Individuals contribute to scratch their own itches
- If anything, a central document, jointly developed, but not binding
  - Open source is code-centric, bug tracker is important

Review: Engineering Management

- No general unifying process, only the 3-4-5 model applied

Review: Software Development

- Uses and enforces programming guidelines
- Has low tolerance for anything non-functional

Review: Quality Assurance

- Utilizes peer review for patch submission
- Relies strongly on community testing

Review / Summary of Session

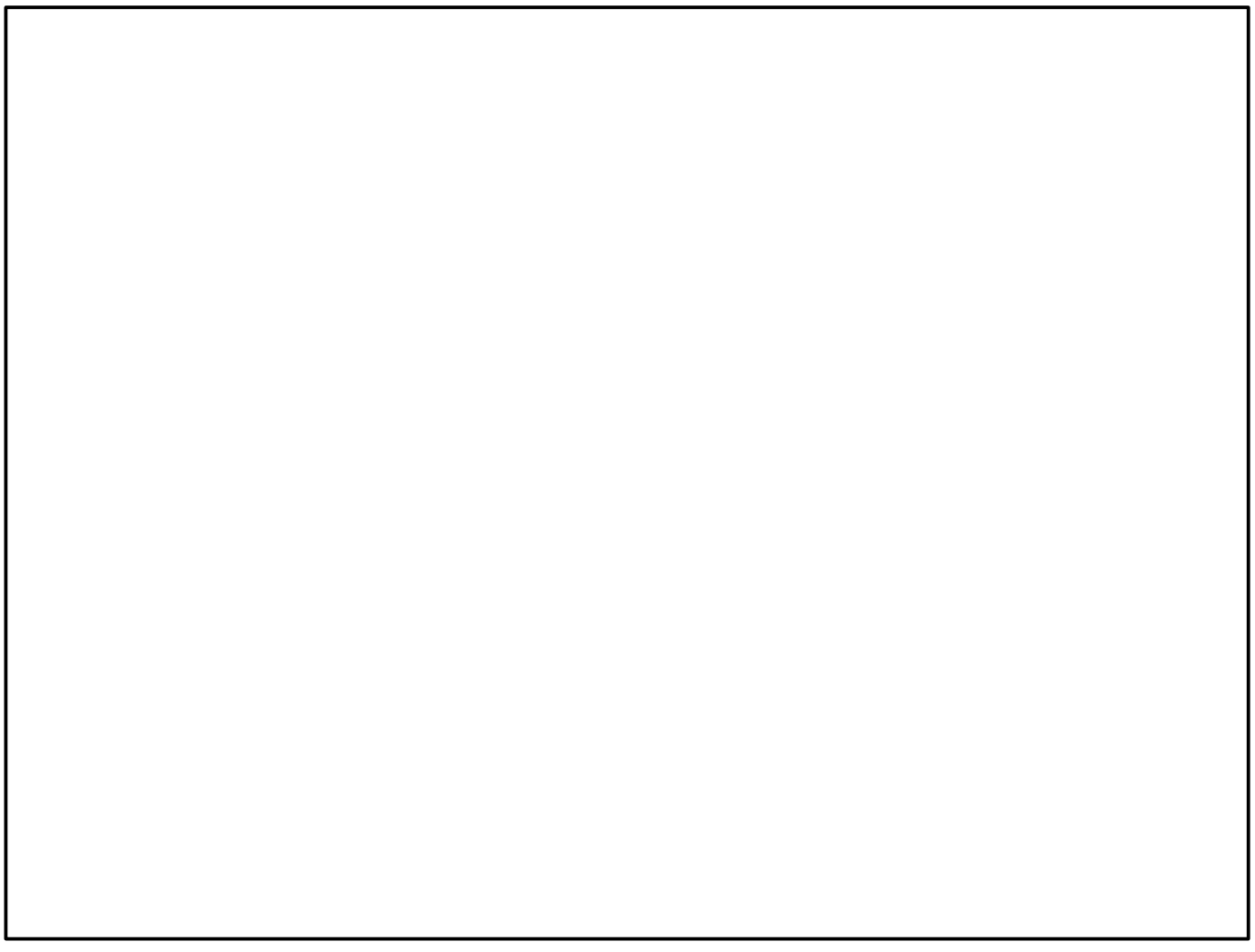
- Position of open source projects
  - Are suppliers to their users
  - As such are products for a market
- Key roles and processes in open source
  - PM, EM, SD, and QA are quite different from closed development
  - A few unique innovations (patch review, distributed version control)
- Three example projects and their processes

Thank you! Questions?

dirk.riehle@fau.de – http://osr.cs.fau.de

dirk@riehle.org – http://dirkriehle.com – @dirkriehle

DR



Credits and License

- Original version
  - © 2012-2019 Dirk Riehle, some rights reserved
  - Licensed under Creative Commons Attribution 4.0 International License
- Contributions
  - …