The Inner Source Revolution: How corporations create commercial software using open source methodologies

UniForum Chicago

The International Association of Open Systems Professionals

Vijay K. Gurbani, Ph.D.

Bell Laboratories, Nokia Networks

NOKIA

Monday, June 19, 2017

Joint work with ...

- James Herbsleb, Carnegie Mellon University.
- Anita Garvert, Lucent Technologies, Inc.
- Developers, testers, and many others who selflessly contributed to this work.

Upcoming Inner Source Events/Resources

- Inner Source Commons Summit
  - September 27-29, 2017, Naperville, Illinois (In Nokia building) (http://paypal.github.io/InnerSourceCommons/events/isc-fall-2017/)
- http://www.inner-sourcing.com/
  - Good repository of corporate interest in Inner Source.
- Linkedin InnerSource Commons group
  - https://www.linkedin.com/groups/4772921

References

[Gurbani 2010] Vijay K. Gurbani, Anita Garvert and James Herbsleb, “Managing a corporate open source software asset,” *Communications of the ACM*, 53(2), pp. 155-159, February 2010.

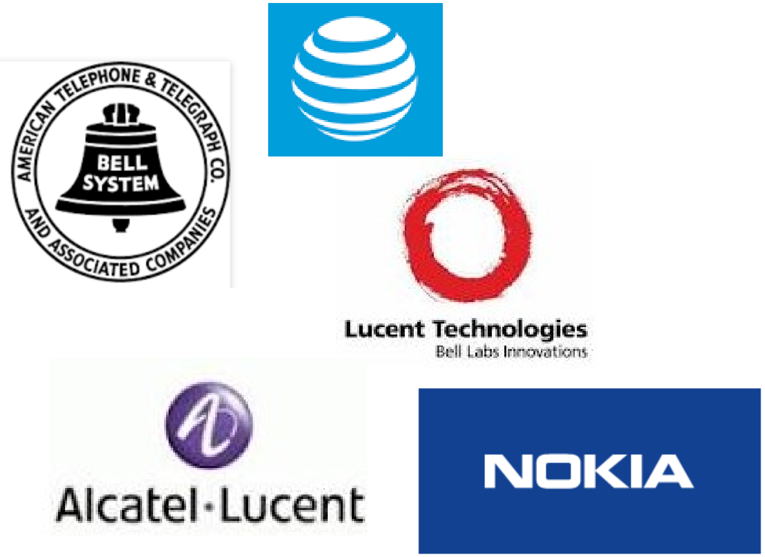
[Gurbani 2006] Gurbani, V.K., Garvert, A., and Herbsleb, J., "A Case Study of a Corporate Open Source Development Model," *Proceedings of the 28th ACM International Conference on Software Engineering (ICSE 2006)*, pp. 472-481, May 20-28, 2006.

[Gurbani 2005] Gurbani, V.K., Garvert, A., and Herbsleb, J., "A Case Study of Open Source Tools and Practices in a Commercial Setting," *Proceedings of the 5th ACM Workshop on Open Source Software Engineering,* pp. 24-29, May 17, 2005

[Stol 2014] Klaas-Jan Stol, et al., “Key factors for adopting Inner Source,” *ACM Transactions on Software Engineering and Methodology*, 23(2), pp. 18:1-18:35, March 2014.

[Capraro, 2017] Capraro, M. and Riehle, D., “Inner Source Definition, Benefits, and Challenges,” *ACM Computing Surveys*, 49(4), pp. 67:1–67:36, 2017.

About Nokia



Mobile Networks

Higher quality and more reliable mobile broadband experiences

Fixed Networks

More bandwidth in more places giving communities more access to the world

IP/Optical Networks

Massively scalable networks securely connecting everyone and everything to the Cloud

Applications & Analytics

Intelligent software platforms optimizing and automating network performance

Nokia Technologies

Connected health devices; professional Virtual Reality capture and broadcast; and highly valuable brand, intellectual property and technologies

Countries of operation

100+

Number of employees at the end of 2016

~101 000

R&D investment in 2016

EUR 4.9bn

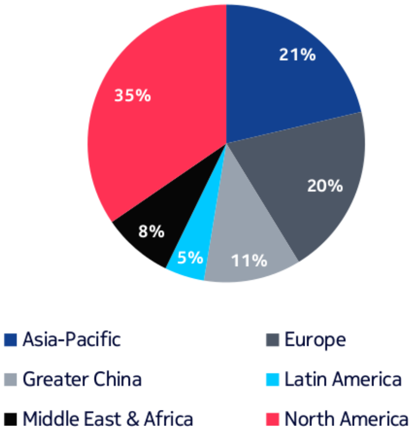
Net sales 2016

EUR 23.6bn

Nokia's Networks business

Net sales by geographic area

Q1 2017



A first definition of the problem

- Can corporations *benefit* from **developing** software using commonly accepted open source software development techniques?

- This is different than “do corporations *benefit* from **using** open source software?”

Inner Source

- [Stol 2014] (and others) define **Inner Source** as adoption of open source development practices within the confines of an organization.
  - The application of **best practices, processes, culture** and **methodologies** taken from open source world and applied to internal software development and innovation efforts [1].
  - No open source is being developed, but the firm's development process is enhanced through the addition of open source practices [2].
- [Gurbani 2005,2006,2010] terms this as **Corporate Open Source.**
- Key question: Can corporations benefit from the open source development methodology, or under what conditions can corporations adopt open source development methodology?

[1] Black Duck Software Inner Source Webinar Series: Community development practices in corporate IT. [Online https://www.blackducksoftware.com/consulting/inner-source]

[2] Dirk Riehle et al., “Open collaboration within corporations using software forges,” *IEEE Software, 26(2), 2009.*

Inner Source

- [Gurbani 2010] establishes the following taxonomy:
  - Infrastructure-based: common open source forge, but re-use is ad-hoc and support sporadic.
  - Project-specific: an owner of the shared asset chartered with developing, maintaining, supporting, and evangelizing the shared asset.
- My classification scheme is used as foundational work and is currently being refined by the Inner Source research community.
  - [Stol 2014] classifies Inner Source programs of 9 organizations using this model; Infrastructure-based is more prevalent .
  - [Capraro, 2017] develops a quantitative model of the elements that constitute Inner Source; applies that model to various Inner Source projects.
  - [??? 20??] Other works are in progress :-)

Inner Source

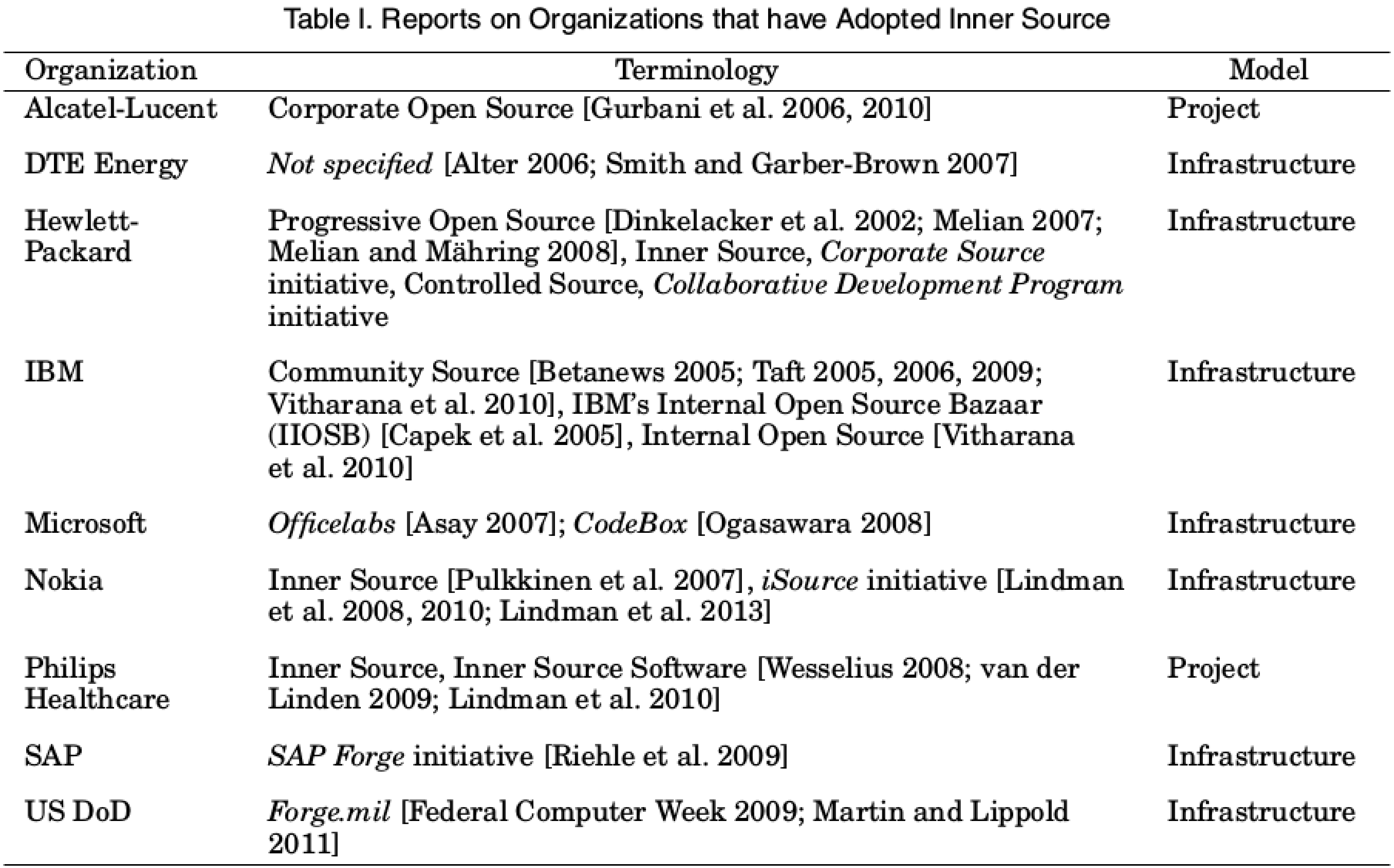


Table I. Reports on Organizations that have Adopted Inner Source

Organization	Terminology	Model
Alcatel-Lucent	Corporate Open Source [Gurbani et al. 2006, 2010]	Project
DTE Energy	*Not specified* [Alter 2006; Smith and Garber-Brown 2007]	Infrastructure
Hewlett-Packard	Progressive Open Source [Dinkelacker et al. 2002; Melian 2007; Melian and Mähring 2008], Inner Source, *Corporate Source* initiative, Controlled Source, *Collaborative Development Program* initiative	Infrastructure
IBM	Community Source [Betanews 2005; Taft 2005, 2006, 2009; Vitharana et al. 2010], IBM's Internal Open Source Bazaar (IIOSB) [Capek et al. 2005], Internal Open Source [Vitharana et al. 2010]	Infrastructure
Microsoft	*Officelabs* [Asay 2007]; *CodeBox* [Ogasawara 2008]	Infrastructure
Nokia	Inner Source [Pulkkinen et al. 2007], *iSource* initiative [Lindman et al. 2008, 2010; Lindman et al. 2013]	Infrastructure
Philips Healthcare	Inner Source, Inner Source Software [Wesselius 2008; van der Linden 2009; Lindman et al. 2010]	Project
SAP	*SAP Forge* initiative [Riehle et al. 2009]	Infrastructure
US DoD	*Forge.mil* [Federal Computer Week 2009; Martin and Lippold 2011]	Infrastructure

Table source [Stol 2014]

Open source methods in Nokia

- Nokia's Community of Sharing.
  - Designed to promote software reuse across business divisions.
  - Search engine for discovery.
  - Mostly an Infrastructure-based model.

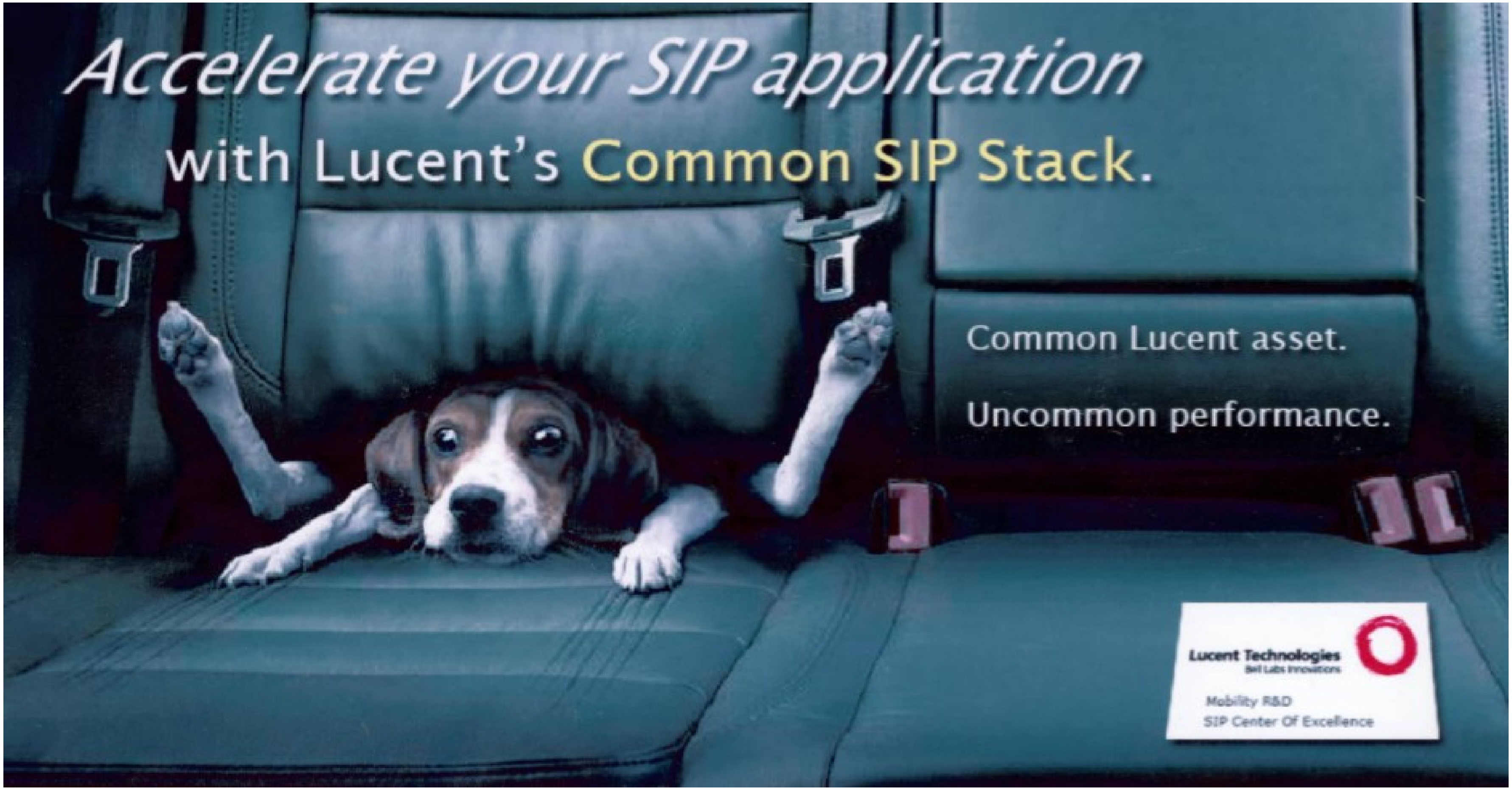
- Mobile Networks CTO has initiatives planned to help facilitate software reuse that leverages open source methods.
- MN CTO will guide and provide tools to facilitate the visibility and traceability of software components from internal repositories.
- MN CTO is defining and promoting best practices for “Inner Sourcing.”

Open source: a brief history in time

- Origins of open source software
  - 1950-1960: software sold with hardware, macros and utilities freely exchanged.
  - 1969 – Ken Thompson writes Unix, source code distributed to whoever asked (“Love, Ken”).
  - 1978 – Donald Knuth publishes TeX.
  - 1979 – AT&T commercializes Unix; BSD arrives.
  - 1983 – Richard Stallman and “GNU Manifesto”.
  - 1986 – Larry Wall releases PERL.
  - 1987 – Andrew Tannenbaum releases MINIX.
  - 1991 – Linus Torvalds releases Linux.
  - 1994 – Marc Ewing forms Red Hat.
  - 1995 – Apache.
  - 1998 – Netscape releases Mozilla source code.
  - 2000 – O'Reilly coins “Inner Sourcing”
  - 2001 – Eric Raymond, “The cathedral and the bazaar”
  - And …

Open source: a brief history in time

- ... and circa 2002 “The Lucent Common SIP Stack”



Graphic courtesy James Knauft, Alcatel-Lucent.

Open source versus traditional development

- Are open source development characteristics incompatible with traditional commercial development?
  - Requirements.
    - Traditional: Considerable time to gather and analyze requirements in an inter-disciplinary team (marketing, product management, software engineering).
    - Open source: Loose requirements, typical user may be a developer, change requests through mailing list, change request may or may not be acted on.

Open source versus traditional development

- Work assignments:
  - Traditional: Management-driven. Developers belong to an organization, and assigned by management on tasks. Usually effort to match skills and assignment, but developer choice generally limited.
  - Open source: Developer-driven. Starts with a perceived shortcoming in the software (“scratching an itch”). Strong contributors take larger role in the project.

Open source versus traditional development

- Software architecture:
  - Traditional: Monolithic, may be modular, but in the end it serves one master: the sponsoring department or organization.
  - Open source: Must be modular with especially well-defined interface points and APIs to support geographically distributed and ad-hoc contributors.

Open source versus traditional development

- Tool compatibility:
  - Traditional: Tools (source code control, debugging, compiling) are dictated by the specific organization or department.
    - clearcase, sccs
  - Open source: Much wider range of tools available to support the isolated software development model.
    - hg, git, svn, cvs.

Open source versus traditional development

- Software processes:
  - Traditional: Process-intensive with various evaluation points (may be easing lately).
  - Open source: Light to non-existent. Often control on whether the contributed source is accepted lies in a “benevolent dictator” or a small group of experts.

Open source versus traditional development

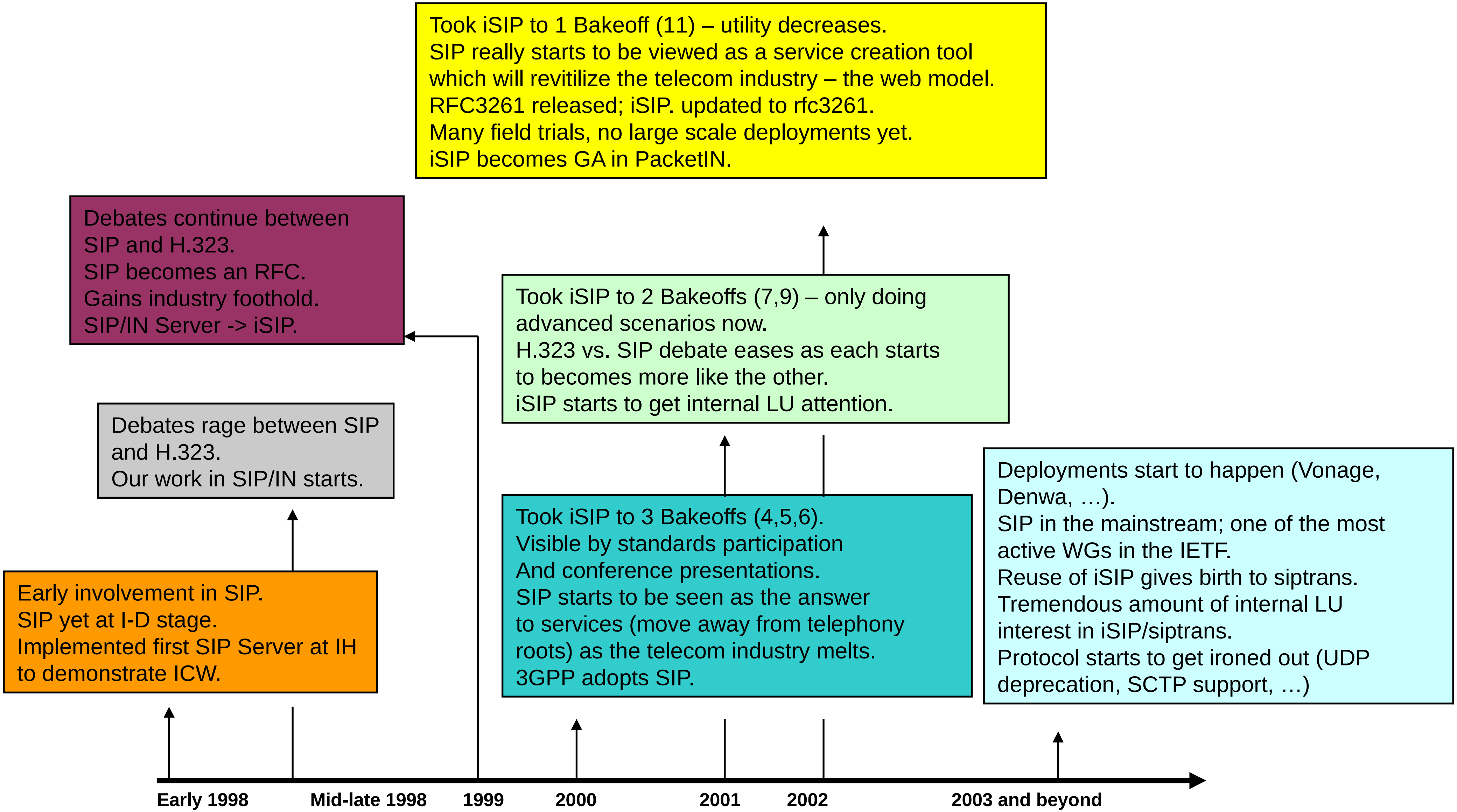
- Incentive structure:
  - Traditional: Profit-driven.
  - Open source: Driven by a more complex set of motives: desire to learn new skills, driven by creating features one needs, altruistic inclinations, etc. Money does NOT play a part in contributing to open source.

The project: A telecommunication signaling server

- SIP: Session Initiation Protocol
  - An multimedia session setup and teardown protocol.
    - Any type of sessions: voice, video, gaming, ...
  - March 1999: RFC 2543
  - August 2002: RFC 3261
  - Used in 3G, 4G, LTE, VoLTE, anywhere where service-provider control of signaling and media elements is/will be required.

The project: A telecommunication signaling server

Frozen in time: A slide from 2003 talk!



Early involvement in SIP.
SIP yet at I-D stage.
Implemented first SIP Server at IH to demonstrate ICW.

Debates rage between SIP and H.323.
Our work in SIP/IN starts.

Debates continue between SIP and H.323.
SIP becomes an RFC.
Gains industry foothold.
SIP/IN Server -> iSIP.

Took iSIP to 3 Bakeoffs (4,5,6).
Visible by standards participation
And conference presentations.
SIP starts to be seen as the answer to services (move away from telephony roots) as the telecom industry melts.
3GPP adopts SIP.

Took iSIP to 2 Bakeoffs (7,9) – only doing advanced scenarios now.
H.323 vs. SIP debate eases as each starts to becomes more like the other.
iSIP starts to get internal LU attention.

Took iSIP to 1 Bakeoff (11) – utility decreases.
SIP really starts to be viewed as a service creation tool which will revitilize the telecom industry – the web model.
RFC3261 released; iSIP. updated to rfc3261.
Many field trials, no large scale deployments yet.
iSIP becomes GA in PacketIN.

Deployments start to happen (Vonage, Denwa, …).
SIP in the mainstream; one of the most active WGs in the IETF.
Reuse of iSIP gives birth to siptrans.
Tremendous amount of internal LU interest in iSIP/siptrans.
Protocol starts to get ironed out (UDP deprecation, SCTP support, …)

The project: Establishing open source

- Timeline: 1998 – 2006.
- Phase 1: 1998 – 2000
  - Following early trajectory of SIP.
  - Closely working with IETF and in-house view on how SIP fits in the telecommunication ecosystem.
  - Code given to anyone (in the company) that asked.
  - Code taken to SIP bakeoffs.
  - Primary sponsor of the work was the host business unit.

The project: Establishing open source

- Phase 2: 2001 - 2004

Cycle 1: Opportunistic partnering.

- Asset primarily owned by one organization.
- Moved to being a *framework* used by other projects.

Graphic source: http://media-cdn.tripadvisor.com/media/photo-s/03/13/49/ee/egyptian-bazaar.jpg

Cycle 2: Branching out.

- User initiated change requests.
- More business units start to take interest in the asset.
- Requests started to arrive to evolve the server to a *platform*.

The project: Establishing open source

- Phase 3: 2004 – 2006.
  - Formal procedures in place to get contributions back.
  - “Benevolent dictator” (me!)
  - Refactored source code to make it a *library*.
  - Business unit interest increases.
  - Code branched, and more formal support role started to be envisioned.

The project: Establishing open source

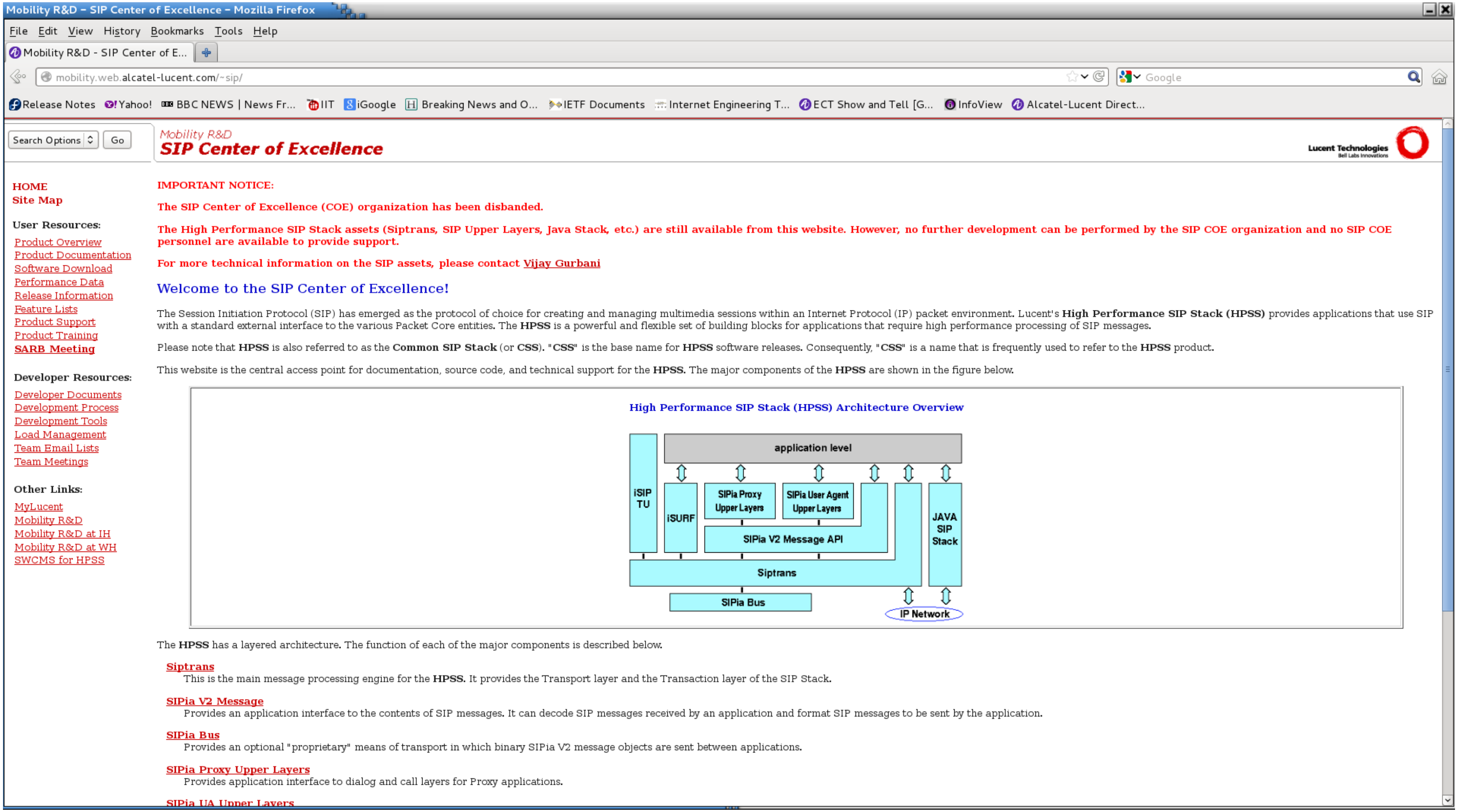
- As size of development community increased from 1-2 developers in Phase I and II to 30 developers working concurrently in Phase III, an open source group was formally formed.
  - The Common SIP Stack (CSS) Group.
- CSS has two goals:
  - Maintain an independent and common source repository such that all projects take their deliverables from CSS.
  - Evangelize the technology and the implementation by creating awareness of the resource within the company.
- (Feb 2006) Email from Jeong Kim (then Bell Labs President) asking R&D to evaluate internal SIP stack before outside requisition.

The project: By the numbers

As of 2006.

- Revenue producing asset.
- > 20 individual Bell Labs and business division projects use the asset.
- >120 individual users of the asset.
- Parts of code reused for other projects (parsing).

The project: CSS – 1 stop shop



Mobility R&D

SIP Center of Excellence

HOME
Site Map

User Resources:

- Product Overview
- Product Documentation
- Software Download
- Performance Data
- Release Information
- Feature Lists
- Product Support
- Product Training
- **SARB Meeting**

Developer Resources:

- Developer Documents
- Development Process
- Development Tools
- Load Management
- Team Email Lists
- Team Meetings

Other Links:

- MyLucent
- Mobility R&D
- Mobility R&D at IH
- Mobility R&D at WH
- SWCMS for HPSS

IMPORTANT NOTICE:

The SIP Center of Excellence (COE) organization has been disbanded.

The High Performance SIP Stack assets (Siptrans, SIP Upper Layers, Java Stack, etc.) are still available from this website. However, no further development can be performed by the SIP COE organization and no SIP COE personnel are available to provide support.

For more technical information on the SIP assets, please contact Vijay Gurbani

Welcome to the SIP Center of Excellence!

The Session Initiation Protocol (SIP) has emerged as the protocol of choice for creating and managing multimedia sessions within an Internet Protocol (IP) packet environment. Lucent's **High Performance SIP Stack (HPSS)** provides applications that use SIP with a standard external interface to the various Packet Core entities. The **HPSS** is a powerful and flexible set of building blocks for applications that require high performance processing of SIP messages.

Please note that **HPSS** is also referred to as the **Common SIP Stack** (or **CSS**). "**CSS**" is the base name for **HPSS** software releases. Consequently, "**CSS**" is a name that is frequently used to refer to the **HPSS** product.

This website is the central access point for documentation, source code, and technical support for the **HPSS.** The major components of the **HPSS** are shown in the figure below.

High Performance SIP Stack (HPSS) Architecture Overview

The **HPSS** has a layered architecture. The function of each of the major components is described below.

Siptrans
This is the main message processing engine for the **HPSS.** It provides the Transport layer and the Transaction layer of the SIP Stack.

SIPia V2 Message
Provides an application interface to the contents of SIP messages. It can decode SIP messages received by an application and format SIP messages to be sent by the application.

SIPia Bus
Provides an optional "proprietary" means of transport in which binary SIPia V2 message objects are sent between applications.

SIPia Proxy Upper Layers
Provides application interface to dialog and call layers for Proxy applications.

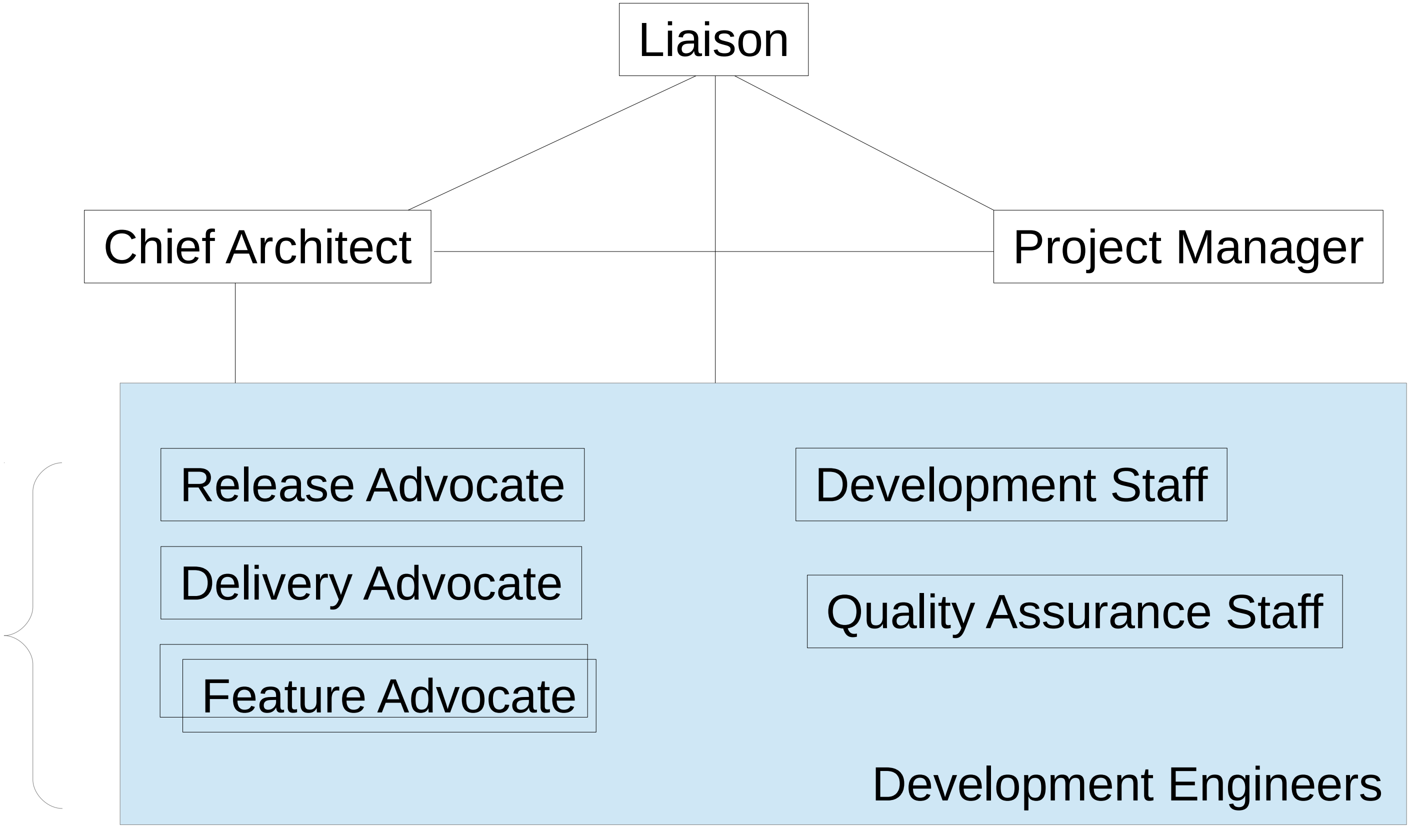
SIPia UA Upper Layers

The project: CSS – 1 stop shop

- CSS consisted of:
  - Product manager / Liaison
  - Chief Architect (“benevolent dictator”)
  - Trusted lieutenants
    - Compression
    - Monoblock
    - ...
  - Project manager
  - Development engineers

Corporate Open Source (COS)

The Project: The COS “core team”



Manage contributions from BD towards the common asset.

Roles on the core team

Liaison

- Overall responsibility for open source project; evangelizes the project
- Management of all activities performed by core team
- Interfaces with each interested business unit for new work requests
- Works closely with: Chief Architect, Project Manager

Chief Architect

- Ideally someone who founded the asset and has invested considerable energy in it
- Good software engineering skills, but also an industry overview of how to position the technology and how the technology evolves
- Must respect business decisions before personal vision (Important!)

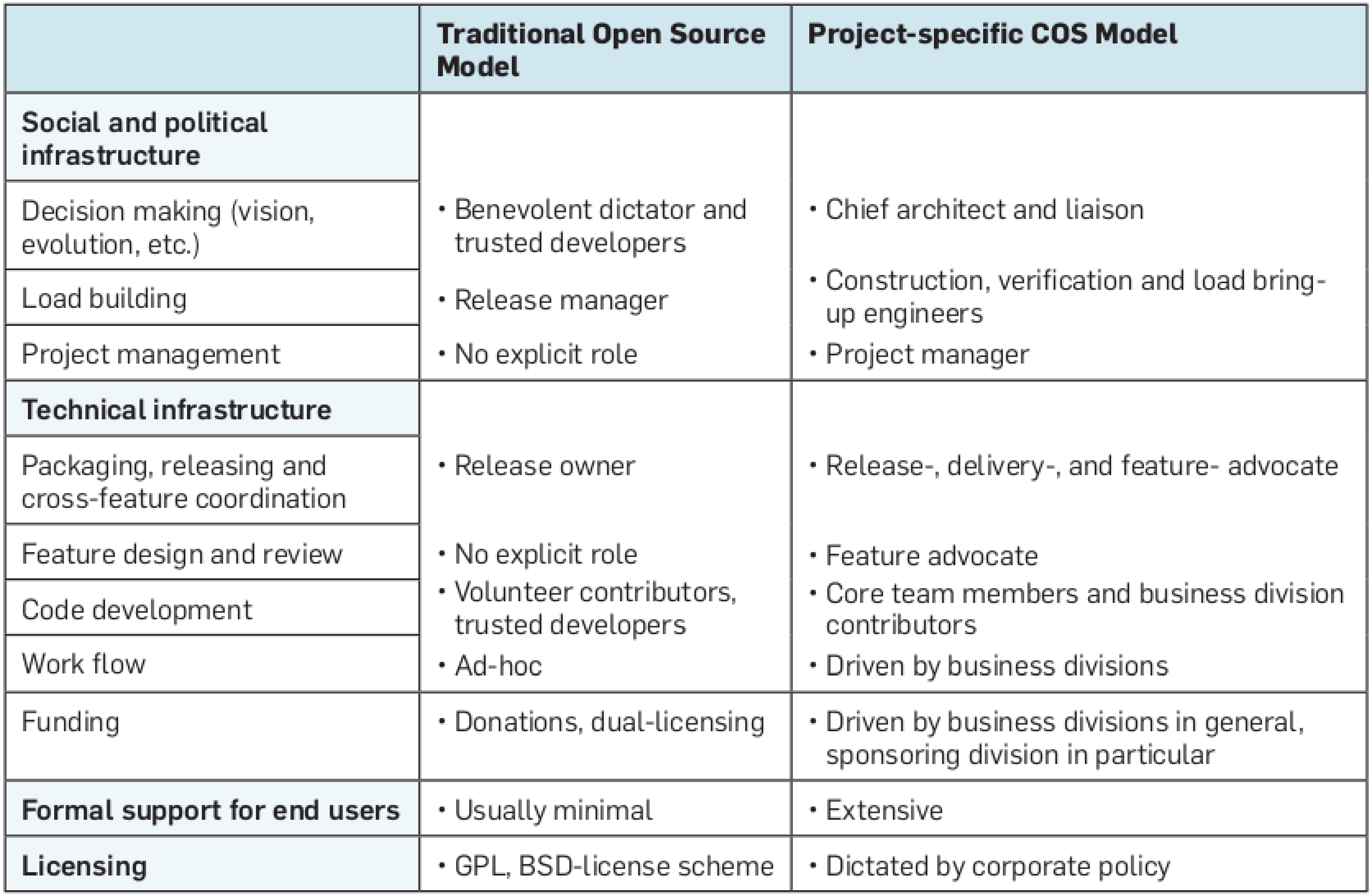
Project Manager

- Assist in release and load planning
- Manage tools to define and track features
- Ensure (light weight) process compliance

Roles on the core team

- Traditional developer and QA roles exist in a COS.
- But also
  - Business unit delivery advocate: assist in build integration and assimilate contributions from the BU into the core software.
  - Feature advocates: In charge of substantive features and saw them to completion (trusted lieutenants).
  - Release advocates: Code czar for a specific release.
- These roles were continuously reassigned to different members.

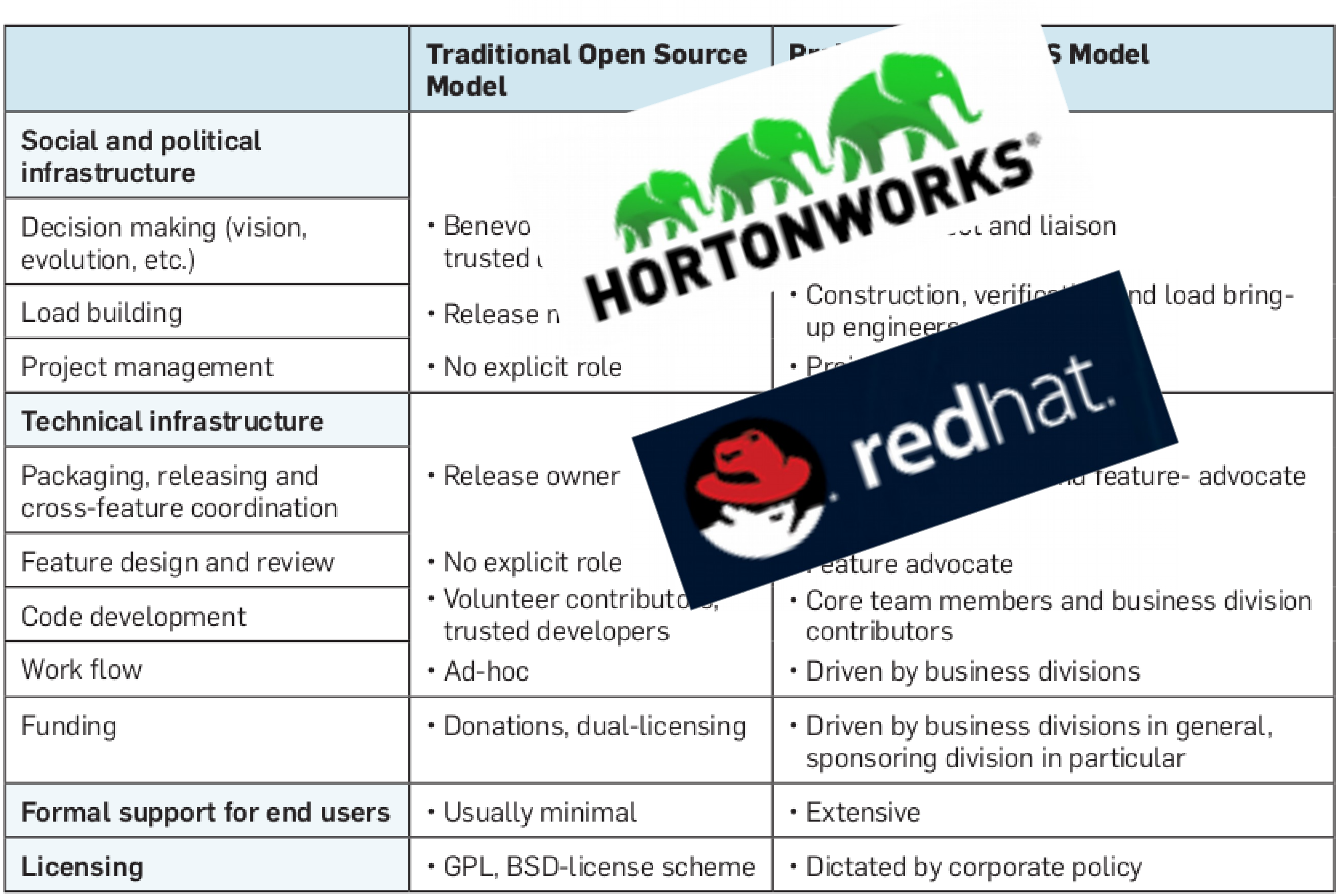
The Project: Summary comparison



	Traditional Open Source Model	Project-specific COS Model
Social and political infrastructure		
Decision making (vision, evolution, etc.)	• Benevolent dictator and trusted developers	• Chief architect and liaison
Load building	• Release manager	• Construction, verification and load bring-up engineers
Project management	• No explicit role	• Project manager
Technical infrastructure		
Packaging, releasing and cross-feature coordination	• Release owner	• Release-, delivery-, and feature- advocate
Feature design and review	• No explicit role	• Feature advocate
Code development	• Volunteer contributors, trusted developers	• Core team members and business division contributors
Work flow	• Ad-hoc	• Driven by business divisions
Funding	• Donations, dual-licensing	• Driven by business divisions in general, sponsoring division in particular
Formal support for end users	• Usually minimal	• Extensive
Licensing	• GPL, BSD-license scheme	• Dictated by corporate policy

From [Gurbani 2010]

The Project: Summary comparison



	Traditional Open Source Model	**Pr[illegible] S Model**
Social and political infrastructure		
Decision making (vision, evolution, etc.)	• Benevo[illegible] trusted [illegible]	• [illegible]t and liaison
Load building	• Release n[illegible]	• Construction, verific[illegible] nd load bring-up engineer[illegible]
Project management	• No explicit role	• Pr[illegible]
Technical infrastructure		
Packaging, releasing and cross-feature coordination	• Release owner	• [illegible] feature- advocate
Feature design and review	• No explicit role	• [illegible]eature advocate
Code development	• Volunteer contributo[illegible], trusted developers	• Core team members and business division contributors
Work flow	• Ad-hoc	• Driven by business divisions
Funding	• Donations, dual-licensing	• Driven by business divisions in general, sponsoring division in particular
Formal support for end users	• Usually minimal	• Extensive
Licensing	• GPL, BSD-license scheme	• Dictated by corporate policy

From [Gurbani 2010]

Postmortem: Why did we succeed?

- ... and can our success be replicated?
- Our success was a convergence of:
  - Being on the cusp of a new technology (protocol development in the IETF);
  - Having a feature-rich, stable, and standards-compliant implementation when the company was looking for SIP assets;
  - Having a significant pool of users who were interested and capable developers.

Postmortem: Why did we succeed?

- Success criteria:
  - Technology is needed by several product groups (hence a reason to pool resources).
  - Technology is relatively immature, thus requirements and features are not fully known at the outset.
  - Product groups have differing needs and specific expertise in customizing the software, ensuring that everyone benefits from contributions of each group.
  - Initial asset has a sound modular architecture, making it easier to evolve.
  - Recognize (and accommodate) the tension between cultivating a common resource and the pressure to get specific releases of products out on time (in other words, the benevolent dictator cannot be petulant).

Postmortem: Lessons learnt (Primary)

- For such projects to succeed, it is imperative that they benefit from a large and organized sponsoring business division within the corporation that can act as a champion for the common asset.
- Formal support and ownership required as the common asset is integrated into products being created by other business divisions cannot be ignored.
- Can't simply “throw the software over the wall.”
- Wide participation, down to supply-chain level.

Postmortem: Lessons learnt (Secondary)

- Requirements and software processes:
  - Must scale from organizational view to a company-wide view: prioritize features across disjoint projects, identify common work, coordinate virtual teams, ensure overall product meets the needs of all customers.
- Work assignment and incentive structure:
  - Management support for the “benevolent dictator”.
  - Management support for “trusted lieutenants”.
  - Cross-organizational support for developers.
  - Need for a *meritocracy*.

Postmortem: Lessons learnt (Secondary)

- Software architecture
  - Unsurprisingly, independent strains must be discouraged or tracked for an eventual merge.
  - Modular architecture, well defined interfaces, “trusted lieutenants” in charge of key components.
  - Refactoring, not reinvention (e.g., SIP stack parser).
  - Customization while preserving core architecture.
  - Need to architect software in ways appropriate for different development styles and organizational settings.

Postmortem: Lessons learnt (Secondary)

- Web location, web location, web location
  - Disseminate COS projects as widely as possible.
  - Developers need to know that the COS is a core company asset.
  - Advertise at grass roots level (developer to developer) to the executive level.
- Tool uniformity:
  - Use common set of development and source control tools. (This is easier said than done; every organization has affinity to their own tools.)
  - Distributed source code should fit the load building strategy of a particular group.

Summary / Wrapup / Q&A

- Sizable interest in Inner Source [Stol 2014].
- Our contributions [Gurbani 2005,2006,2010] demonstrates a model for corporations adopting what is now being termed as Inner Source.
- Obligatory question: is the “bazaar” model the best model?
  - The curious case of ~~benjamin~~ the config button*

* Poul Henning-Kamp, “A generation lost in the bazaar,” *Communications of the ACM*, 55(10), 2012.

Thank you!

- Vijay K. Gurbani
  vijay.gurbani@nokia-bell-labs.com
  https://www.bell-labs.com/usr/vijay.gurbani
  Bell Laboratories, Nokia Networks