Software Libre:

Conceptos. Ingeniería del Software Libre.

Juan José Amor
(GSyC LibreSoft, URJC)
jjamor@gsyc.escet.urjc.es

Universidad del Valle de Atemajac
Guadalajara, 8 Noviembre, 2007

Licencia

Resumen

1. Introducción al software libre 
2. La Catedral y el Bazar
3. Contando código (patatas, etc)
4. Evolución
5. Dinámica de los desarrolladores
6. Esfuerzo
7. Conclusiones

Esto del software libre ...

- ¿Nos suenan? GNU/Linux, Apache, GNOME, KDE, Openoffice ...
- ¿Qué es lo realmente nuevo?
  - El modelo de software libre
  - La propiedad intelectual clásica, cuestionada
  - Los usuarios retoman el control
  - ¿un nuevo modelo para un nuevo mundo?
  - La viabilidad del modelo empieza a quedar probada

¿Qué es el software libre?

- En resumen, es aquél que garantiza 4 libertades:
  - La libertad de uso
  - La libertad de redistribución
  - La libertad de modificación
  - La libertad de distribuir modificaciones
- Consecuencia: disponibilidad del código fuente.
- Las licencias garantizan las libertades.

¿Qué aporta al mercado?

- A veces, elegido como primera opción: sistema operativo GNU/Linux, servidor web Apache, ofimática Openoffice...
- Grandes sinergias:
  - Por reutilización de código.
  - Por reutilización de conocimiento.
  - ...

¿Qué aporta al mercado?

- Competitividad:
  - El código fuente permite mejoras y adaptaciones a gran escala.
  - Se promueve la estandarización, pero manteniendo competitividad entre fabricantes.
  - La competición se consigue con el soporte, no con la venta de licencias.

Consecuencias

- A medida que el software libre se hace popular:
  - La industria del software tradicional necesita reciclarse.
  - Necesaria nueva industria basada en soporte, servicios y desarrollo libre.
  - Fomenta la competitividad en soporte y en evolución de software.

Impacto

- En los modelos de coste.
- En la apertura: puede ser estudiado, modificado, inspeccionado, mejorado...
- En la distribución: nuevos canales o métodos de distribución.
- En el desarrollo: ¿nuevos modelos?
- En soporte y mantenimiento: verdadera competición.
- Se mezclan cooperación y competición.

Para nosotros, los usuarios

- El software libre no es ni mejor, ni peor. Simplemente diferente.
- En varias aplicaciones, contamos con excelentes productos con excelente soporte.
- No se compra el software, se contrata el soporte.
- Servicios, no productos.

Resumen

1. Introducción al software libre
2. La Catedral y el Bazar 
3. Contando código (patatas, etc)
4. Evolución
5. Dinámica de los desarrolladores
6. Esfuerzo
7. Conclusiones

La Catedral y el Bazar

- Autor: Eric S. Raymond (1997).
- Linux presenta un “nuevo” modelo de desarrollo.
- Modelo “revolucionario”: permite construir grandes sistemas software en entornos sin aparente organización.
- El artículo es un ensayo sobre las observaciones sobre el modelo de Linux y su replicación en otros proyectos.

El modelo “Catedral”

- Es el modelo “clásico”.
- Entorno “cerrado”.
- Pequeño grupo de desarrolladores líderes.
- Solo liberaciones estables.
- Modelo típico de aplicaciones que siguen modelos tradicionales de cascada o espiral.
- También en algunos proyectos clásicos de software libre: Emacs, GNU GCC...

El modelo “Bazar”

- Introducido por Linus Torvalds.
- Entorno abierto. Cualquiera puede participar.
- No hay líderes claros. Número de desarrolladores indeterminado.
- “Dictador Benevolente”.
- Libera rápido y a menudo.
- Ejemplo: Linux.

¿Cuál es la sorpresa?

- El estilo bazar:
  - Crea una comunidad de programadores cada uno con sus propios intereses y agendas, a modo de un “bazar” oriental.
  - Con repositorios de código en los que cualquiera puede proponer modificaciones.
- Lo sorprendente:
  - Emerge un sistema completo, estable, coherente.
  - Crecimiento muy rápido.
  - Linux no acaba en un mar de confusión.

Las “Lecciones”

- El ensayo se organiza en una serie de “lecciones” que recibe el autor del análisis del desarrollo de Linux.
- Estas lecciones serían extrapolables a otros proyectos de desarrollo de software libre.
- El autor verifica las lecciones en un proyecto propio: “fetchmail”.

Algunas lecciones interesantes

- *Todo buen trabajo de software comienza a partir de las necesidades personales del programador.*
  - Muchos empiezan como “juguetes” de sus programadores.
  - En el software propietario, los programadores no hacen lo que quieren, sino lo que les mandan.
  - ¿explica esto la motivación de los que hicieron Linux?

Algunas lecciones interesantes

- *Los buenos programadores saben qué escribir. Los mejores, qué reescribir (y reutilizar).*
  - Linux empezó como derivación de Minix.
  - La tradición del mundo UNIX de compartir las fuentes siempre se ha prestado a la reutilización del código.

Algunas lecciones interesantes

- *Cuando se pierde el interés en un programa, el último deber es heredarlo a un sucesor competente.*
  - Si abandonamos el desarrollo de un programa, es interesante que alguien nos suceda.
  - En el mundo del bazar, siempre habrá un “hacker” motivado para retomar el proyecto y dirigirlo hacia sus fines.

Algunas lecciones interesantes

- *Tratar a los usuarios como colaboradores es la forma más apropiada de mejorar el código, y la más efectiva de depurarlo.*
  - En Linux, muchos usuarios son también “hackers”.
  - Gracias al código fuente, los “hackers” pueden ser “hackers efectivos”.
  - Reducción del tiempo de depuración.
  - Usuarios que diagnostican fallos, sugieren correcciones, ayudan a las mejoras.

Algunas lecciones interesantes

- *Libera rápido y a menudo.*
  - Característica Linux: en períodos de desarrollo, se liberan muchas versiones.
  - A veces más de una al día.
  - Mantiene a los hackers estimulados y recompensados:
    - porque ven inmediatamente publicadas sus contribuciones
    - porque ven que gracias a ellos mejora el sistema

Algunas lecciones interesantes

- *Ley de Linus*: "*si hay muchos ojos, los fallos son triviales*".
  - Si tenemos una gran base de betatesters y co-programadores, todos los fallos serán diagnosticados rápidamente y serán de corrección obvia para alguno.
  - Una de las principales características del modelo Bazar.
- Hay varias lecciones más ...

Críticas

- Es un ensayo, no hay rigor científico.
- Asistemático.
- Intento de generalizar el caso de Linux a todo el software libre.
- Algunos críticos ven Linux más como catedral que como bazar:
  - Estructura piramidal de mando.
  - Responsabilidades distribuidas aunque no de manera explícita.

Catedrales y Bazares

- ¿Software libre = Bazar?
  - ¿Hay modelo de desarrollo en el software libre?
  - ¿Cuántos proyectos tienen realmente muchas contribuciones de muchos desarrolladores?
  - La definición de bazar no es precisa.
- Software libre no es Bazar:
  - Software libre hace referencia a 4 libertades.

“No Silver Bullet” (Brooks)

- Prácticas que mejoran el proceso de desarrollo:
  - Comprar, no siempre crear.
    - Evitar desarrollar cuando se pueden integrar piezas de terceros.
  - Refinar requisitos, prototipado rápido.
    - Ayuda a conocer mejor qué se quiere conseguir.
  - Contar con buenos diseñadores.
    - Para detectar mejor los problemas en las fases iniciales o de diseño.

En el software libre...

- ... estas prácticas de Brooks llevan años aplicándose:
  - Integrando componentes de otros proyectos de software libre (gracias a su licencia).
  - Con el principio “release early, release often”.
  - Varios estudios prueban que los grandes proyectos de software libre son liderados por un pequeño grupo de desarrolladores expertos (meritocracia).

Los análisis cuantitativos

- Medición del proceso de desarrollo de software libre
  - Desde un punto de vista cuantitativo
- Análisis técnicos
  - Código fuente
  - Documentación
  - Erratas ...
- Análisis sociales

Ingeniería SW Empírica

- Podríamos definirla como:
  - Comprensión del proceso de desarrollo del software a partir del estudio histórico de varios proyectos.
- Software libre:
  - Aporta gran cantidad de datos que facilita su estudio empírico.
  - Métricas de código fuente, CVS, listas de correo, BTS, etc.

Resumen

1. Introducción al software libre
2. La Catedral y el Bazar
3. Contando código (patatas, etc) 
4. Evolución
5. Dinámica de los desarrolladores
6. Esfuerzo
7. Conclusiones

Acerca de Debian

- Sistema Operativo Libre
- Basado en núcleos libres (Linux, HURD, *BSD), y aplicaciones libres (GNU, mozilla, etc).
- Creado y mantenido por voluntarios.
- “Contrato Social”
- “Policies”
- Versiones simultáneas: estable, pruebas, “sid”, experimental.

Cómo estudiamos Debian

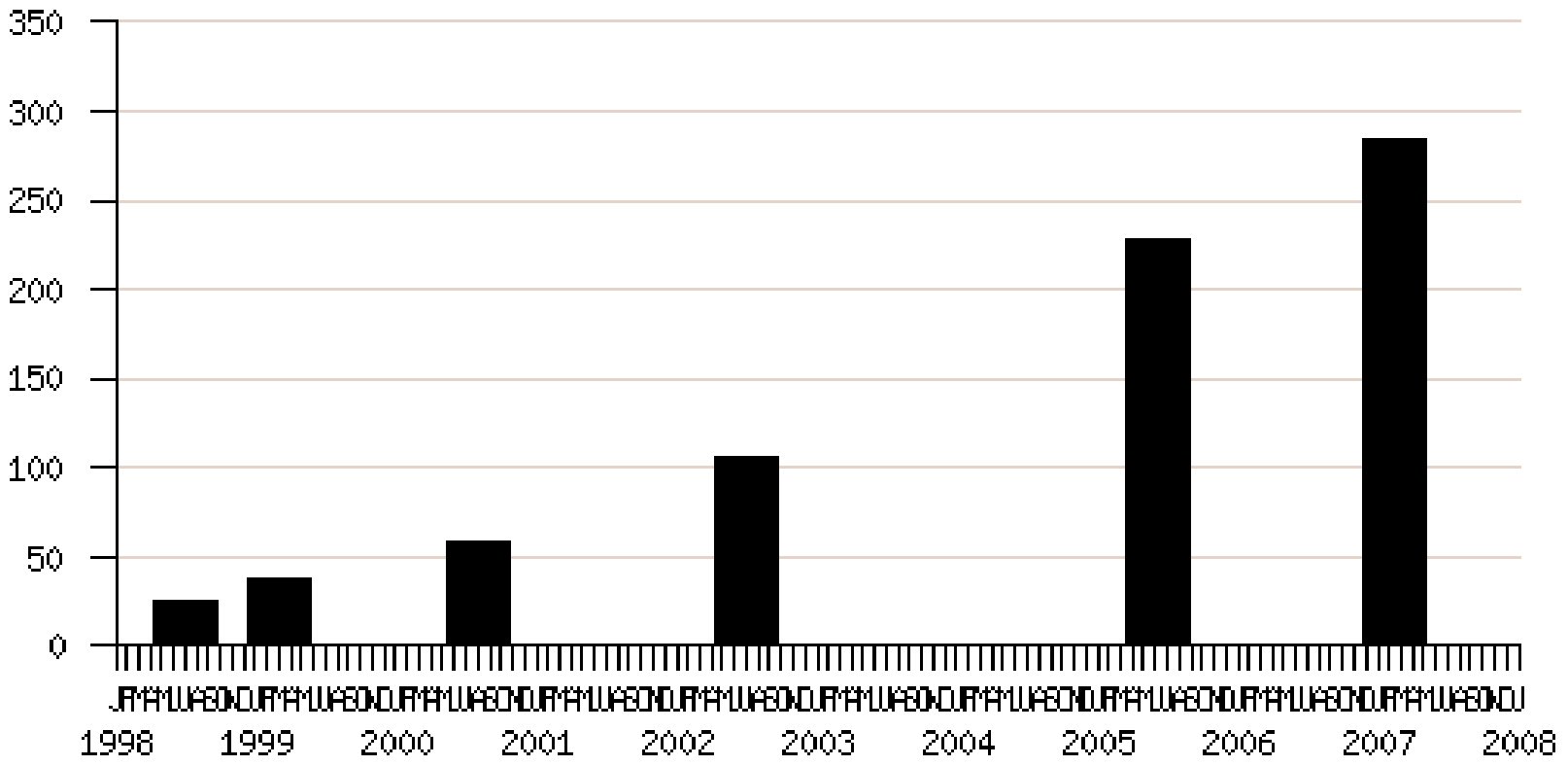
- Metodología: descarga, descompresión, cuentas y sumas.
- Resultados: medidos en SLOCs.
- Identificación de Lenguajes de Programación.
- Identificación de ficheros iguales (MD5).
- Estimaciones mediante técnicas clásicas (COCOMO).
- Versiones estudiadas: 2.0 (hamm) a 4.0 (etch)

Tamaño de Debian

Version	Release	Source Pkgs.	Size	Mean pkg siz
2.0	Jul 1998	1,096	25	23,050
2.1	Mar 1999	1,551	37	23,910
2.2	Aug 2000	2,611	59	22,650
3.0	Jul 2002	4,579	105	22,860
3.1	Jun 2005	8,560	216	25,212
4.0	Apr 2007	10,106	288	28,544

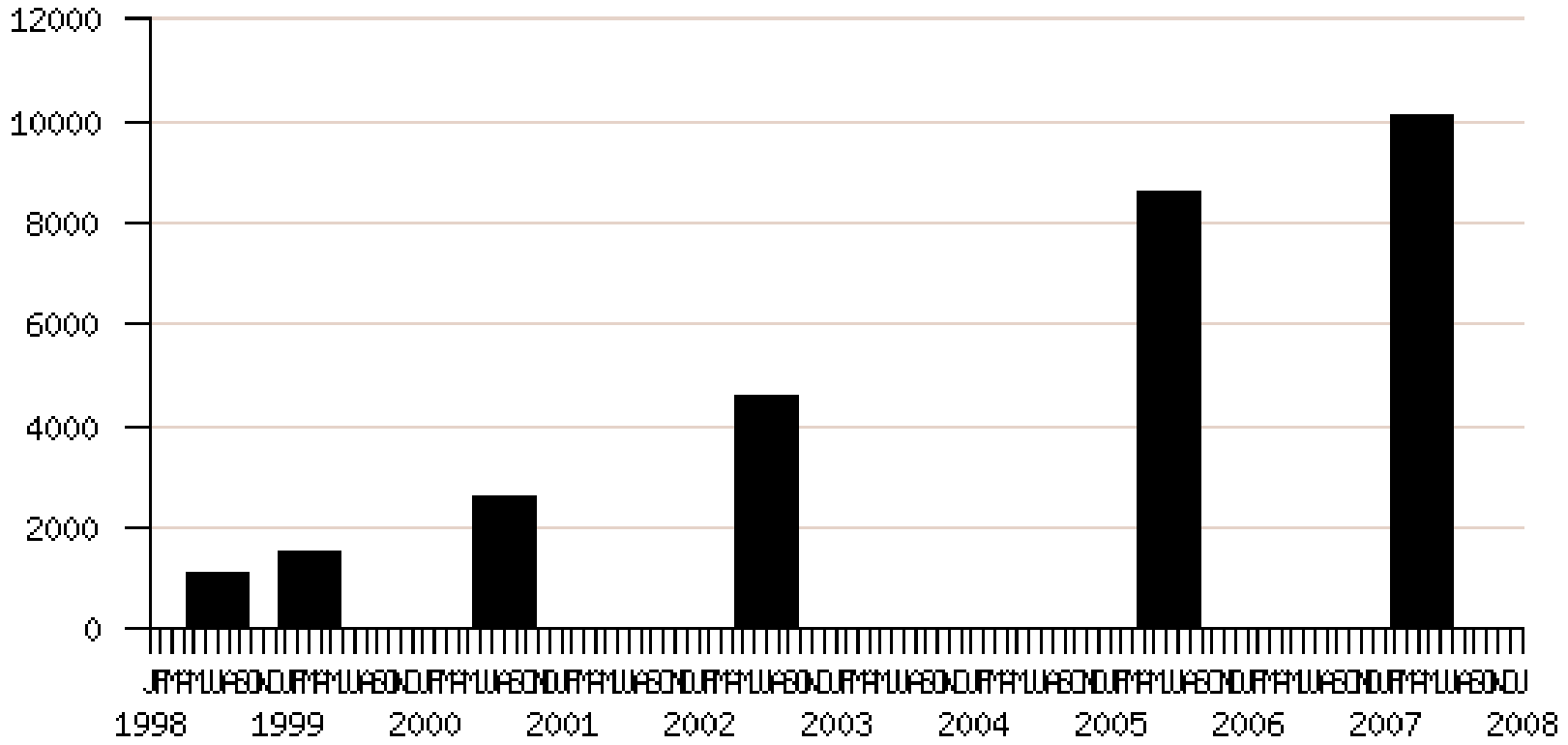
Evolución de tamaño de distribuciones.
(Tamaño en MSLOC).

Tamaño de Debian



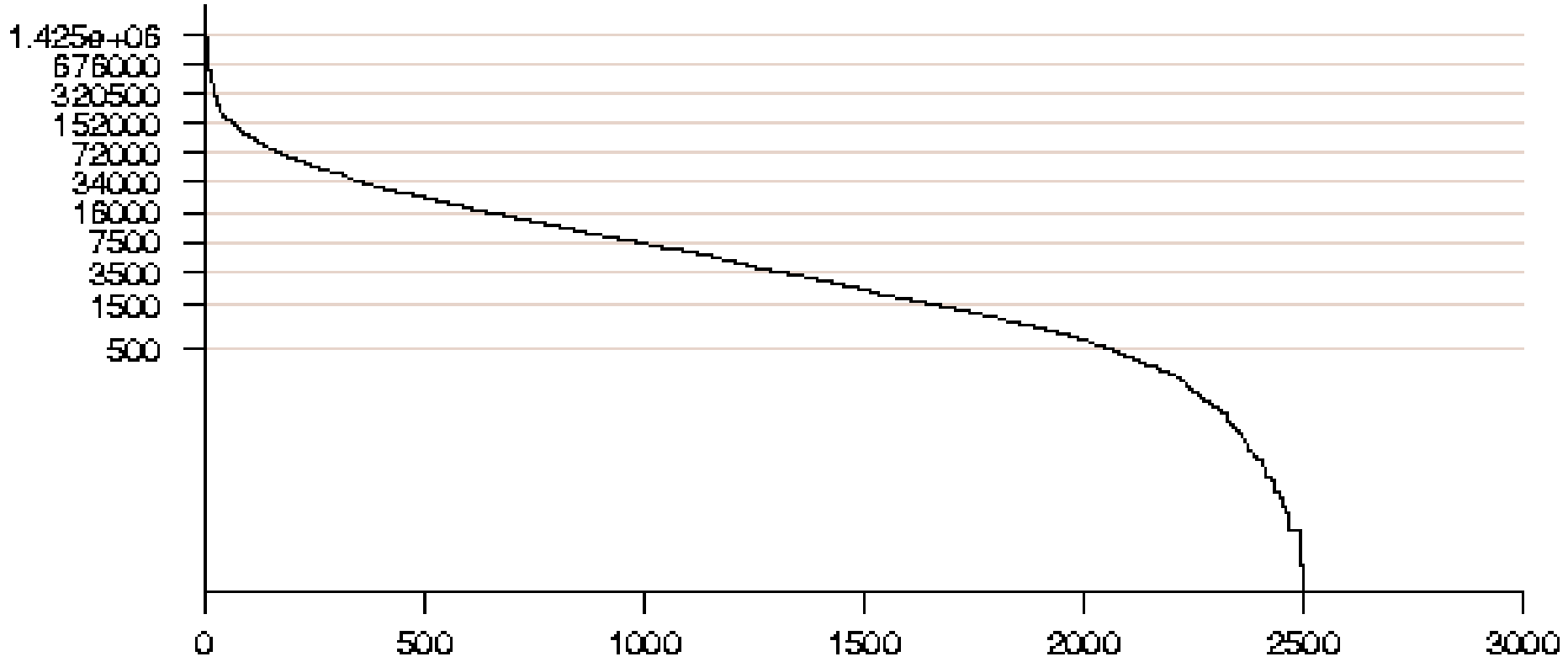
Evolución de tamaño (MSLOC)

Tamaño de Debian



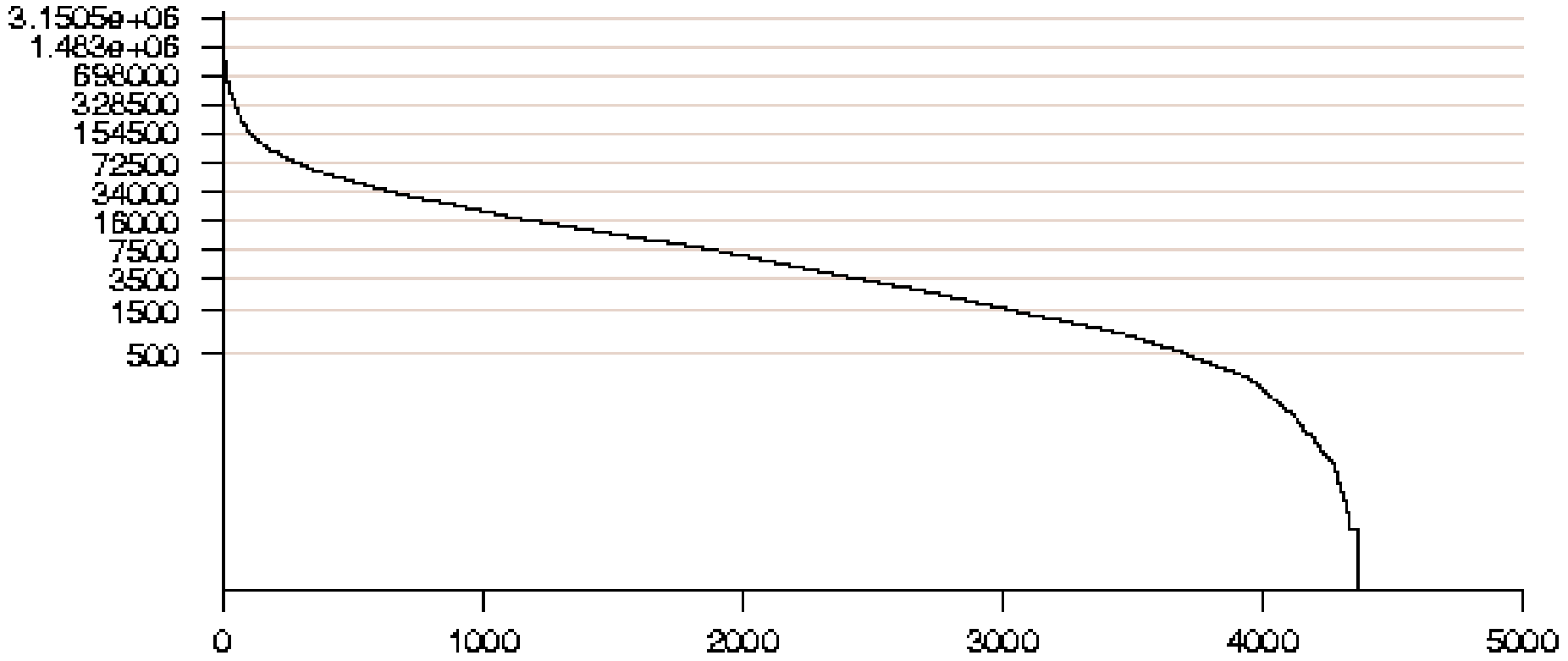
Evolución de tamaño (Paquetes)

Tamaño de Debian



Distribución de tamaños (Debian 2.2)

Tamaño de Debian



Distribución de tamaños (Debian 3.0)

10 mayores paquetes

Rank	Package	Version	Size	Files
1	Xfree86	3.3.2.3	1200000	4100
2	Xemacs20	20.4	780000	1794
3	Egcs	1.0.3a	705000	4437
4	Gnat	3.10p	600000	1939
5	Kernel-source	2.0.34	575000	1827
6	Gdb	4.17	570000	1845
7	Emacs20	20.2	560000	1061
8	Lapack	2.0.1	400000	2387
9	Binutils	2.9.1	390000	1105
10	Gcc	2.7.2.3	350000	753

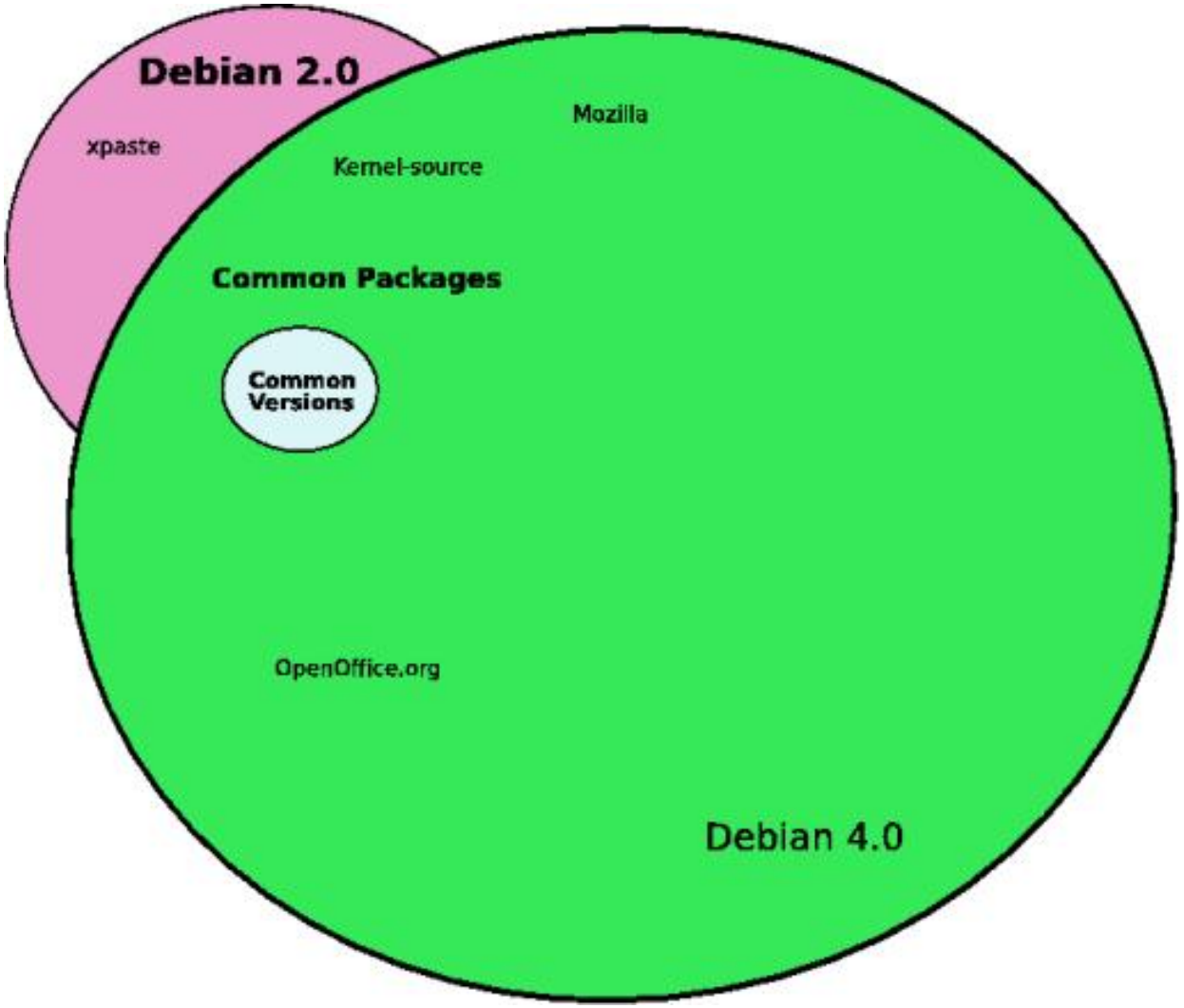
"Top-ten" paquetes en Debian 2.0

10 mayores paquetes

Rank	Package	Version	Size	Files
1	Openoffice.org	1.1.2...	5180000	19011
2	Kernel-source	2.6.8	4040000	13974
3	nvu	0,8	2470000	10780
4	Mozilla	2:1.7.3	2440000	10713
5	gcc-3.4	3.4.3	2420000	22992
6	xfs-xtt	1	2330000	7081
7	xfree86	4.3.0...	2320000	7216
8	vnc4	4.0	2060000	6790
9	insight	6.1+...	1690000	4063
10	kfreebsd5-src	5,3	1630000	4968

"Top-ten" paquetes en Debian 3.1

Mantenimiento de paquetes

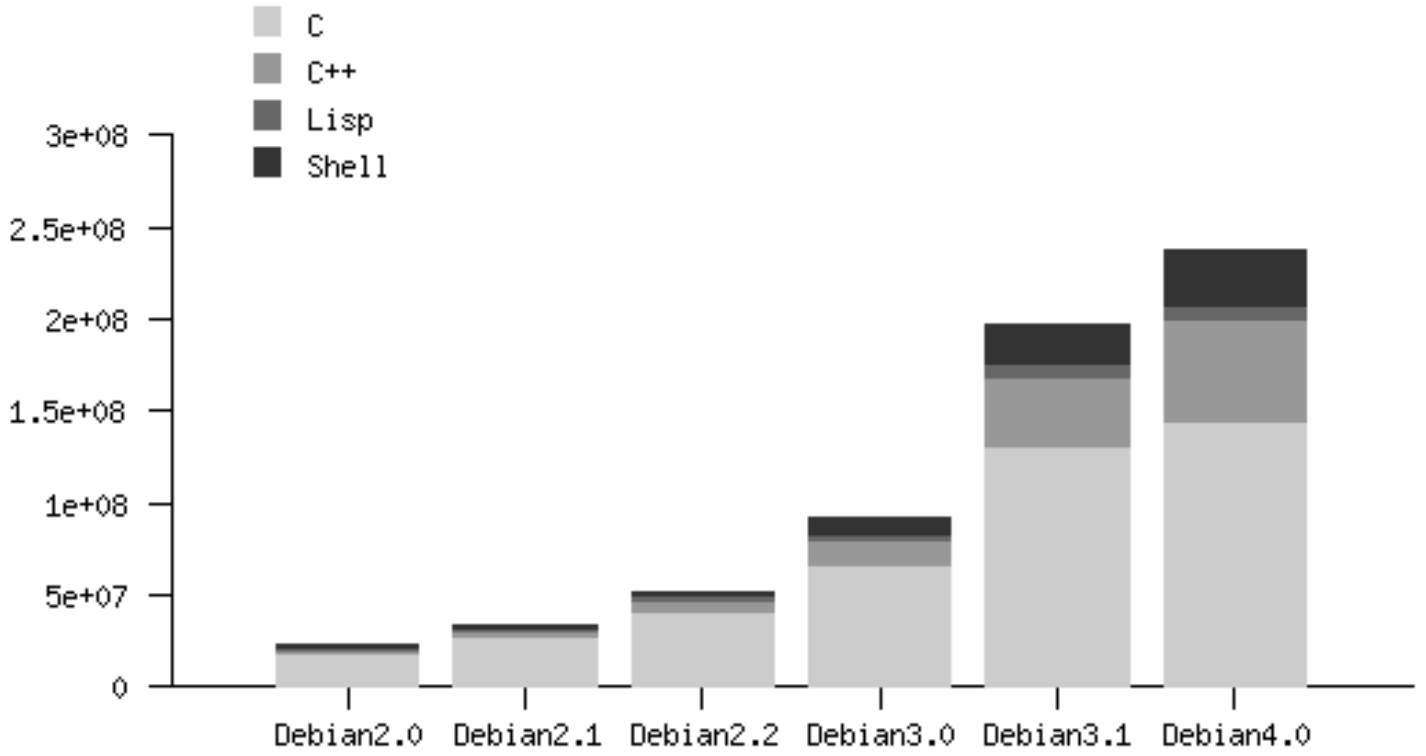


Mantenimiento de paquetes

Version	Com. pkgs	Com. vers.	SLOC com. vers.	Files com. vers.
2.0	721	132	1,235,327	6,338
2.1	995	188	1,759,679	8,772
2.2	1,710	388	3,855,356	18,781
3.0	3,236	1,020	10,173,837	42,474
3.1	7,300	3,843	59,235,197	258,537
4.0	10,106	10,106	288,461,743	1,198,735

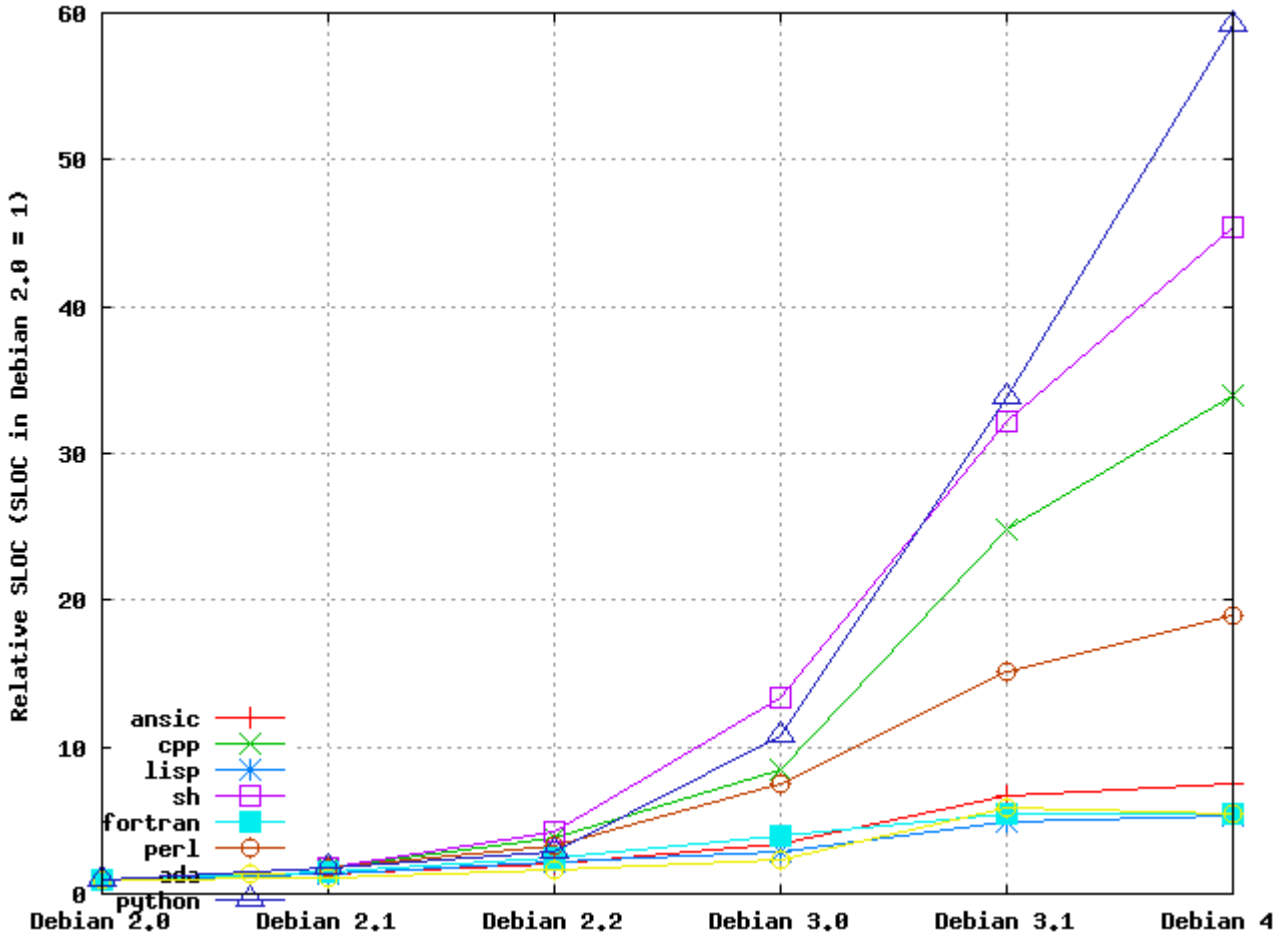
Paquetes / SLOC en común entre todas las versiones y Debian 4.0.

Evolución de lenguajes



Evolución de los lenguajes más usados.

Evolución de lenguajes



Crecimiento relativo de lenguajes.

Conclusiones: Debian

- Duplicación de tamaño cada 2 años, salvo 4.0
- Tamaño medio de paquetes casi constante
- Existen paquetes que no han cambiado a lo largo de años
- Evolución hacia el escritorio
- El lenguaje más usado es C, pero los nuevos son los que crecen más rápidamente
- Debian GNU/Linux, probablemente la mayor agrupación de software realizada hasta la fecha

Referencia

http://debian-counting.libresoft.es

Resumen

1. Introducción al software libre
2. La Catedral y el Bazar
3. Contando código (patatas, etc)
4. Evolución 
5. Dinámica de los desarrolladores
6. Esfuerzo
7. Conclusiones

Evolución de Software

- Los sistemas grandes evolucionan (crecen)
- El mantenimiento también crece y se hace más costoso y difícil.
  - Por adición de nuevas características.
  - Refactorizaciones / mejoras en general.
  - Corrección de fallos.
- Estudios de evolución previos:
  - Sistemas “cerrados” y libres.

Evolución del software

- “Leyes de Lehman”:
  - Cuando un sistema crece, es más difícil añadir nuevas características salvo si se reorganiza diseño/refactoriza.
- Turski:
  - El crecimiento del sistema se ralentiza con el tiempo (“comportamiento sublineal”).
- Gall et al:
  - Considerar crecimiento de subsistemas en en lugar del sistema completo.

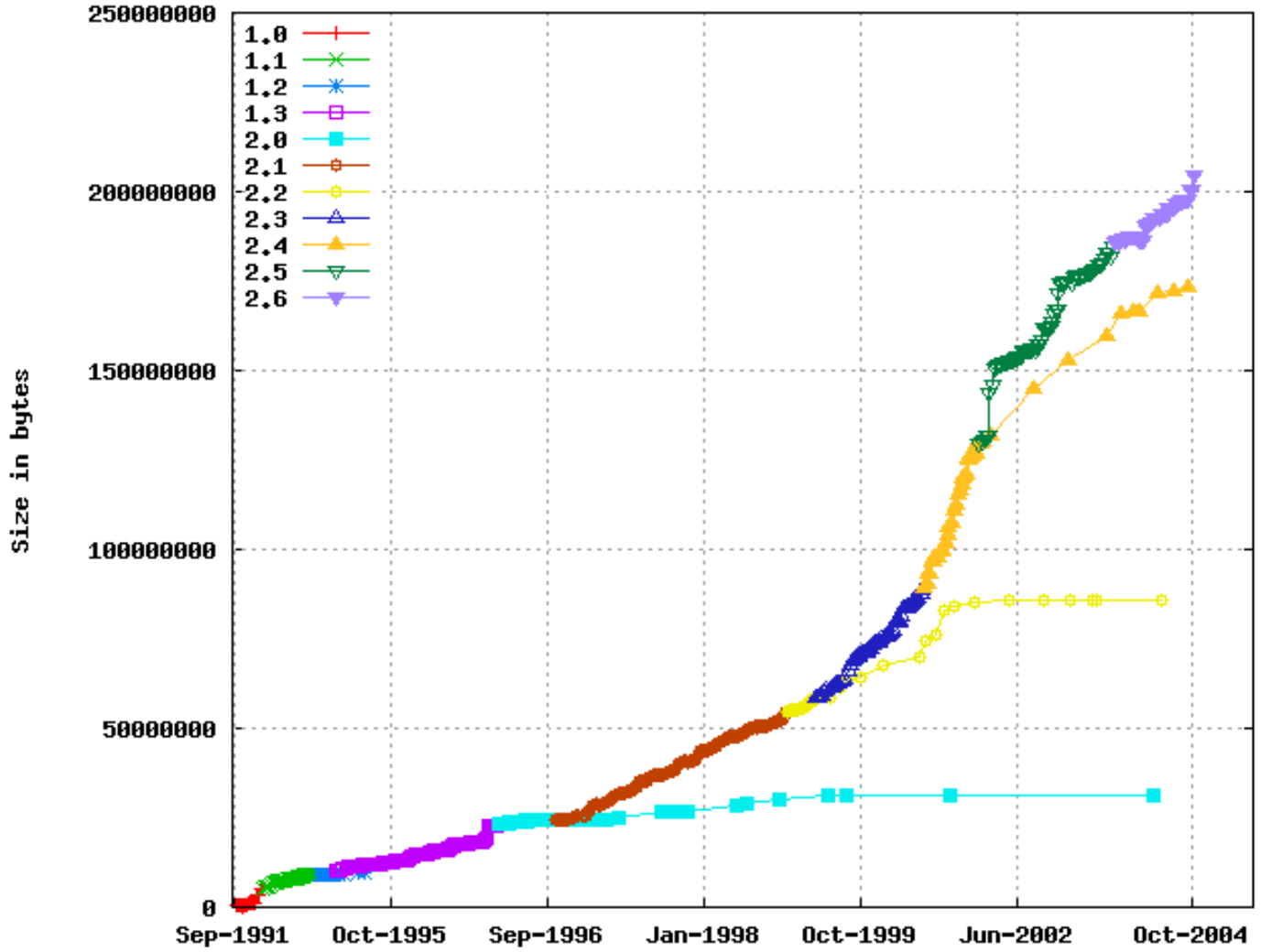
Evolución de Linux

- Trabajo previo:
  - Godfrey, Tu: Evolution in Open Source Software: A Case Study.
    - Escrito en 2000
    - Núcleos Linux 1.0 a 2.3.39
    - Comparación con ViM
- Nosotros repetimos el estudio
  - Finales de 2004
  - Núcleos 1.0 a 2.6.10
  - Comparación con *BSD

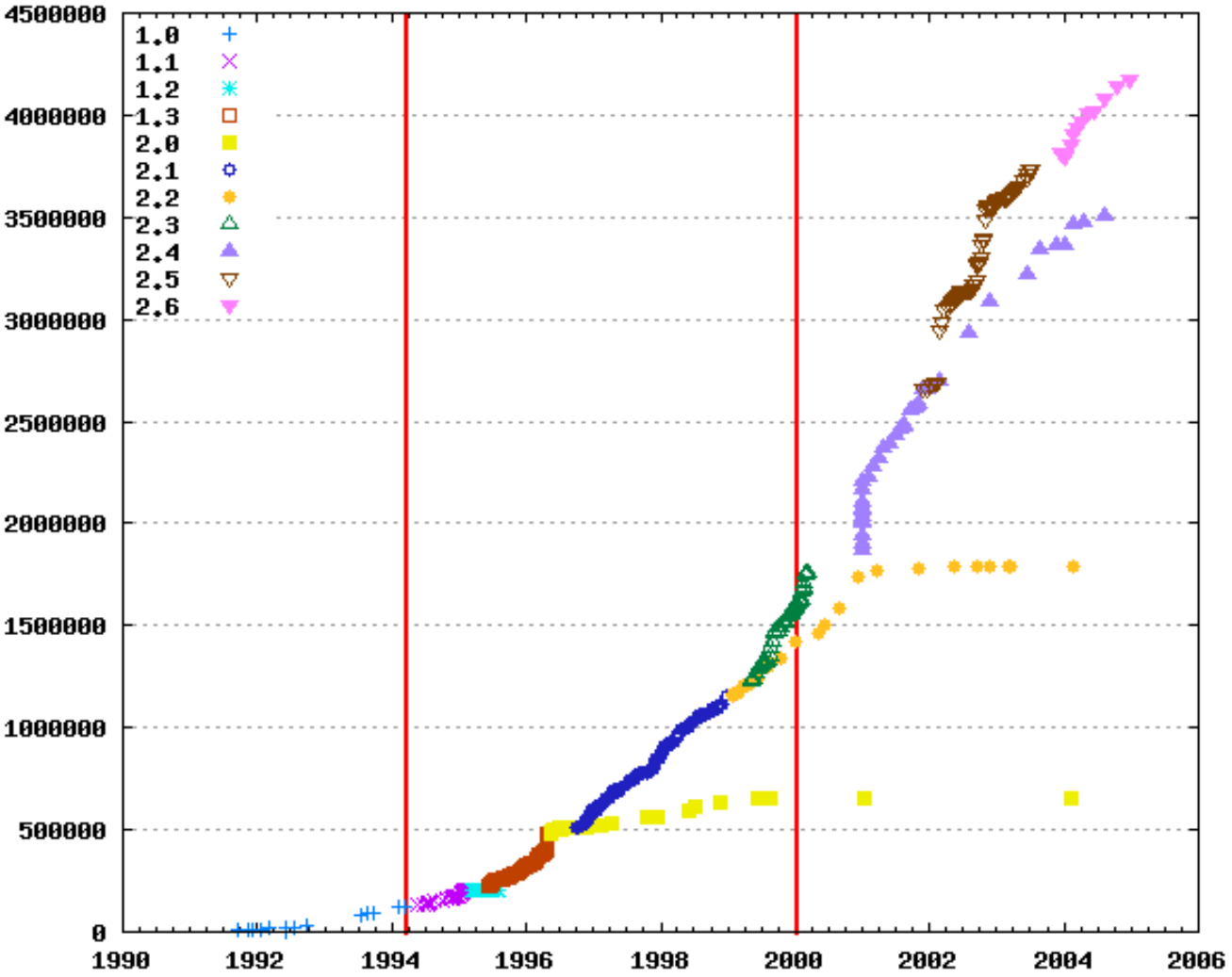
Metodología

- Estudio del crecimiento de Linux:
  - Por tamaño de distribución de los tar.gz
  - SLOCs:
    - Godfrey usó LOCs (“wc -l”) y ULOC (con awk).
    - Usamos SLOCCount:
      - Identifica lenguaje de programación, a la hora de identificar líneas que no cuentan.
      - Se pueden extraer estadísticas de uso de lenguajes de programación.
      - Pero siguen siendo LOC físicas.

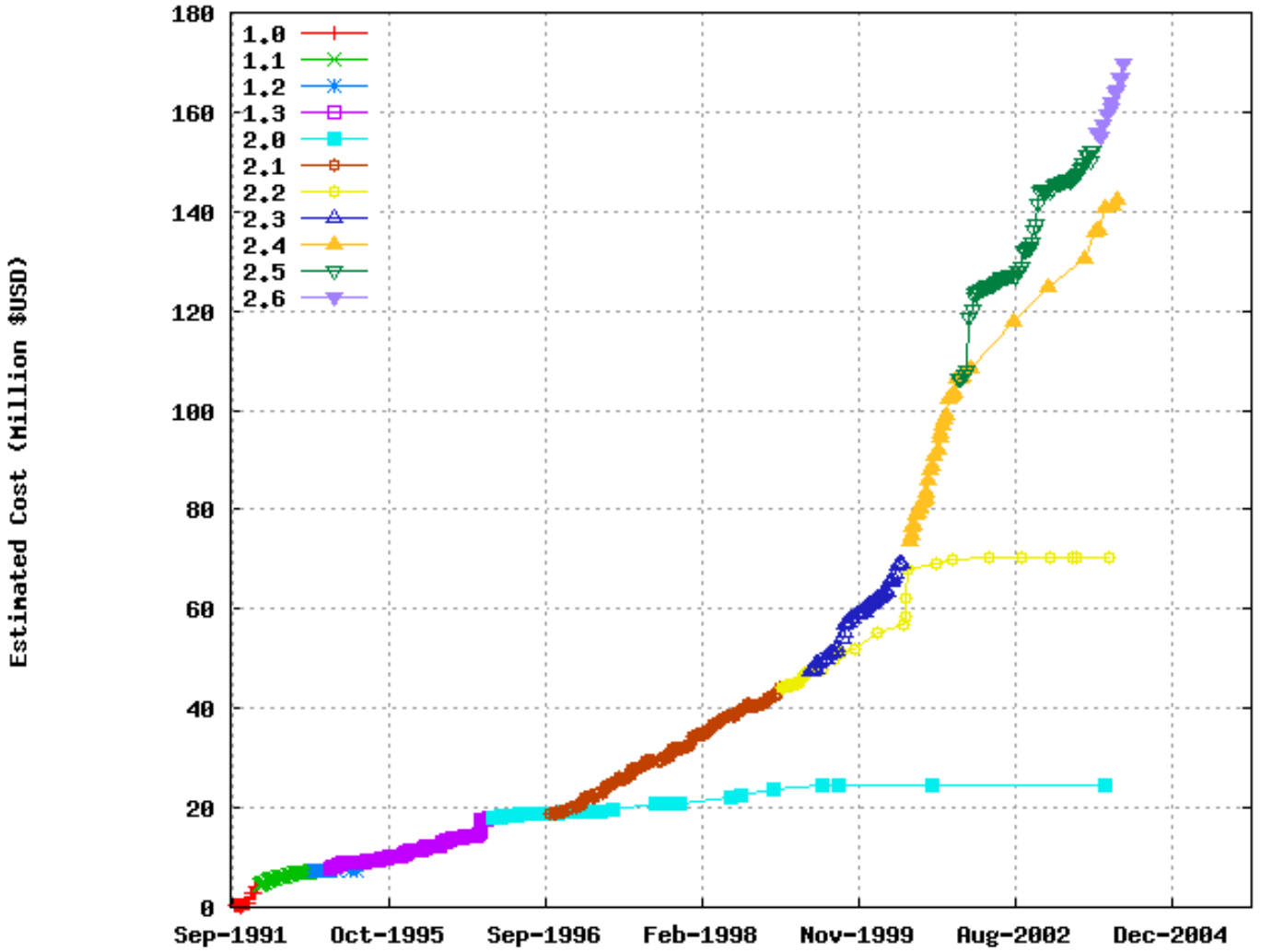
Evolución de Linux (tarballs)



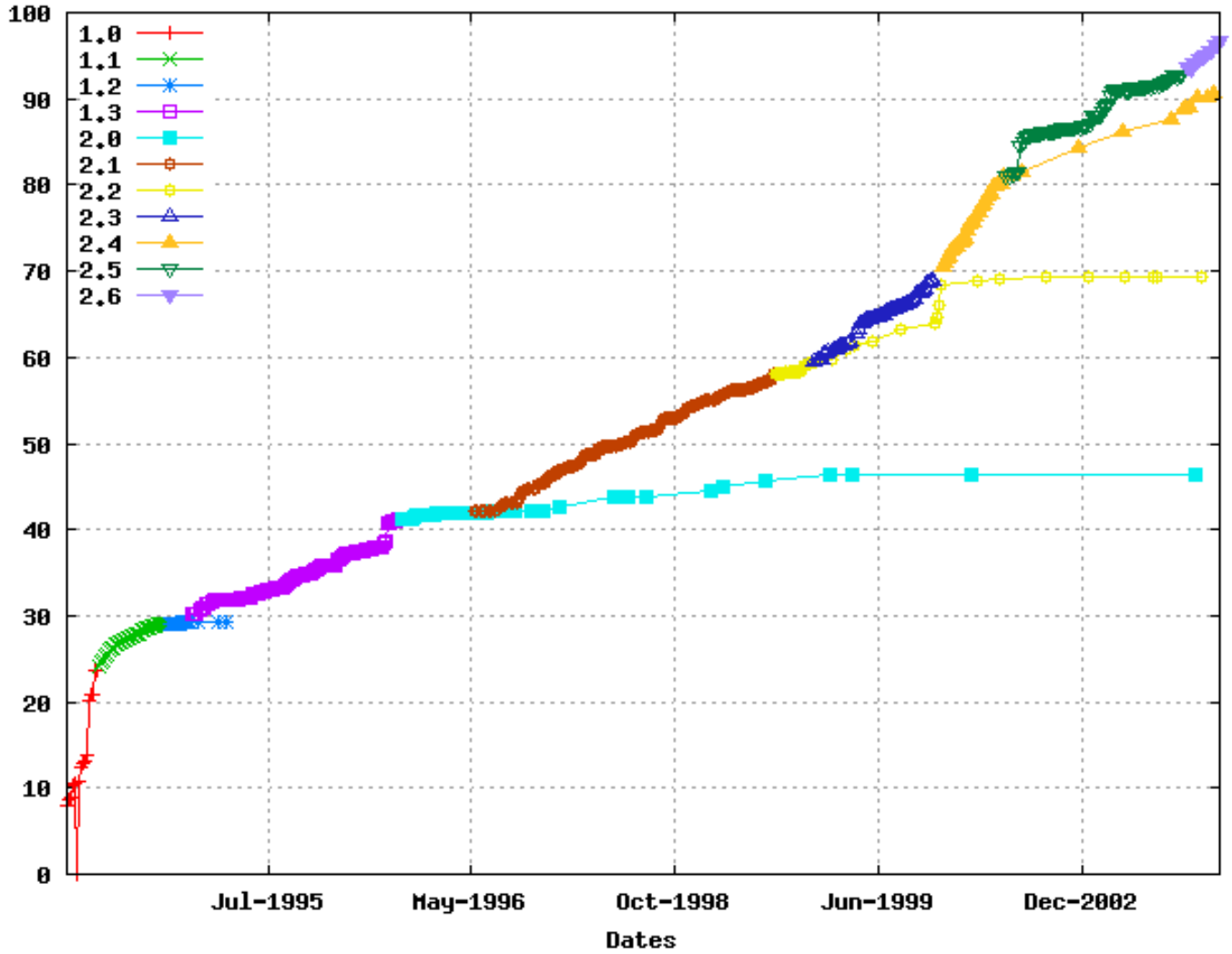
Evolución de Linux (SLOCs)



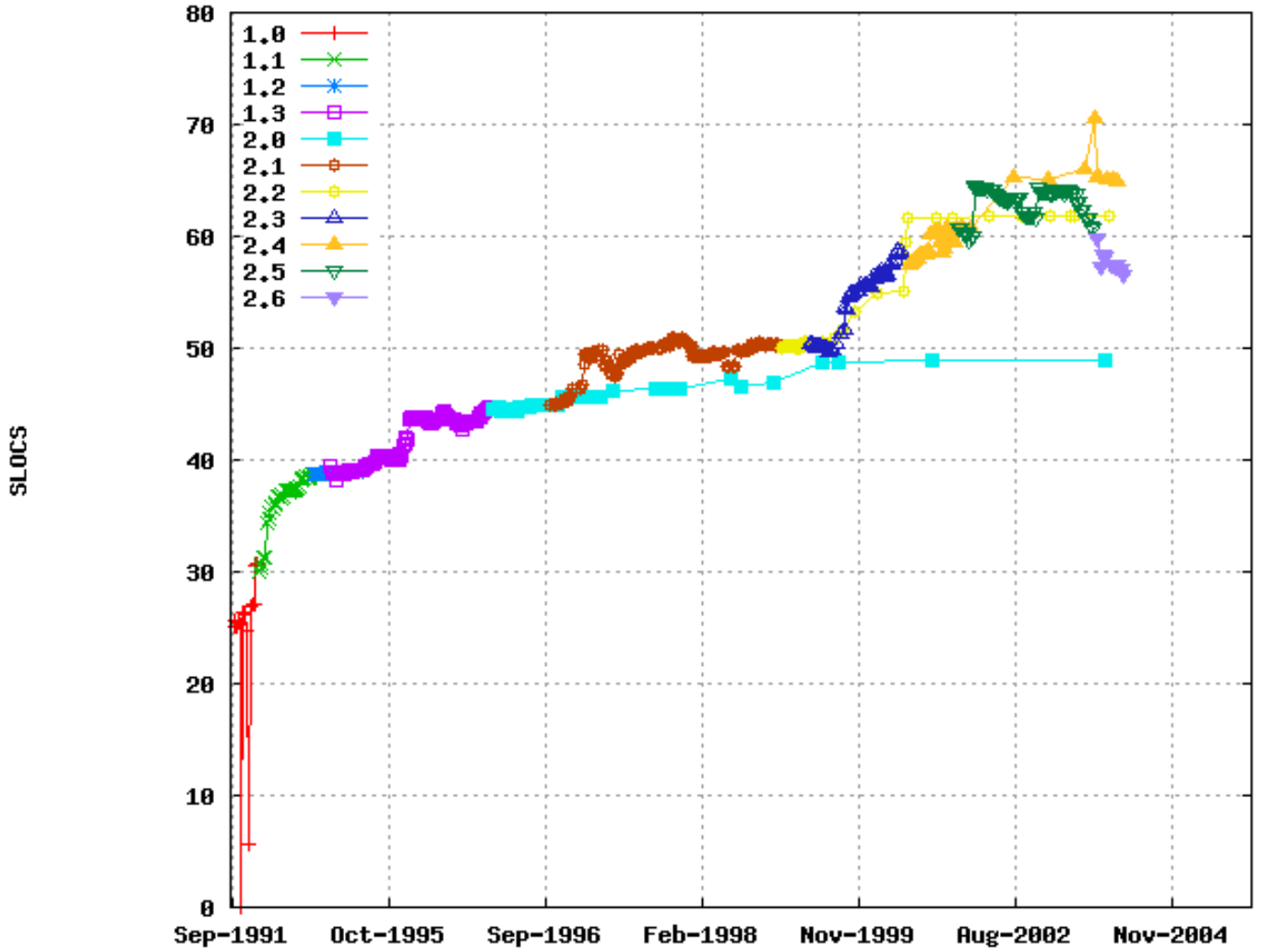
Evolución de Linux (coste según COCOMO)



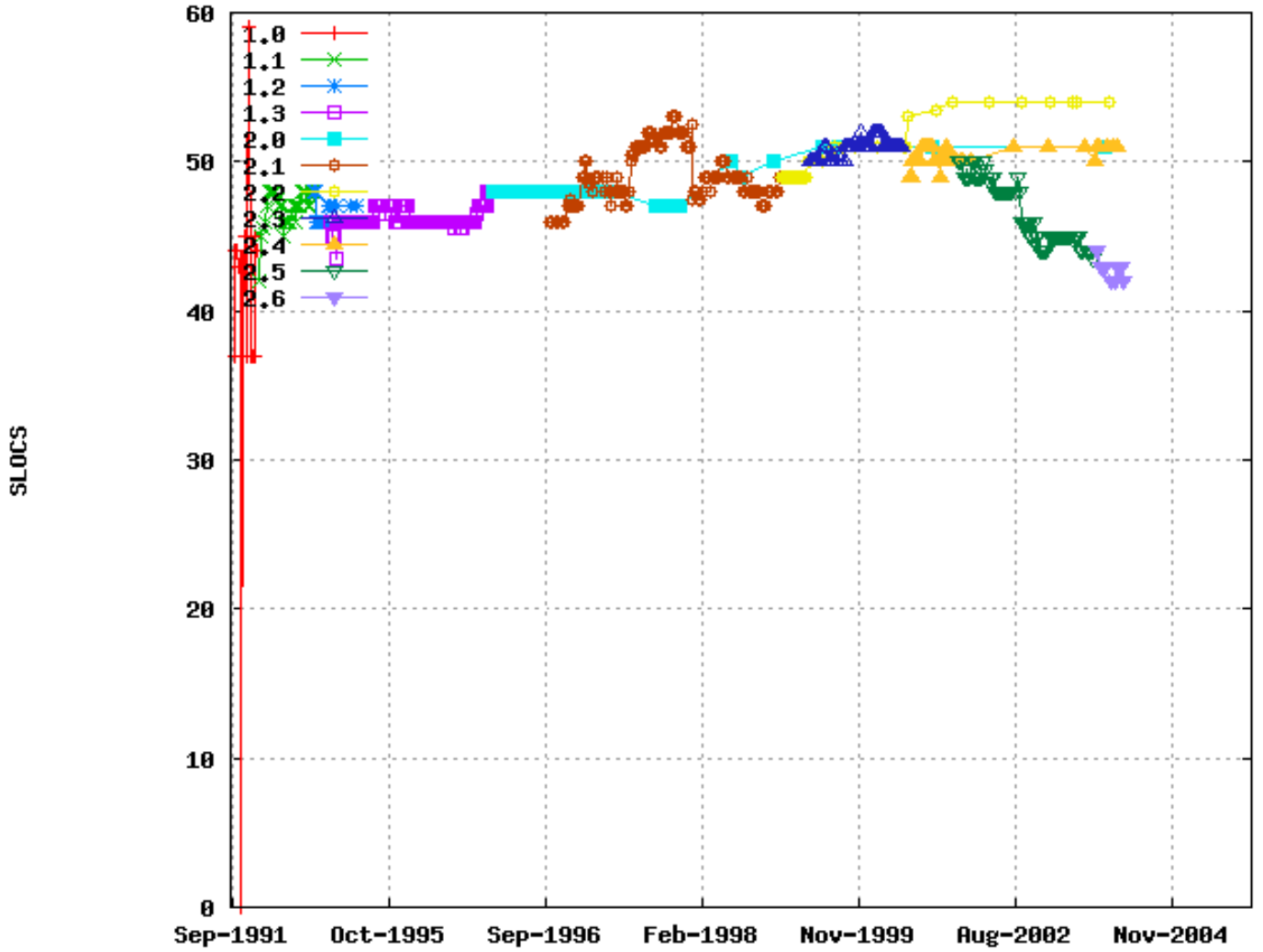
Evolución de Linux: tiempo de desarrollo (COCOMO)



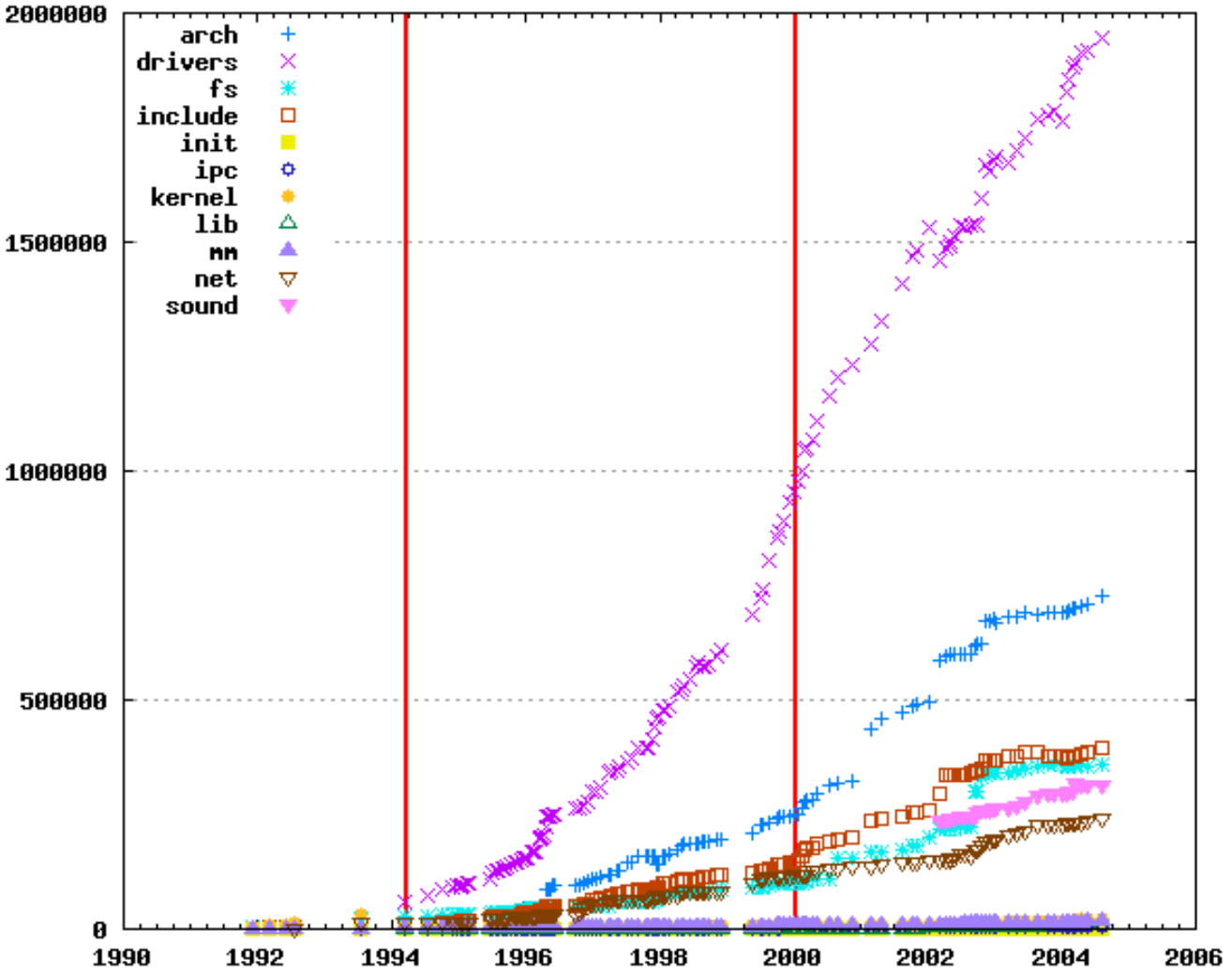
Evolución de Linux: media tamaños .h



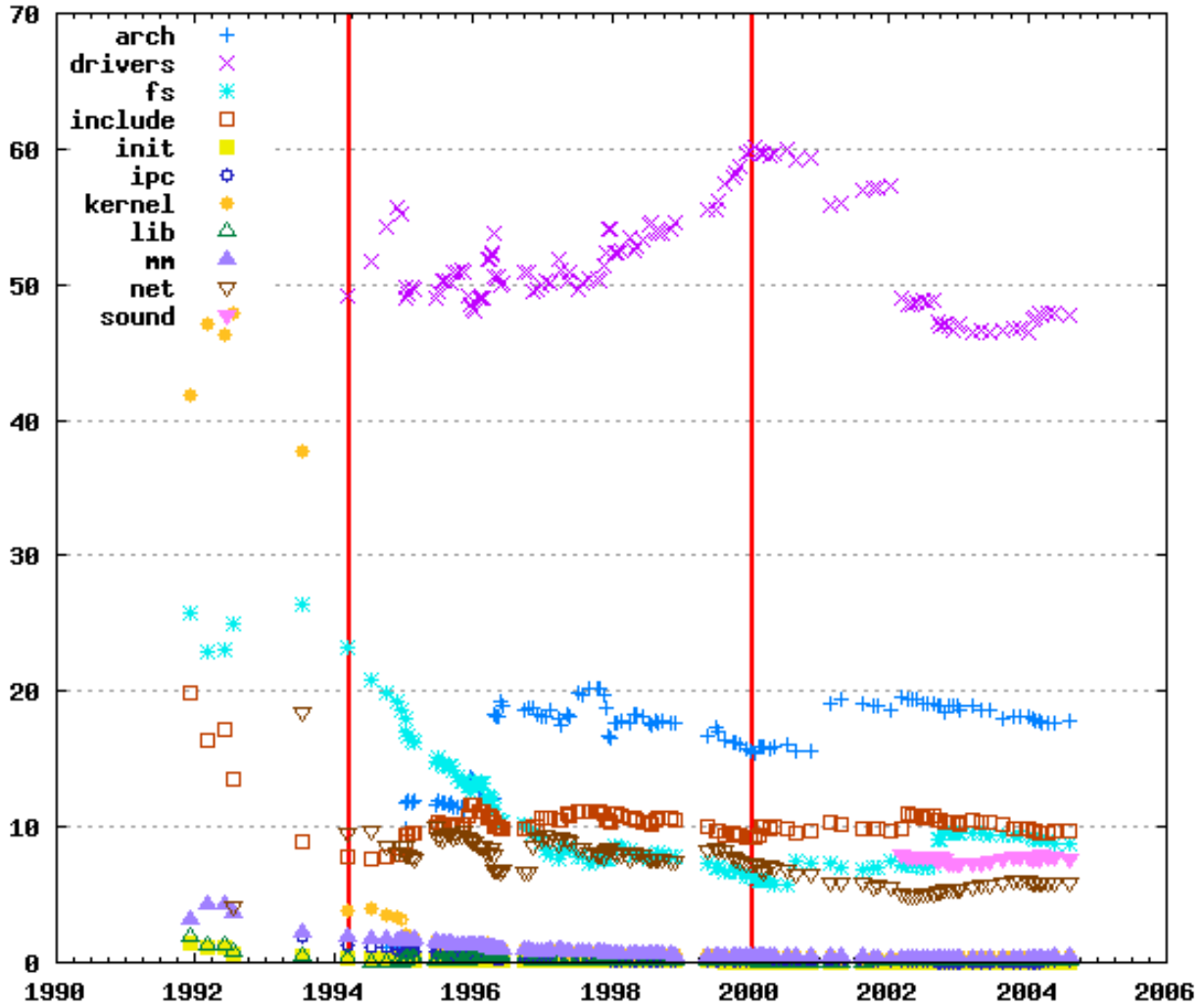
Evolución de Linux: mediana tamaños .h



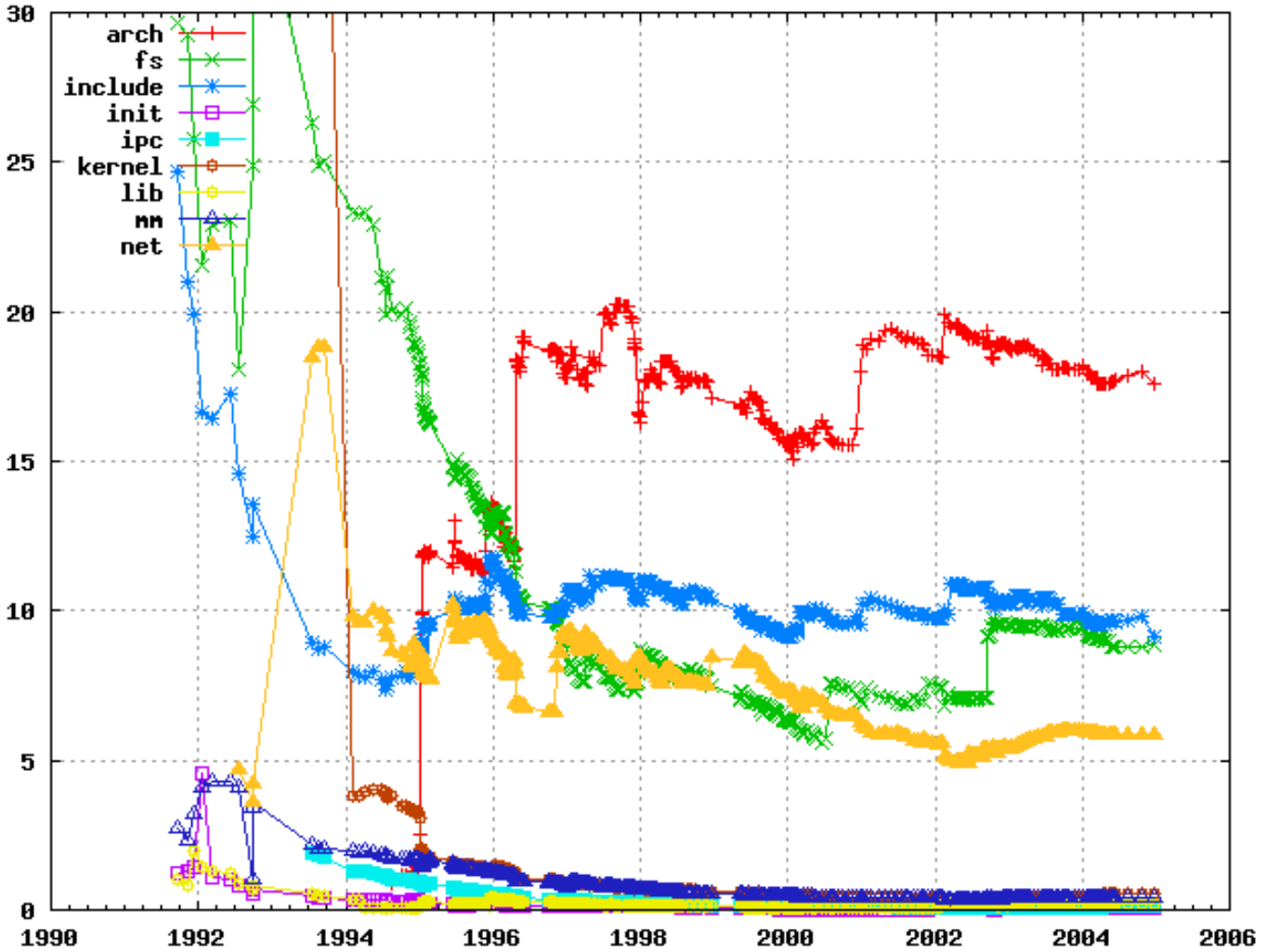
Evolución de Linux: Subsistemas



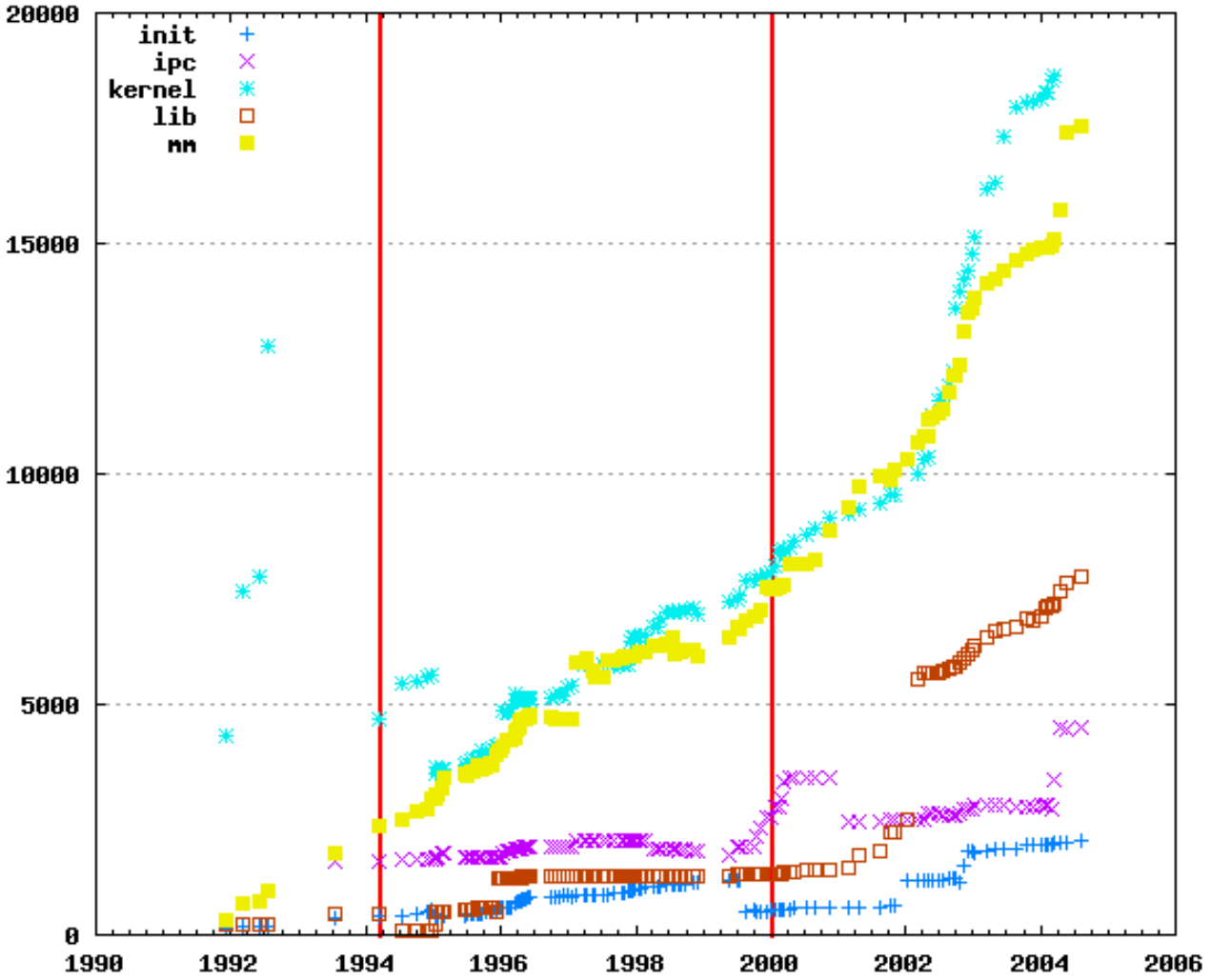
Evolución de Linux: subsistemas (%)



Evolución de Linux: subsistemas, sin drivers



Evolución de Linux: subsistemas pequeños “principales”



Evolución de Linux: Conclusiones

- Crecimiento superlineal, usando diversas métricas.
  - De un sistema grande y complejo.
  - Con modelo de desarrollo “abierto”
    - Grandemente cooperativo
    - Desarrolladores geográficamente distribuidos
    - Participando muchas veces con su tiempo libre
- Examinando subsistemas (como sugiere Gall)
  - Análisis “caja negra” no es suficiente

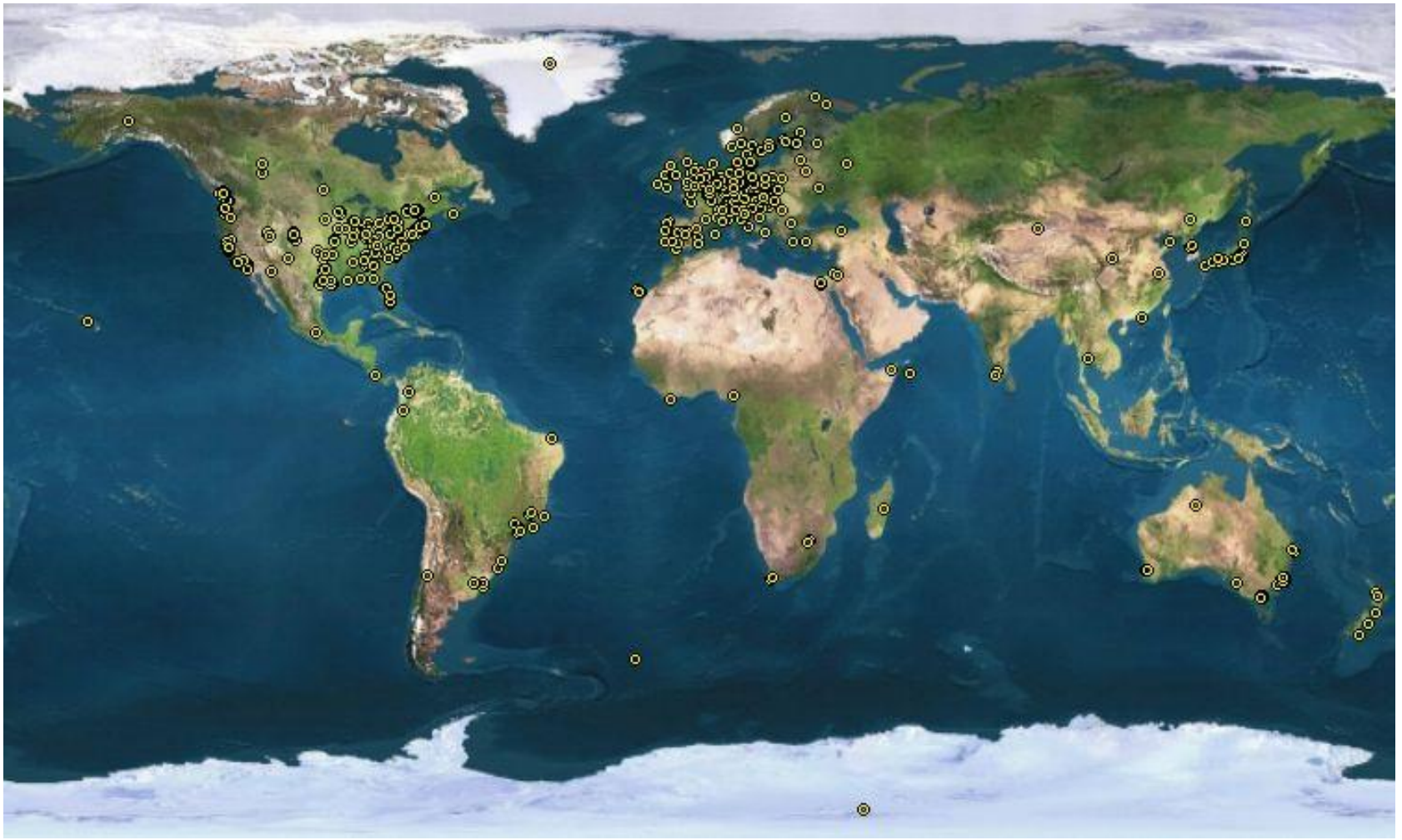
Resumen

1. Introducción al software libre
2. La Catedral y el Bazar
3. Contando código (patatas, etc)
4. Evolución
5. Dinámica de los desarrolladores 
6. Esfuerzo
7. Conclusiones

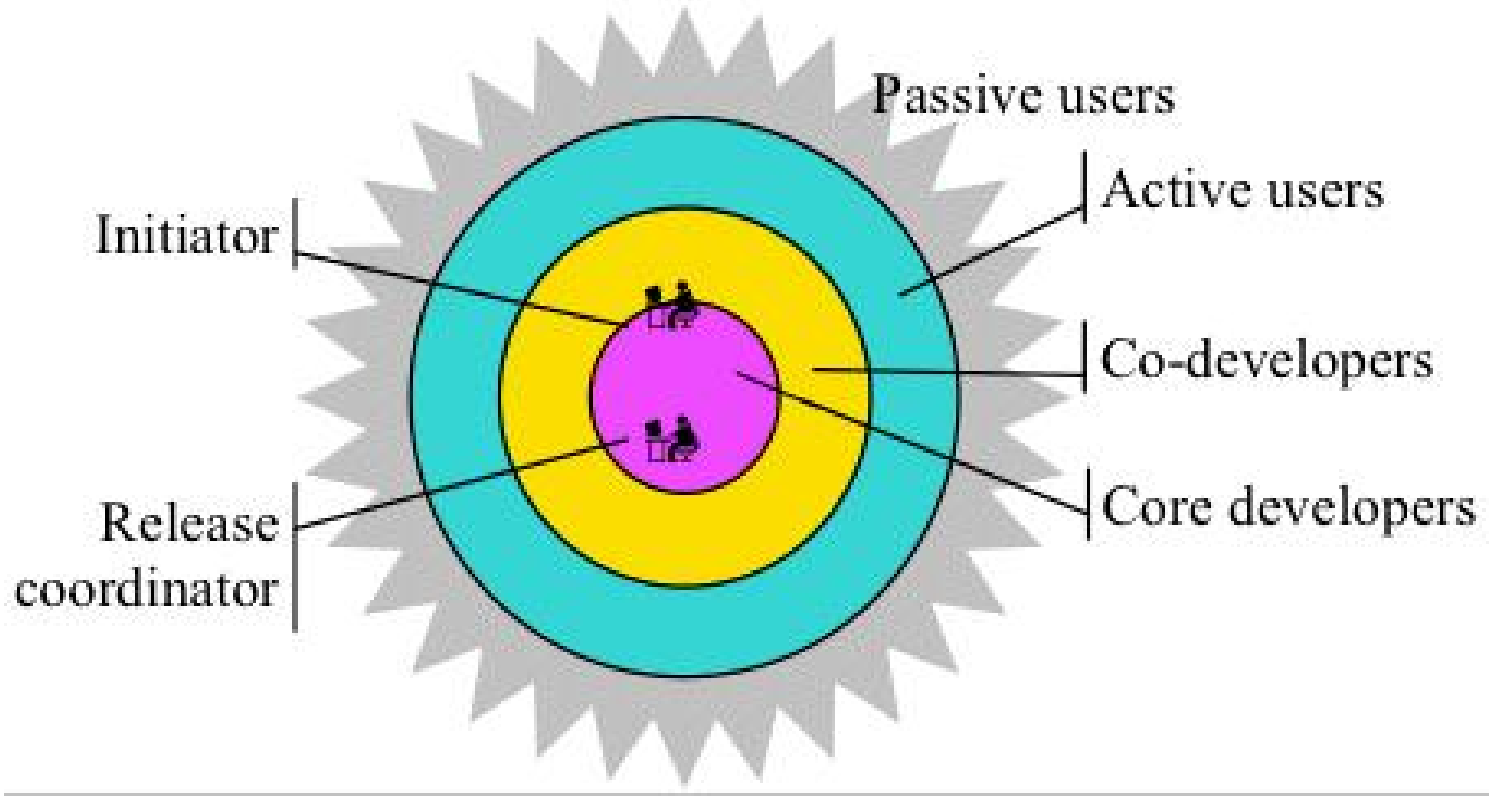
Desarrolladores

- Responder preguntas:
  - ¿Cómo se entra en un proyecto de software libre?
  - ¿Cómo se reparten el trabajo?
  - ¿Qué hay del liderazgo?
  - ¿Algo más?

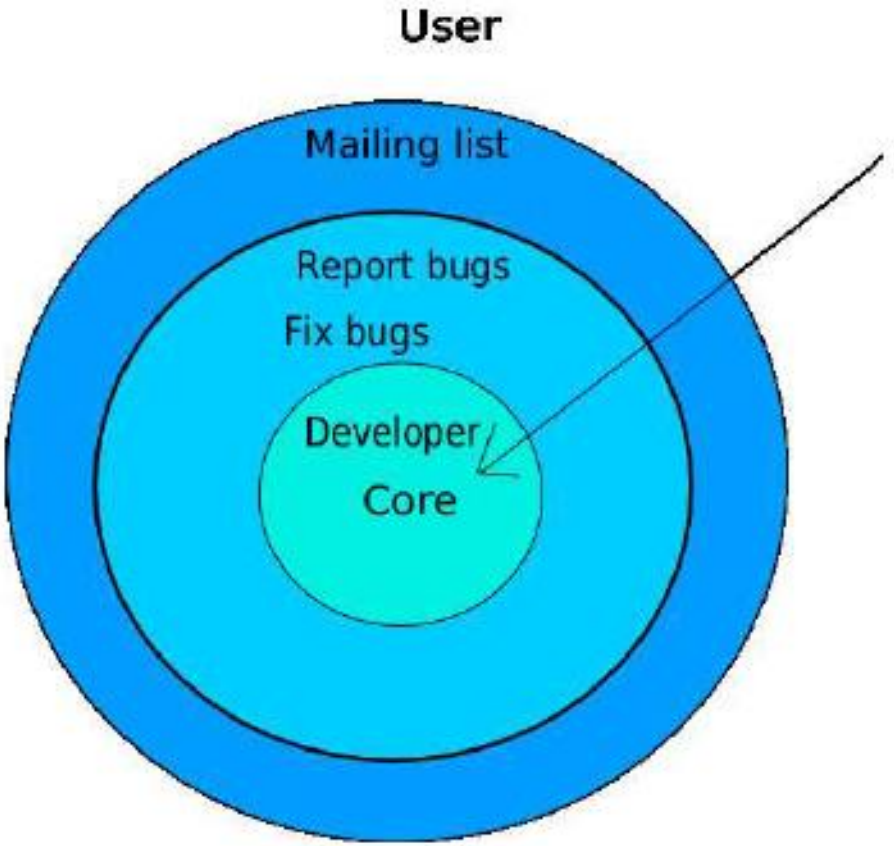
Un vistazo a los desarrolladores de Debian



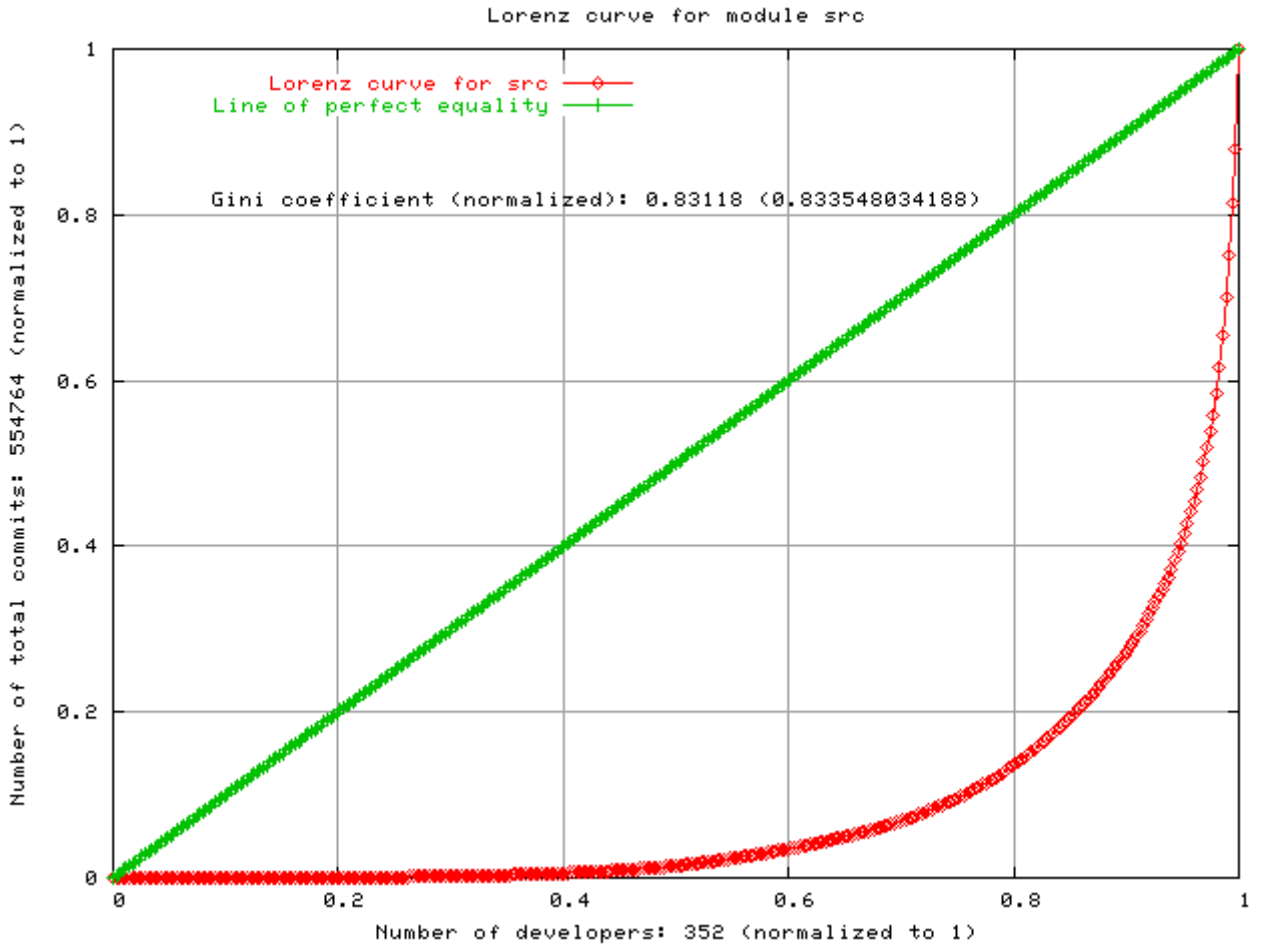
El modelo “cebolla”



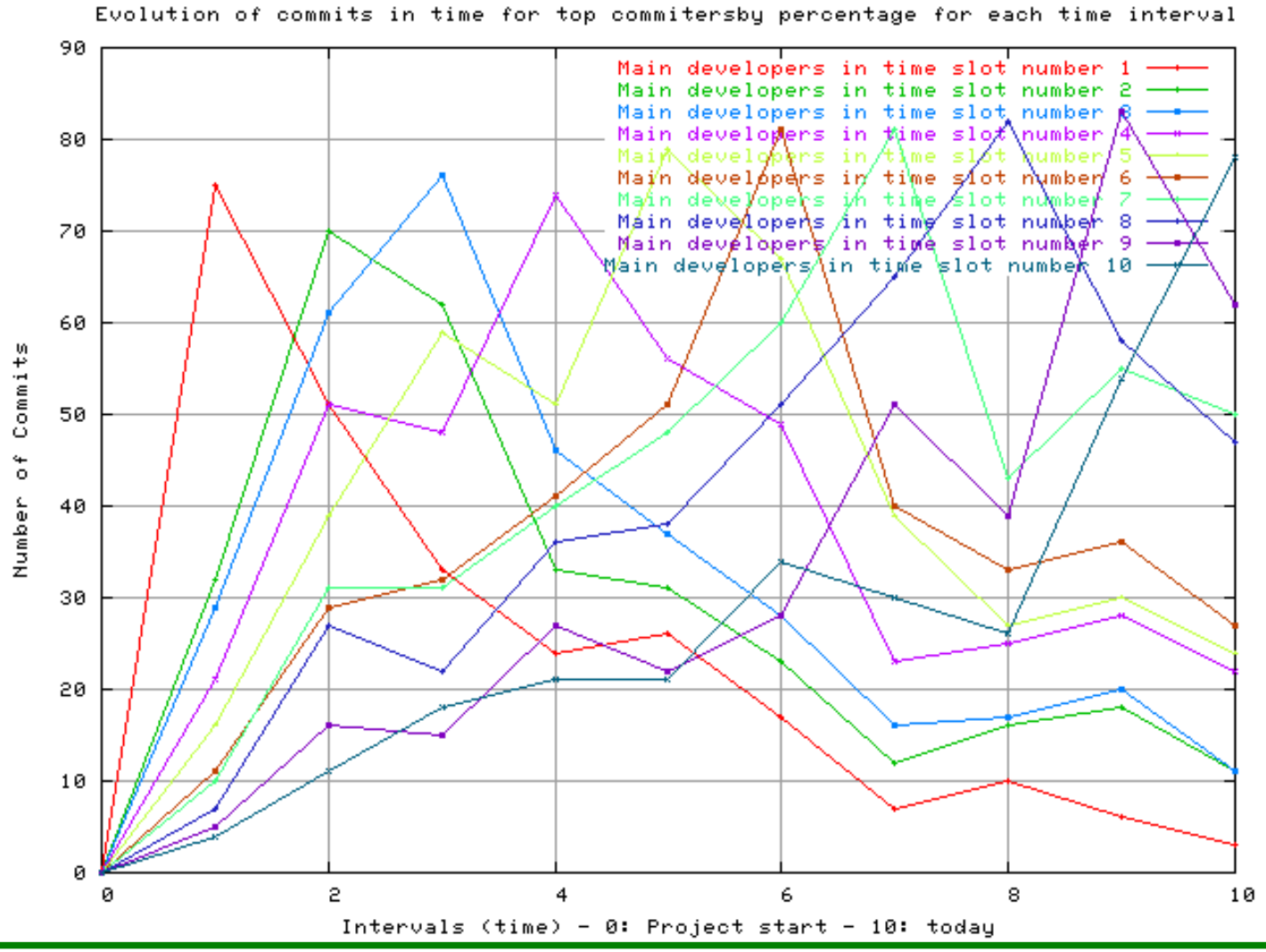
Un modelo contrastado



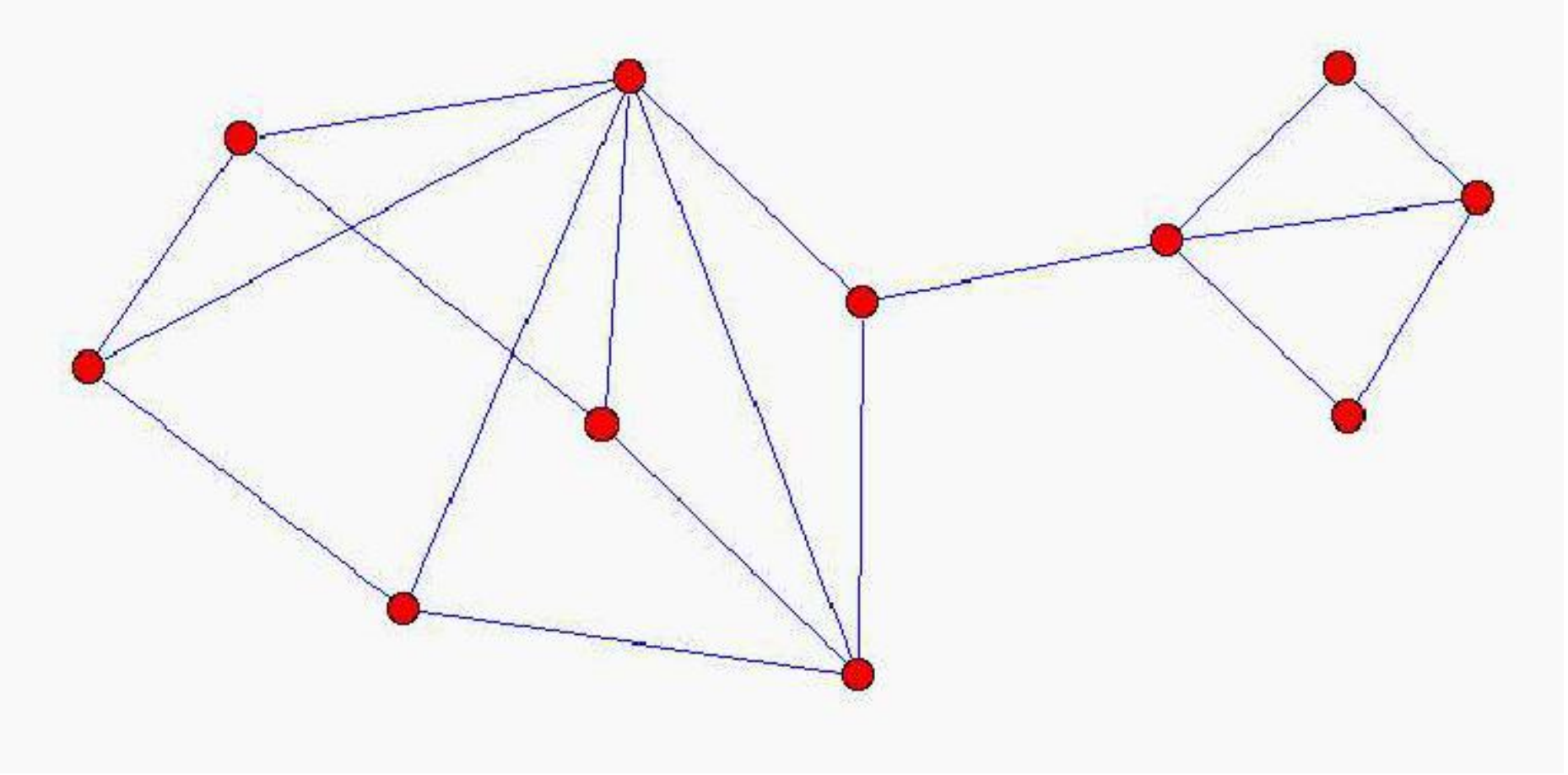
El reparto del trabajo



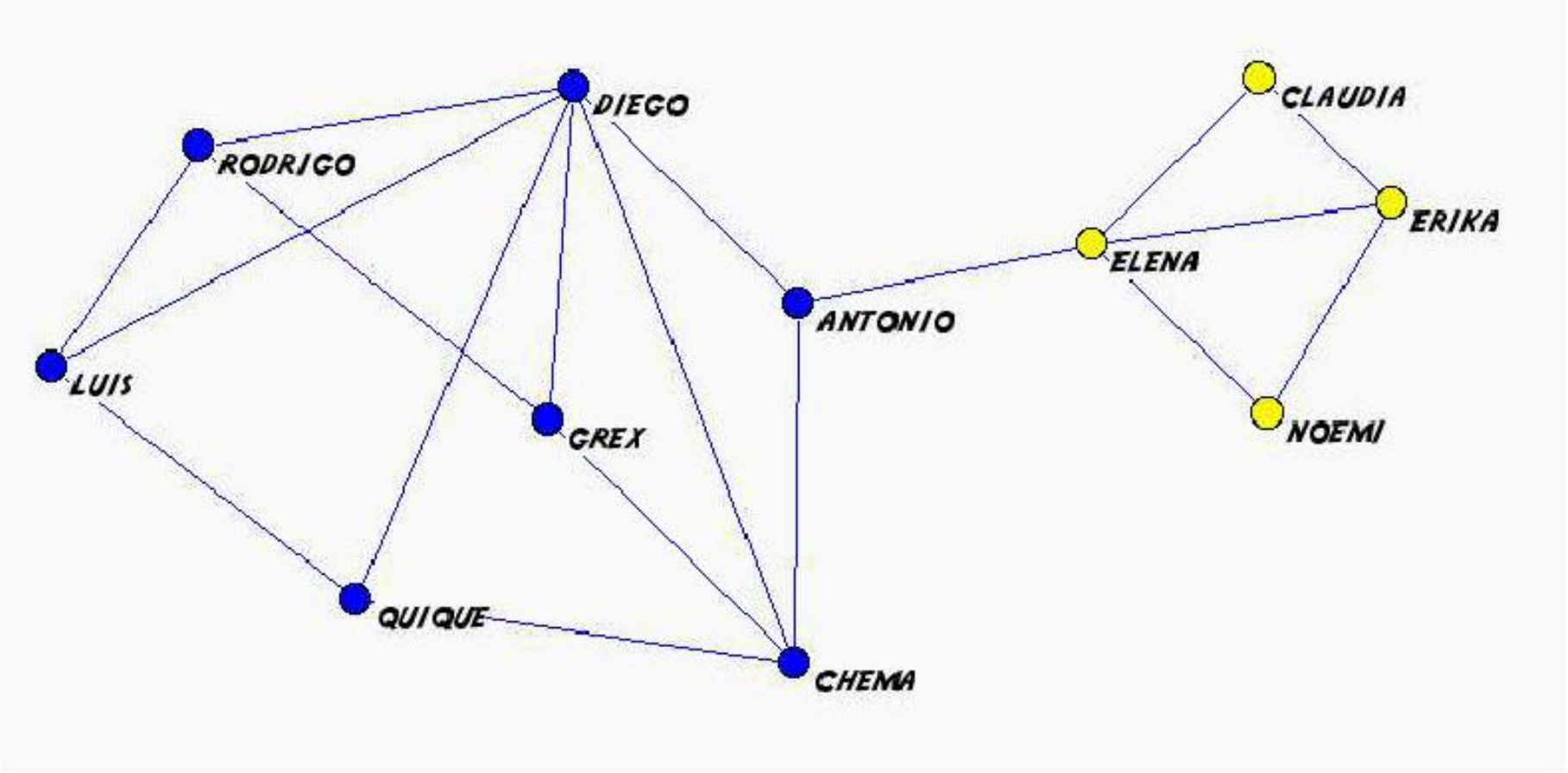
Sobre el liderazgo



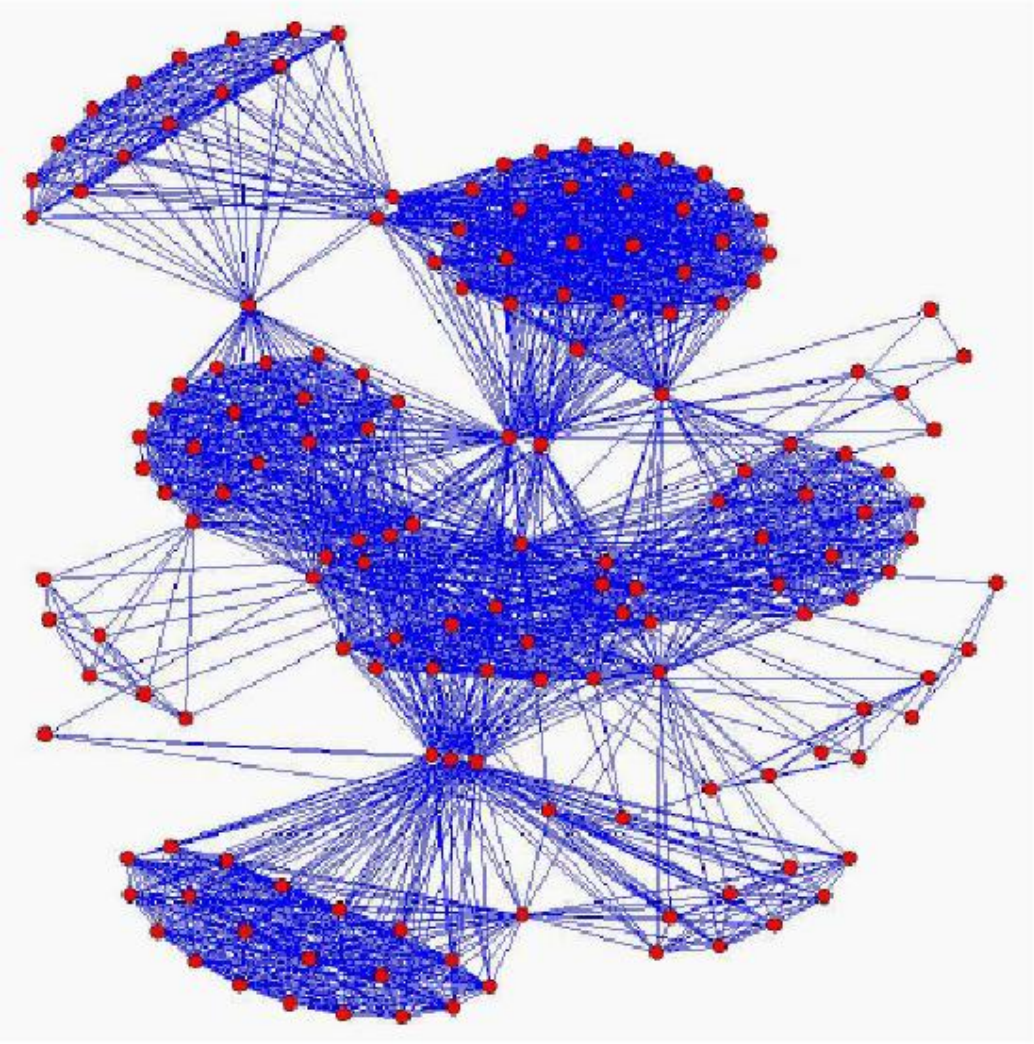
Algo más...



Algo más...


DIEGO
RODRIGO
CLAUDIA
ERIKA
ELENA
ANTONIO
LUIS
GREX
NOEMI
QUIQUE
CHEMA

Linux 1.0



Conclusiones

- Modelo de entrada en proyectos.
- Colaboración sigue Ley de Pareto.
- Liderazgo en renovación.
- No tan simple como en un “bazar”

Resumen

1. Introducción al software libre
2. La Catedral y el Bazar
3. Contando código (patatas, etc)
4. Evolución
5. Dinámica de los desarrolladores
6. Esfuerzo 
7. Conclusiones

¿Cómo podemos aplicar todo esto?

- ¿Podemos ayudarnos de estos análisis para mejorar algo en Ingeniería del Software?
- Estimación de esfuerzo
  - En el software libre, los estudios son escasos.
  - Limitados a estudios relacionados con tamaño de proyectos (líneas de código).
  - ¿alguna mejora?

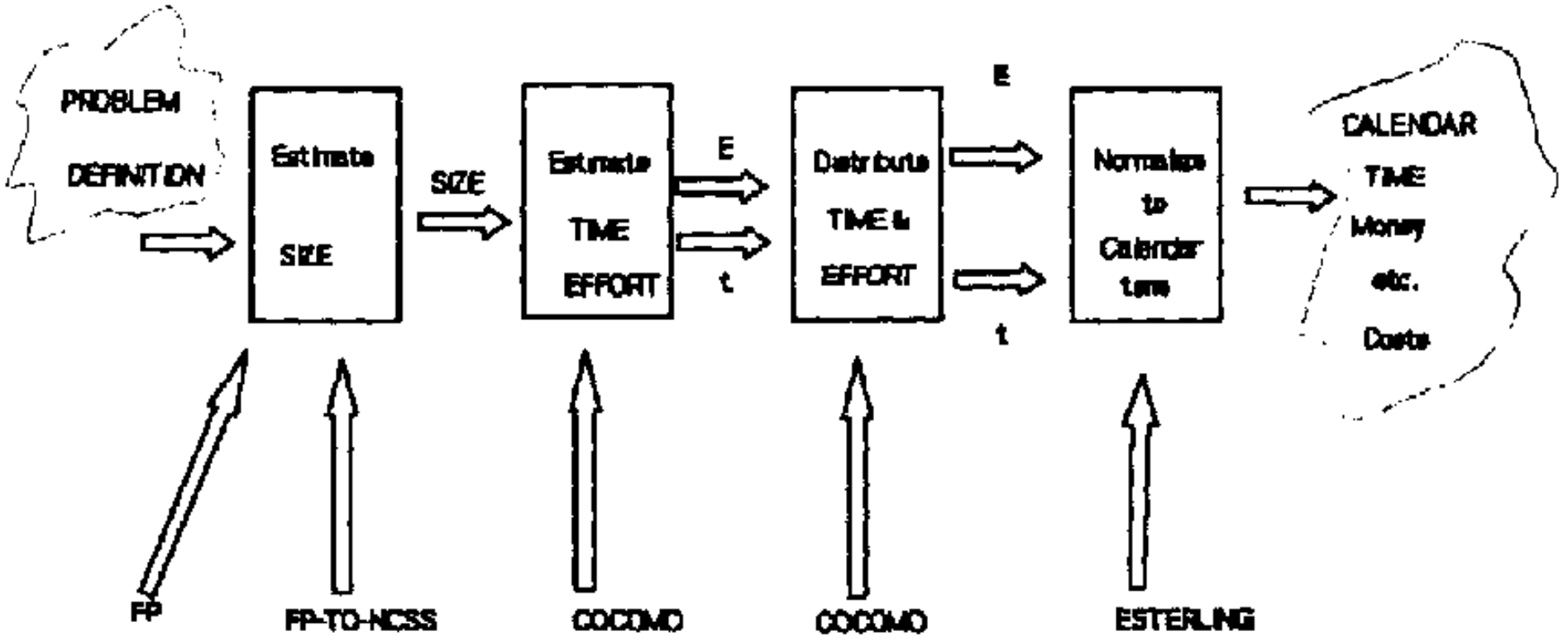
Estimación “clásica”

- Clasificación de [Conte, 1986].
  - Modelos históricos/experimentales
    - Expertos, historia, analogías...
  - Modelos estadísticos
    - Regresiones
  - Modelos teóricos
    - Suposiciones y leyes matemáticas
  - Modelos compuestos
    - Mezcla de los anteriores

Estimación “clásica”

- Ejemplos:
  - Regresión: $E = A + B * S^C$
  - COCOMO: $E = m * KLOC^n$
- Métricas utilizadas:
  - SLOC: Líneas de código, físicas o lógicas...
  - FP: Puntos de función: información acerca de actividades planificadas nos da información acerca de los FP.
- Usando técnicas como éstas, se puede estimar el coste del proyecto partiendo de la definición del problema o los requisitos.

Ejemplo de metodología de estimación de coste



(Arifoglu, 1993)

Estimación en I.S. Libre

- Estimaciones clásicas:
  - Basadas en una métrica (normalmente SLOC)
- Esfuerzo en software libre. El tiempo se invierte en:
  - Codificar
  - Escribir correos
  - Gestionar bugs
  - ... etc ...

Costes en software libre

- Contribuciones de la “plantilla”:

$$coste_{internos} = \sum coste_{interno}$$

- Contribuciones de la “comunidad”

$$coste_{externos} = \sum coste_{colaborador\,externo}$$

- Coste total

$$coste = coste_{internos} + coste_{externos}$$

Coste y esfuerzo

- Cada coste es función de esfuerzo:

$$coste = f(esfuerzo)$$

- Esfuerzo es función de la actividad:

$$esfuerzo = g(actividad) \Rightarrow coste = f(g(actividad))$$

Coste y esfuerzo

- Dos problemas:
  - ¿Cómo medir las actividades?
  - ¿Cómo establecer las funciones f y g?
- En software libre podemos medir las actividades en grano más fino.

La propuesta

- Si consideramos las medidas de actividad en código, correo y gestión de erratas, podríamos decir que:

$$esfuerzo = g(actividad) = a.h(CVS) + b.j(ML) + c.k(BTS) + d$$

- Pero aun nos queda determinar: funciones h, j, k; así como constantes a, b, c y d.
  - La traza nos puede ayudar a determinarlo.
- Esto no es nuevo. Por ejemplo, COCOMO sería función de “CVS” (SLOC):

$$esfuerzo = m.KLOC^{n} = a.h(CVS)$$

¿Conclusiones?

- Son solo ideas.
- Largo camino por recorrer:
  - Medición de actividades.
  - Traza de esfuerzo individual.
  - Establecimiento del modelo: *constantes* y *funciones*.

Resumen

1. Introducción al software libre
2. La Catedral y el Bazar
3. Contando código (patatas, etc)
4. Evolución
5. Dinámica de los desarrolladores
6. Esfuerzo
7. Conclusiones 

Conclusiones

- “Ingeniería del software libre”
  - No es muy diferente a la tradicional
  - Pero aporta gran cantidad de datos para la mejora de la I.S. tradicional.
- Paralelamente, estudiando el software libre
  - Se justifica su uso
    - ¿Más rentable? ¿más económico? ¿TCO y qué más?

El Grupo Libresoft

- Grupo Libresoft
  - Comienza en torno a 1995. Grupo GSyC de U. Carlos III, Grupo de interés “SoBre”.
  - Miembros:
    - 2 Doctores.
    - 1 Gestor contratado.
    - 1 Ayudante no doctor.
    - 5 doctorandos.
    - Contratados, becarios ...

El Grupo Libresoft

- Proyectos de investigación:
  - Terminados: CALIBRE (EU), Flossworld (EU), Flossimpact (EU), DeSobre (ES), Morfeo (ES), Tree (ES).
  - En curso: Qualoss (EU), Flossmetrics (EU), SobreTodo (ES), Vulcano (ES), Infojobs (ES), CENATIC (ES).
  - Industriales: Qualipso (EU), OSO-R (EU)

Libresoft, Hoy

http://www.libresoft.es/

¿Preguntas?

Gracias :-)

http://www.libresoft.es/