

Dinamica delle innovazioni nei software open-source: analisi dell'archivio di GitHub

A. Sciortino, V. Sorichetti, C. Valensise, V. Venturini

28 febbraio 2016

Sommario

I software open-source (OSS) sono una presenza fondamentale nel panorama dell'informatica contemporanea. A partire dai dati messi a disposizione dalla piattaforma di sviluppo collaborativo GitHub, attraverso gli strumenti e i metodi della fisica dei sistemi complessi si vuole caratterizzare il processo di evoluzione degli OSS. Si verifica inoltre che le previsioni dei modelli di dinamica delle innovazioni sono in larga parte rispettate. L'analisi mette dunque in evidenza la presenza di dinamiche e correlazioni non banali all'interno di questo ambiente, aprendo la strada a ulteriori studi.

Indice

1 Introduzione	2
2 Metodi	3
3 Analisi globale	4
3.1 Mappa degli eventi	4
3.2 Attività annuale	8
3.3 Rete delle repository	8
3.3.1 Rete delle repository con basso numero di eventi l'anno	9
3.3.2 Rete delle repository con alto numero di eventi l'anno .	11
4 Dinamica delle innovazioni	13
4.1 Analisi delle singole repository	13
4.1.1 Definizione di <i>evento di innovazione</i>	13
4.1.2 Distribuzione frequency-rank e numero di elementi distinti	13
4.1.3 Intervalli temporali	14
4.1.4 Entropia	14
4.2 Analisi estesa a più repository	18
5 Conclusioni	21
Appendice A	
Modelli matematici di dinamica delle innovazioni	23

1 Introduzione

I software open-source (OSS), già studiati diffusamente in letteratura ([1, 2, 3]), sono ormai una presenza sempre più importante nel panorama dell'informatica contemporanea: basti pensare al sistema operativo GNU/Linux e al suo parente Android, o a Mozilla Firefox, Apache HTTP Server, LibreOffice, Vim, GIMP, VLC... Software usati da milioni di persone nel mondo, che possono ormai rivaleggiare con le migliori tra le controparti closed-source.

Dai loro parenti closed-source, gli open-source si differenziano anche e soprattutto nel processo di produzione e aggiornamento. Mentre nel caso di un software proprietario lo sviluppatore e l'utilizzatore sono due persone ben distinte, nel caso degli OSS la comunità degli utilizzatori e quella degli sviluppatori si sovrappongono in larga parte. Ciò porta naturalmente a un processo di evoluzione completamente diverso da quello che riguarda un software proprietario: un processo di evoluzione che rappresenta il banco di prova ideale per mettere alla prova le previsioni dei modelli di dinamica delle innovazioni ([4]).

La miglior banca dati da cui partire per studiare tale processo è l'archivio di GitHub[6]. GitHub è la più autorevole e utilizzata piattaforma per lo sviluppo collaborativo di software. Basata sul software open-source Git (ideato e sviluppato dal padre di Linux, Linus Torvalds) e lanciata nel 2008 da Tom Preston-Werner, Chris Wanstrath e PJ Hyett, conta attualmente più di trentun milioni di repository (cartelle) e più di dodici milioni di utenti.

Oltre ai codici sorgente e alla documentazione relativa a ogni software, GitHub contiene anche i registri delle attività relative a ogni progetto. Ci sono 25 tipi di eventi possibili:

1. **CommitCommentEvent**: viene creato un commento
2. **CreateEvent**: viene creata una repository
3. **DeleteEvent**: viene eliminato un branch di una repository o una sua tag
4. **DeploymentEvent**: viene aggiornato il codice
5. **DeploymentStatusEvent**: stato dell'aggiornamento
6. **DownloadEvent**: viene eseguito un download (deprecato)
7. **FollowEvent**: due utenti iniziano a seguirsi
8. **ForkEvent**: una repository viene divisa in due sotto progetti
9. **ForkApplyEvent**: viene modificata la lista delle fork (deprecato)
10. **GistEvent**: viene creata una gist (versione più compatta di una repository)
11. **GollumEvent**: viene creata o aggiornata una pagina wiki
12. **IssueCommentEvent**: viene creato un commento su una richiesta di pull o su una issue
13. **IssuesEvent**: viene creata una issue (nota che avverte di un problema di un certo progetto)
14. **MemberEvent**: un utente diventa collaboratore di una repository
15. **MembershipEvent**: un utente viene aggiunto/rimosso ad un team
16. **PageBuildEvent**: viene costruita una pagina su GitHub
17. **PublicEvent**: una repository privata viene resa pubblica

- 18. **PullRequestEvent**: viene eseguita una richiesta di pull
- 19. **PullRequestReviewComment Event**: viene aggiunto un commento ad una richiesta di pull
- 20. **PushEvent**: viene eseguito un push
- 21. **ReleaseEvent**: viene pubblicata una release
- 22. **RepositoryEvent**: viene creata una repository
- 23. **StatusEvent**: viene modificato lo stato di una repository
- 24. **TeamAddEvent**: un team inizia a lavorare su una repository
- 25. **WatchEvent**: una repository riceve una stella (segno di apprezzamento)

Tra gli eventi più significativi al fine di quantificare la crescita di un progetto vi sono Fork, Pull Request e Push. La Fork consiste nel creare una copia di una repository appartenente a un altro utente sul proprio account, in modo da poter applicare delle modifiche al progetto senza modificare l'originale. Una volta effettuate le modifiche, è possibile inviare una Pull Request, chiedendo ai responsabili della repository originale di inserire le modifiche nel progetto (*merge*). L'azione di inviare le modifiche effettuate localmente a una repository remota, in modo che gli altri utenti vi possano accedere, è detta Push.

2 Metodi

GitHub permette di accedere ai suoi dati salvati in formato JSON attraverso richieste HTTP o una sua API, il tutto attraverso GitHub Archive.

Il tipo di dati messi a disposizione è la sequenza di eventi apparsi su GitHub in un certo periodo di tempo (giorno, mese, anno), a partire dal 2011. Ogni volta che qualcuno esegue un'azione su GitHub (e.g. un evento di tipo Fork), tale evento con tutte le sue caratteristiche (nome dell'utente, repository su cui è stata eseguita l'azione, tipo di azione, orario dell'evento...) viene codificato in formato JSON e reso scaricabile via HTTP.

Attraverso uno script in Python, è stato possibile scaricare uno dopo l'altro gli eventi disponibili sull'archivio di GitHub, salvando su disco fisso solo quelli necessari per l'analisi.

Inoltre, i dati tra il 2011 e il 2015 sono anche conservati in un database SQL sui server di Google e accessibile tramite query SQL attraverso il servizio BigQuery[7] di Google Project, che però limita la quantità di eventi scaricabili per ricerca a 15000. Visto tale limite, questo metodo è stato utilizzato per scaricare i dati relativi all'attività mensile delle singole repository.

Vista la maggiore velocità delle query SQL rispetto l'approccio *brute force* di Python con HTTP e soprattutto data la possibilità di SQL di selezionare solamente gli eventi e le relative informazioni cui si è interessati, questo approccio è stato inoltre utilizzato per le analisi preliminari e per individuare le caratteristiche salienti dei dati accessibili.

Le analisi su quantità di dati più corpose sono state invece effettuate scaricando eventi relativi all'intero anno 2015 attraverso HTTP e Python.

```

select type, actor_id, created_at
from githubarchive:month.201501
where repo_name='docker/docker'
order by created_at

```

Figura 1: Esempio di ricerca SQL effettuata su Google BigQuery

3 Analisi globale

3.1 Mappa degli eventi

Per avere un’idea più chiara dello sviluppo temporale dell’archivio dal punto di vista delle repository e degli eventi, è stata considerata l’attività del mese di Gennaio 2015, studiando l’occorrenza dei diversi eventi durante il mese. Il risultato è riportato nella figura 2. Gli eventi predominanti (un ordine di grandezza di più rispetto a tutti gli altri) sono i Push event. Gli eventi più tipici seguono questo ordinamento:

- | | |
|----------------------|----------------------------------|
| 1. PushEvent | 8. DeleteEvent |
| 2. CreateEvent | 9. PullRequestReviewCommentEvent |
| 3. IssueCommentEvent | 10. GollumEvent |
| 4. WatchEvent | 11. CommitCommentEvent |
| 5. IssuesEvent | 12. MemberEvent |
| 6. PullRequestEvent | 13. ReleaseEvent |
| 7. ForkEvent | 14. PublicEvent |

Si nota immediatamente che l’attività generale dell’archivio si concentra prevalentemente nei giorni feriali. È ragionevole presumere che le stesse relazioni quantitative tra gli eventi si ripresentino uguali per gli altri mesi.

Il secondo aspetto interessante di queste analisi mostra che la distribuzione degli eventi rispetto alle repository non è uniforme. In altri termini, risulta che gli eventi tipici di una repository più recente sono sostanzialmente diversi rispetto agli eventi di una repository presente su GitHub da più tempo. La distinzione tra repository vecchie e nuove è effettuata attraverso l’id della repository stessa: più questo valore è grande, più il progetto è nuovo.

Di conseguenza si può costruire un grafico per mappare l’attività di un giorno, costituita da circa $5 \cdot 10^5$ di eventi, in cui sull’asse delle ascisse è riportato il tempo, mentre sulle ordinate, in ordine crescente, si trovano le varie repository. Infine è stato adottato un codice di colore per distinguere tra i diversi tipi di evento.

Nella figura 3 è riportata la mappa relativa al 14 Gennaio 2015. Sull’asse delle ascisse sono stati segnati i nomi di tre repository a scopo esemplificativo.

Salta immediatamente all’occhio che per repository più recenti l’evento Push è dominante, mentre per repository più vecchie anche gli altri tipi di eventi risultano essere altrettanto rilevanti.

In altre parole, risulta che per repository più nuove il Push domina il fondo di tutti gli altri eventi, mentre questo non succede per progetti più vecchi. A conferma di ciò si può analizzare la mappa relativa ai soli ForkEvent o CreateEvent, che mostrano una distribuzione più uniforme (figura 5); si nota

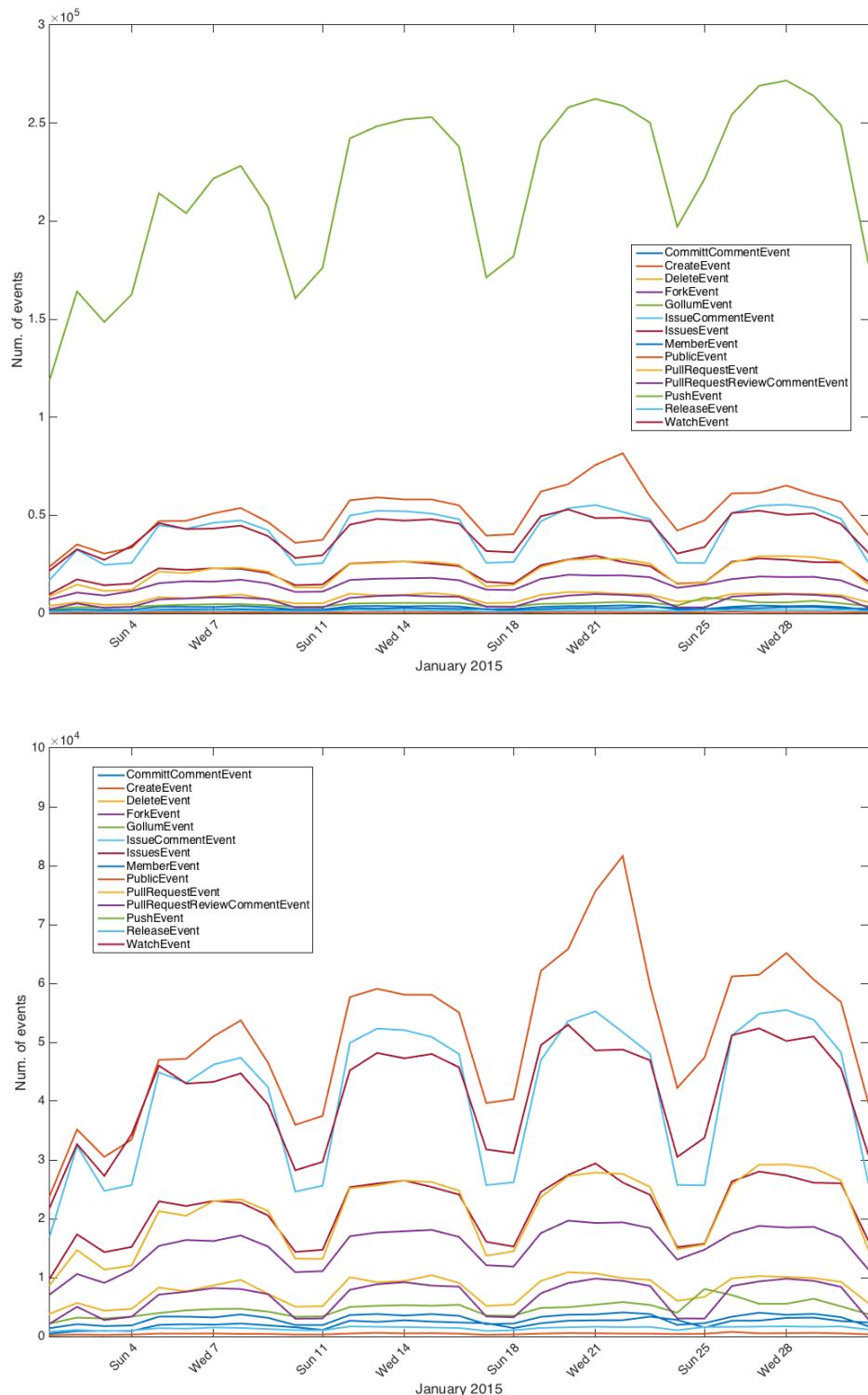


Figura 2: Attività su GitHub nel mese di Gennaio 2015

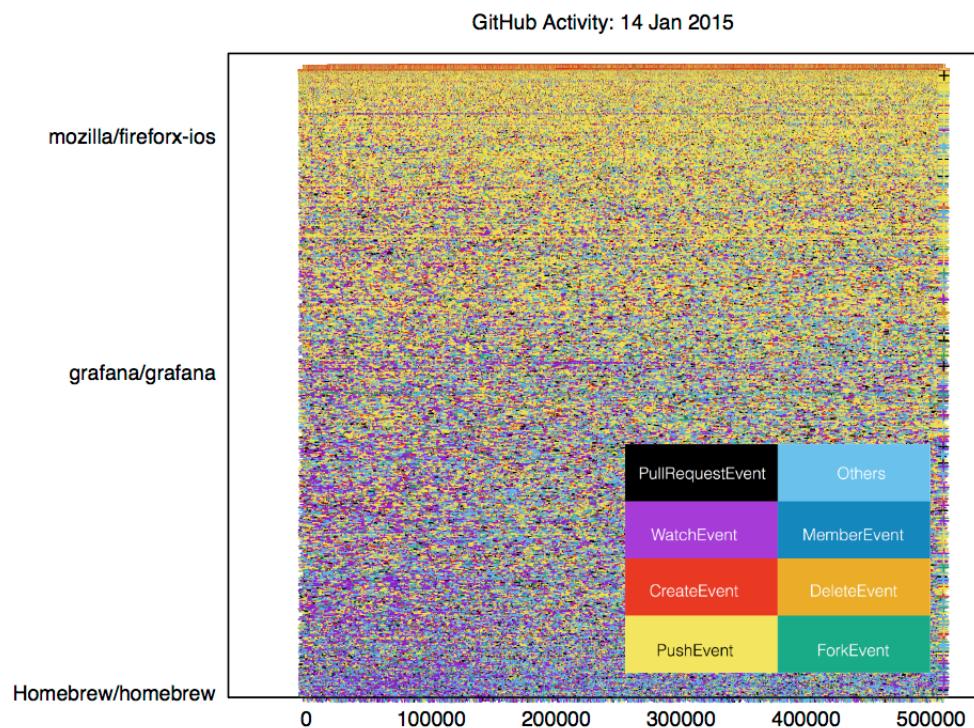


Figura 3: Mappa del 14/1/15

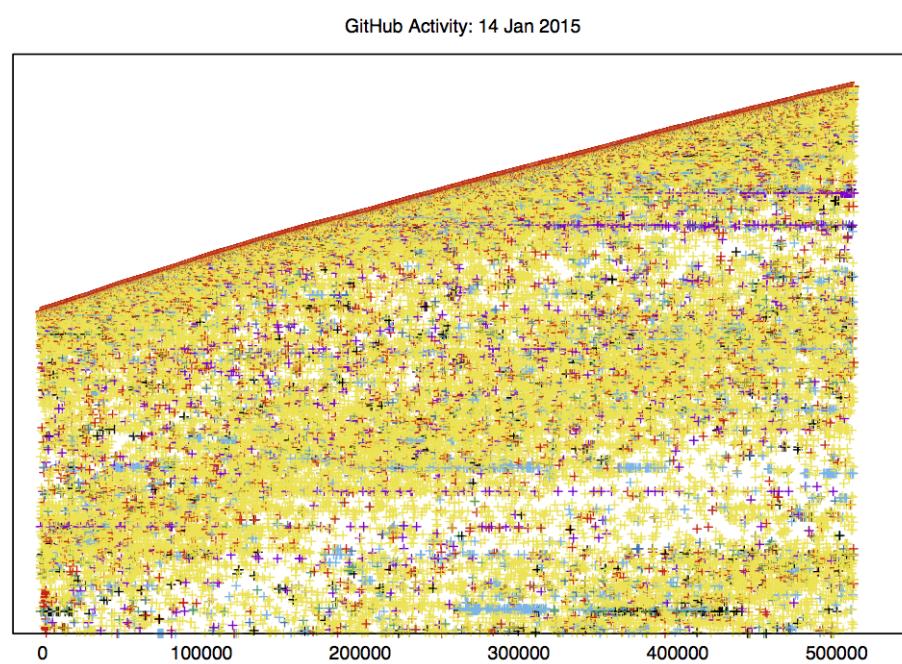


Figura 4: Un dettaglio della mappa relativa al 14/1/15 (figura 3)

chiaramente come la distribuzione dei Create è più densa in corrispondenza di repository più nuove, ma sia comunque presente nella parte bassa del grafico: ciò è dovuto al fatto che non solo la creazione di un nuovo progetto, ma anche di un sotto-progetto, una branca o un tag vengono classificati dall'archivio come eventi di questo tipo.

La distribuzione degli eventi Fork risulta essere più densa agli estremi dell'asse delle repository, come se la creazione e lo sviluppo di nuove branche fosse preponderante in progetti più stabili, di cui si possono creare ulteriori releases e modifiche, e per progetti in via di sviluppo.

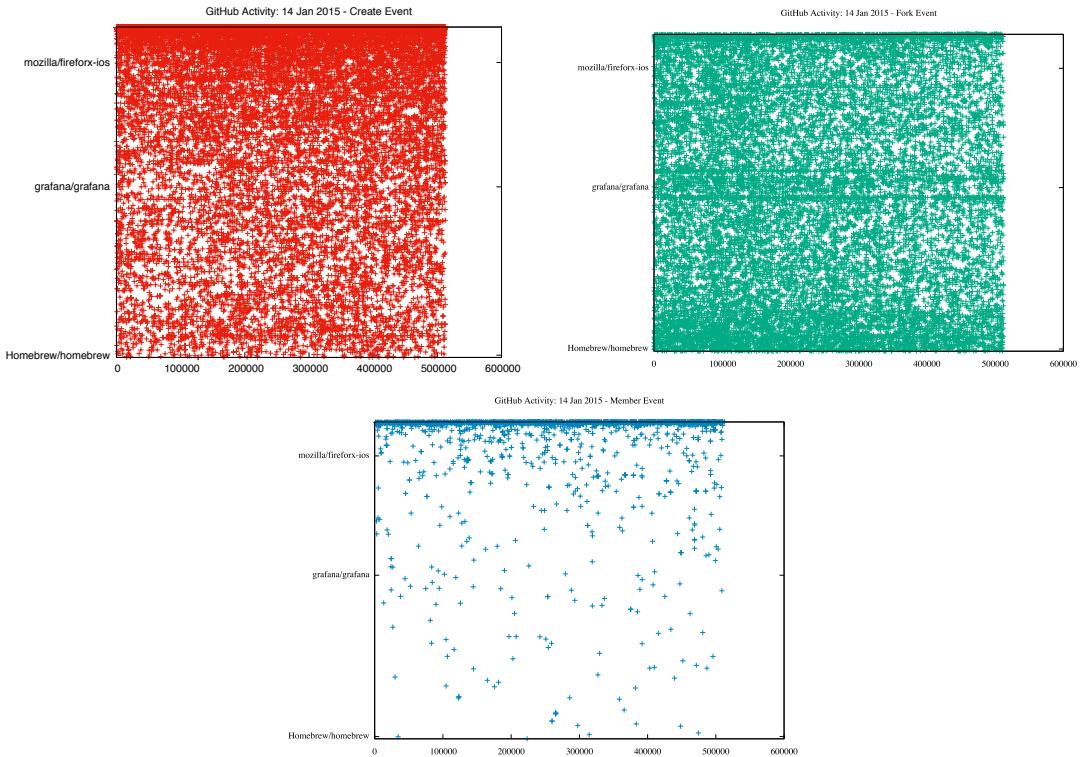


Figura 5: Mappe per eventi Create, Fork e Member

È interessante analizzare anche la distribuzione degli eventi di tipo Member, che segnalano l'ingresso di un nuovo contributore in un progetto.

Come ci si aspetta gli eventi di questo tipo coinvolgono maggiormente repository più recenti, che all'inizio del loro sviluppo vedono aumentare il numero di contributori.

Si conclude questa sezione con un dettaglio della parte alta del grafico (figura 4) in cui si mette in evidenza come anche gli eventi di tipo Create (rosso) interessino soprattutto le repository più recenti. In più si nota come effettivamente questa regione della mappa sia effettivamente dominata dagli eventi di tipo Push (giallo).

3.2 Attività annuale

Dallo studio dell'attività di tutto il 2015 dell'archivio, si ottengono interessanti informazioni su questo ambiente. Nel 2015 risultano essere attive oltre 13 milioni di repository. Dall'analisi dell'istogramma del numero di eventi l'anno per ogni repository si deduce che il 98% di questi progetti ha meno di 100 eventi l'anno.

Dalle analisi numeriche risulta che la distribuzione del numero di eventi l'anno per le repository segue la legge a potenza mostrata in figura 6, con esponente $b = -2.23$. Questo andamento suggerisce uno scenario in cui la

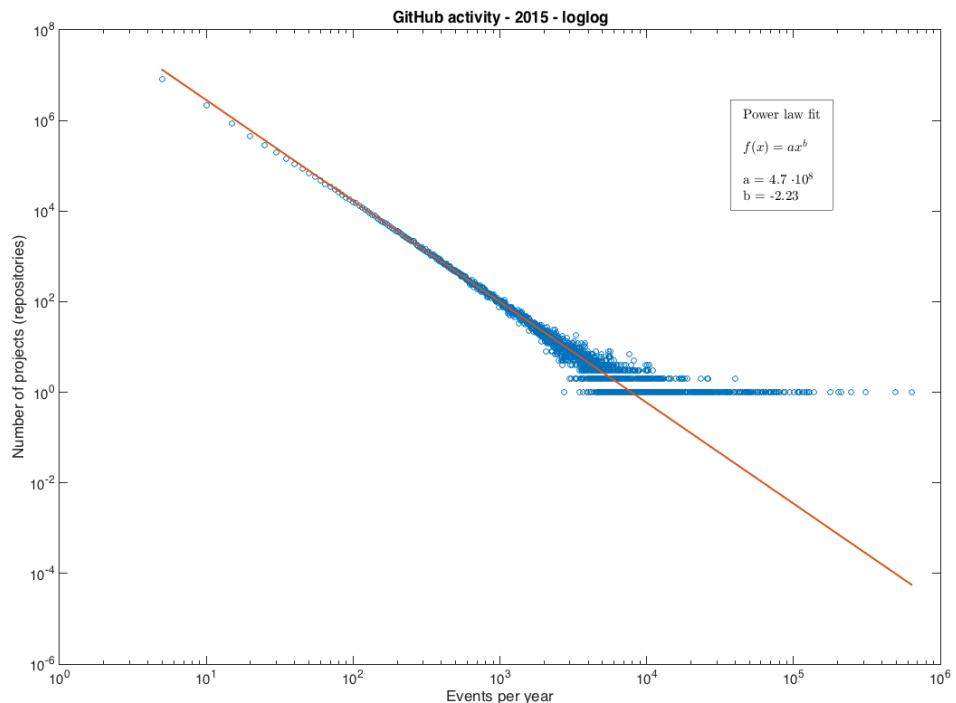


Figura 6: Distribuzione del numero di eventi relativi all'anno 2015

maggior parte dei progetti presenti nell'archivio sono molto poco attivi. Per quanto riguarda i punti molto lontani dalla distribuzione, si tratta di repository automatizzate, in cui non si ha una vera misura dell'attività umana rispetto ad un dato progetto.

3.3 Rete delle repository

Una delle possibili analisi dell'archivio di GitHub consiste nel considerare una rete in cui i nodi sono le repository e gli archi rappresentano, con il relativo peso, il numero di utenti in comune tra le due repository. L'analisi della rete è stata effettuata con il pacchetto NetworkX [9] di Python unitamente al software Gephi [8].

La rete globale per tutto un anno di attività consisterebbe di $13 \cdot 10^6$ nodi, ed un numero sicuramente molto maggiore di archi, e risulta pertanto proibitiva da calcolare. Considerando i risultati illustrati in precedenza si

possono analizzare diverse reti, selezionando le repository a seconda del numero di eventi annuali.

3.3.1 Rete delle repository con basso numero di eventi l’anno

Nel caso in cui gli eventi annuali siano minori o uguali a 1000, nella cui categoria rientra la maggior parte delle repository, la situazione tipica è rappresentata in figura 7.

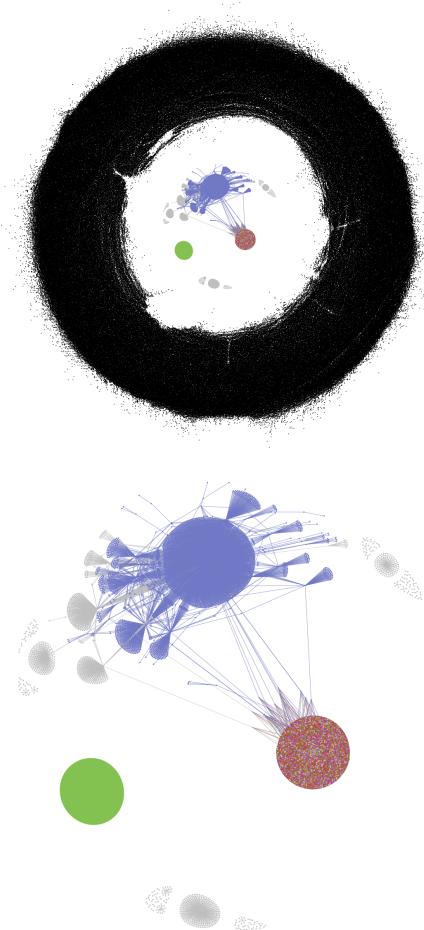


Figura 7: Rete per $5 \cdot 10^5$ repository con meno di 1000 eventi l’anno. In basso: dettaglio della parte centrale, costituita dalle sole repository con grado non nullo.

Sulla rete è stato condotto uno studio di modularità per evidenziare la presenza di eventuali comunità. Si nota immediatamente la galassia di nodi colorati di nero, cui corrisponde un grado nullo: si tratta di progetti che non condividono utenti con altri progetti e dunque sono scarsamente interessanti dal punto di vista della dinamica dell’innovazione.

Concentrandosi invece sulle repository con grado non nullo, l’andamento tipico in questo caso mostra la presenza di alcune repository rispetto a cui si addensano molti altri nodi non connessi tra di loro (nel dettaglio in basso), oltre alla presenza di una regione estremamente connessa. Questo tipo di topologia si può spiegare considerando delle repository che fungono da liberire o da repository di servizio, a cui sono agganciati i contributori delle altre

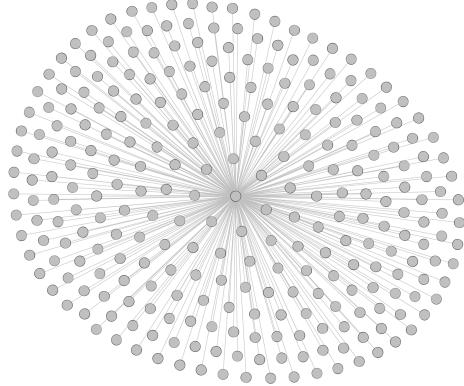


Figura 8: Dettaglio del cluster google-code, in cui si può apprezzare la struttura a "puntaspilli".

repository. Ciò spiegherebbe il comportamento a "puntaspilli" mostrato in figura 8, dove è riportato in dettaglio il cluster grigio in basso. Soffermandosi invece sulla parte più colorata della figura precedente, risulta essere un cluster in cui le repository sono collegate a sette repository centrali, le quali a loro volta sono collegate al resto della rete tramite altri nodi, come si può notare facilmente confrontando le immagini dello stesso cluster escludendo i nodi con grado pari a 7 (figura 9)¹. Si tratta dunque di un cluster con una struttura sostanzialmente simile a quello degli altri gruppi di nodi della rete, con l'unica differenza che esistono sette nodi centrali e non uno solo. Relativamente a

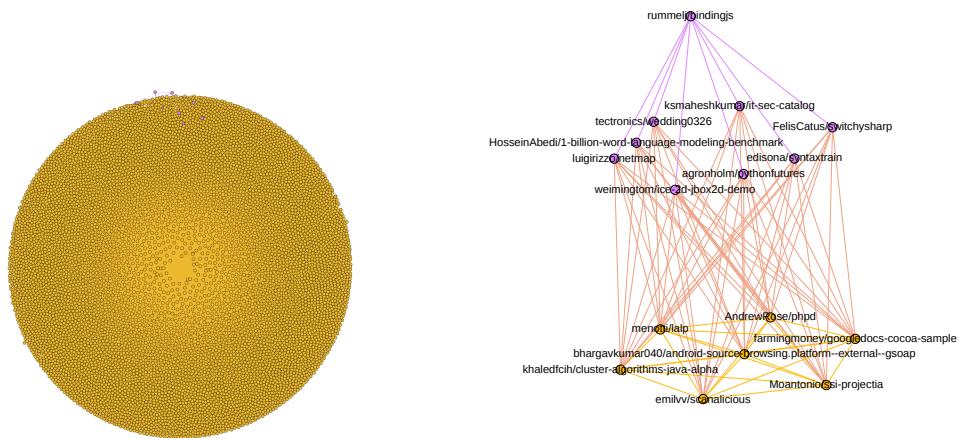


Figura 9: Secondo esempio di cluster

questo cluster dunque c'è un numero molto grande di repository collegate con tutti e sette questi nodi centrali, i quali a loro volta sono collegati ad altre parti della rete (i nodi fucsia). Studiando caso per caso queste repository centrali risultano tutte quante essere repository esportate automaticamente da progetti di codice di Google.

¹Lo studio di modularità effettuato in precedenza mostrava un cluster molto colorato, dunque con molte comunità al suo interno. Con un'analisi più puntuale e osservando il comportamento del cluster alle operazioni di *filtering* dei nodi con un certo grado, le comunità risultava essere unica.

Nel caso preso in esame in cui ci si limita a considerare solo una rete con $5 \cdot 10^5$ nodi, il diametro della rete risulta essere pari a 9, mentre il coefficiente di clustering pari a 0.921.

3.3.2 Rete delle repository con alto numero di eventi l'anno

Considerando invece repository con un numero di eventi superiore a 4000 l'anno, i nodi sono 2145, per un totale di 1.3 milioni di link, ed è possibile calcolare la rete completa, riportata in figura 11. Anche in questo caso sono presenti progetti isolati rispetto a tutti gli altri, mostrati al bordo della rappresentazione; tra questi sono presenti anche delle repository automatizzate.

Oltre ad una regione centrale estremamente connessa, si nota anche la presenza di alcuni cluster di repository, evidenziati cromaticamente in seguito ad una analisi di modularità, i cui elementi sono estremamente connessi gli uni con gli altri, come mostrato nel dettaglio in basso della figura. Questa rete ha un diametro pari a 5 ed un coefficiente di clustering di 0.901.

In figura 10 è riportata la distribuzione $P(k)$ del grado per questa rete. È possibile osservare due andamenti distinti, entrambi a legge di potenza: uno per gradi bassi, con esponente pari a 1.02, e uno per gradi alti, con esponente pari a 3.20. Il valore asintotico 3.20 è compatibile con il modello di Barabasi-Albert ([5]) per le reti complesse, che prevede un esponente pari a 3.

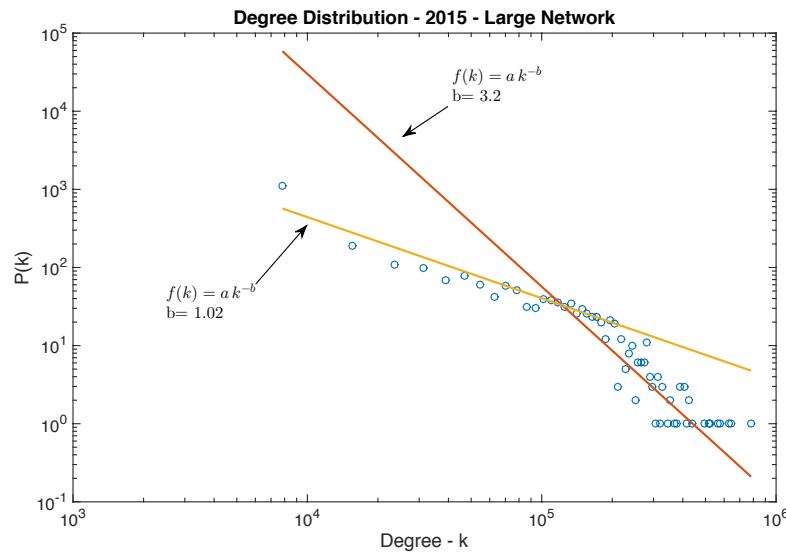


Figura 10: Distribuzione del grado per la rete comprendente le repository con più di 4000 eventi l'anno.

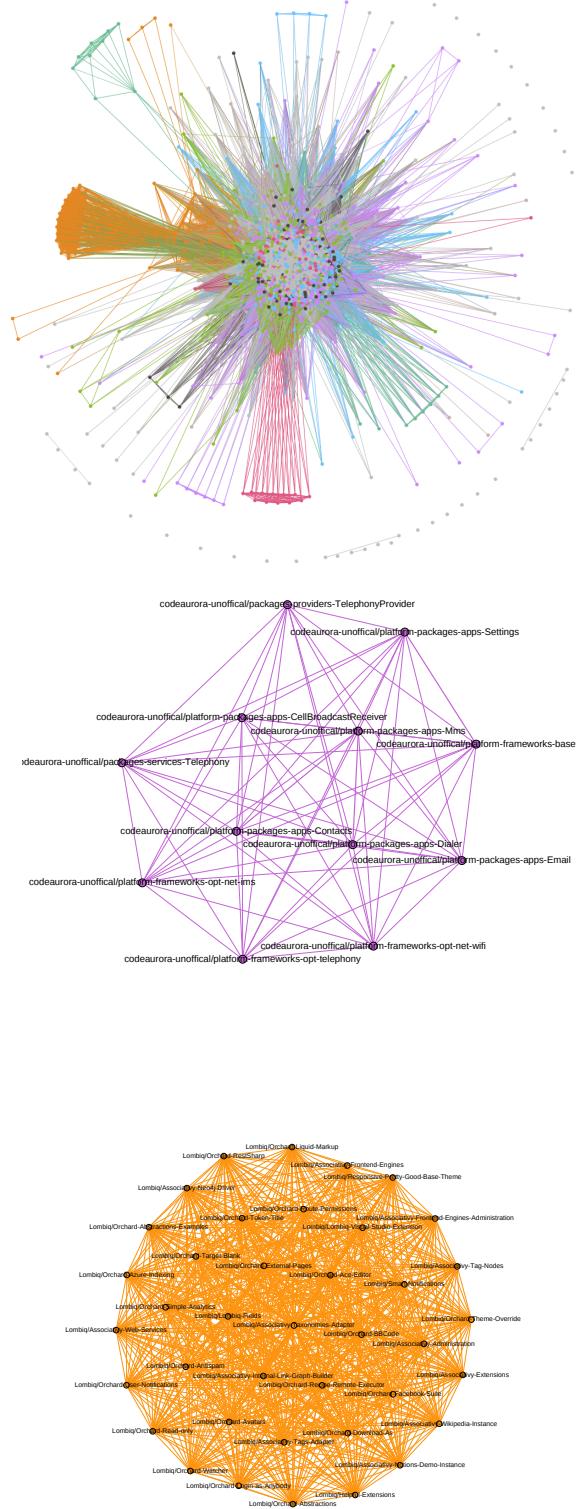


Figura 11: Rete completa per nodi con più di 5000 eventi con dettaglio di due cluster di repository appartenenti a uno stesso progetto (codeaurora in alto, Lombiq in basso).

4 Dinamica delle innovazioni

4.1 Analisi delle singole repository

L'analisi sull'attività delle singole repository è stata effettuata sui dati relativi al 2015 di alcune tra le repository con più di 10^4 eventi, la cui rete è mostrata in figura 12.

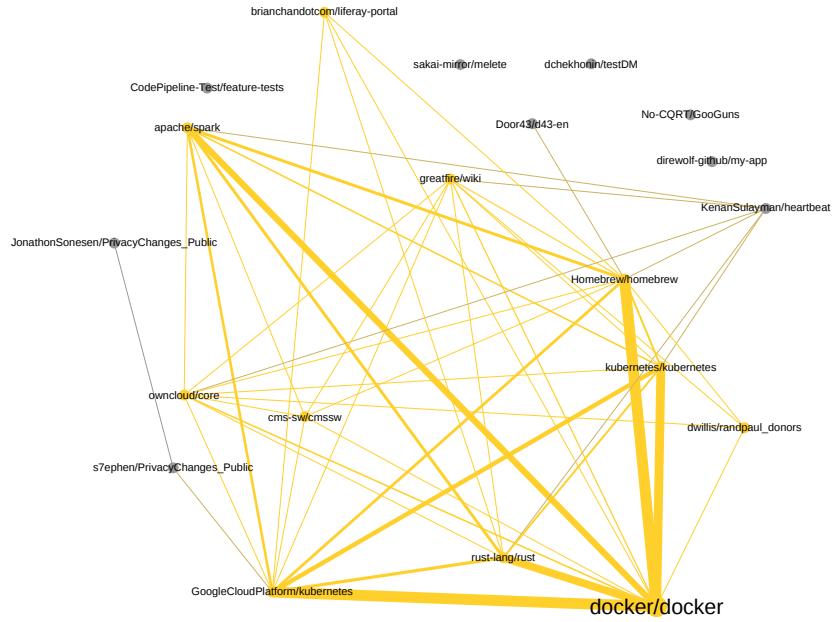


Figura 12: Rete delle repository con più di 10^4 eventi.

4.1.1 Definizione di *evento di innovazione*

Gran parte delle analisi che seguono sono state effettuate individuando due diverse definizioni di "evento di innovazione". Sono state considerate prima la coppia nome utente-tipo di evento e poi semplicemente il nome utente. Poiché ci sono solamente 25 tipi di evento, ci si attende che l'andamento sia comunque dominato dai nomi utente.

4.1.2 Distribuzione frequency-rank e numero di elementi distinti

Per le singole repository si può analizzare la distribuzione frequency-rank $f(R)$ e l'andamento del numero di elementi distinti $D(N)$ all'interno di una sequenza di lunghezza N .

Come si può osservare in figura 14, gli andamenti previsti dal modello (vedi Appendice oppure [4] per una trattazione più dettagliata) sono rispettati: il frequency-rank dell'occorrenza di eventi distinti nella sequenza, $f(R)$, ha un andamento a legge a potenza del tipo R^{-b} (legge di Zipf) per valori alti di R , mentre il numero di eventi distinti ha un andamento sublineare: $D(N) \propto N^a$ (legge di Heaps) con $a < 1$. Inoltre l'esponente b è asintoticamente compatibile con $1/a$.

Come già accennato, l'analisi è stata effettuata utilizzando due diverse definizioni di "evento", considerando prima la coppia nome utente-tipo di evento e poi semplicemente il nome utente. Non è stata osservata una differenza sostanziale tra gli andamenti per valori sufficientemente grandi dei parametri R e N , come si può osservare in figura 15.

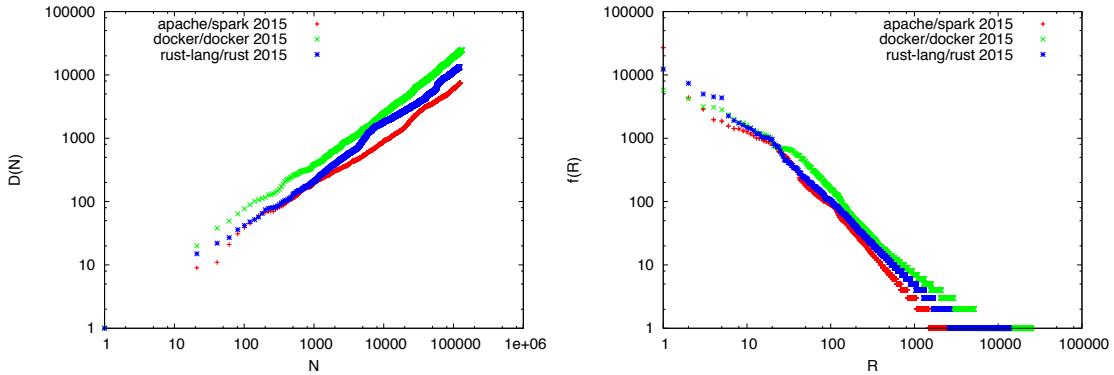


Figura 13: **Distribuzione frequency-rank e numero di elementi distinti per alcune delle repository più attive del 2015.** In particolare nel 2015 per docker/docker sono registrati circa $13 \cdot 10^4$ eventi, $20 \cdot 10^4$ per apache/spark e $12 \cdot 10^4$ per rust-lang/rust.

4.1.3 Intervalli temporali

Il primo modo per verificare la presenza di una correlazione tra gli eventi è quello di studiare la distribuzione degli intervalli temporali tra occorrenze successive di eventi dello stesso tipo. Partendo da un elenco temporalmente ordinato di eventi, è possibile visualizzare la distribuzione degli intervalli temporali e confrontarla con il caso in cui gli eventi sono stati rimescolati globalmente.

I dati sono ordinati temporalmente e il tempo è misurato secondo il tempo unix (*unix epoch time*)² e pertanto gli intervalli di tempo sono in secondi.

L'andamento ottenuto, mostrato in figura 16 mostra un picco più elevato rispetto a quello del caso random per brevi intervalli di tempo, implicando quindi che gli eventi dello stesso tipo si presentano raggruppati nel tempo e che gli eventi presi in considerazione non hanno una dinamica casuale ma vi sono delle correlazioni tra eventi successivi. L'idea è quindi che, in gran parte, gli eventi di uno stesso tipo avvengono a breve distanza temporale, come si poteva prevedere.

4.1.4 Entropia

L'entropia, come la distribuzione degli intervalli temporali, è una grandezza che ci dà informazione sul clustering degli eventi. Per calcolare l'entropia è necessario innanzi tutto individuare un' "etichetta" da poter assegnare agli eventi in esame. Una volta individuata tale etichetta, che indicheremo con E , possiamo calcolare l'entropia $S_E(k)$ di una sequenza \mathcal{S} di eventi come funzione del numero k di occorrenze degli eventi di tipo E. Per farlo, identifichiamo

²Il tempo unix è misurato in secondi rispetto alla mezzanotte (UTC) del 1/01/1970 (detta epoca)

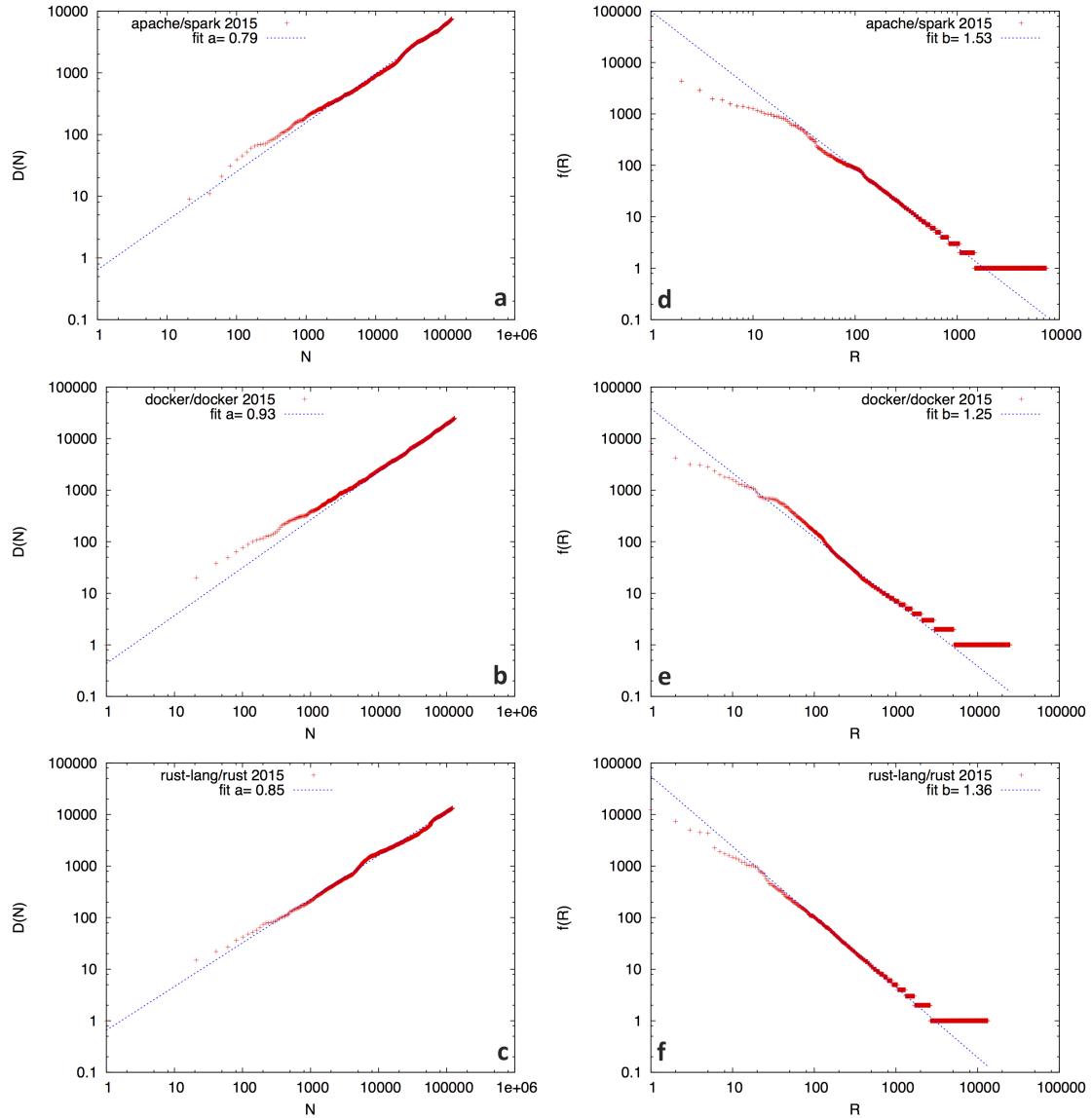


Figura 14: **Numero di elementi distinti (a-c) e distribuzione frequency-rank (d-f) per alcune delle repository più attive del 2015 con fit.** Come funzioni per i fit sono state utilizzate leggi a potenza (lineari in scala log-log) e i fit sono stati eseguiti nelle code delle distribuzioni. In particolare: per $D(N)$ è stata usata la forma funzionale cN^a ; per $f(R)$, cR^{-b} . Riportiamo i valori degli esponenti delle leggi a potenza ottenuti dai fit come verifica della relazioni tra le due leggi.
apache/spark (a), (d): $a = 0.79$, $b = 1.53$. **docker/docker** (b), (e): $a = 0.93$, $b = 1.25$.
rust-lang/rust (c), (f): $a = 0.85$, $b = 1.36$.

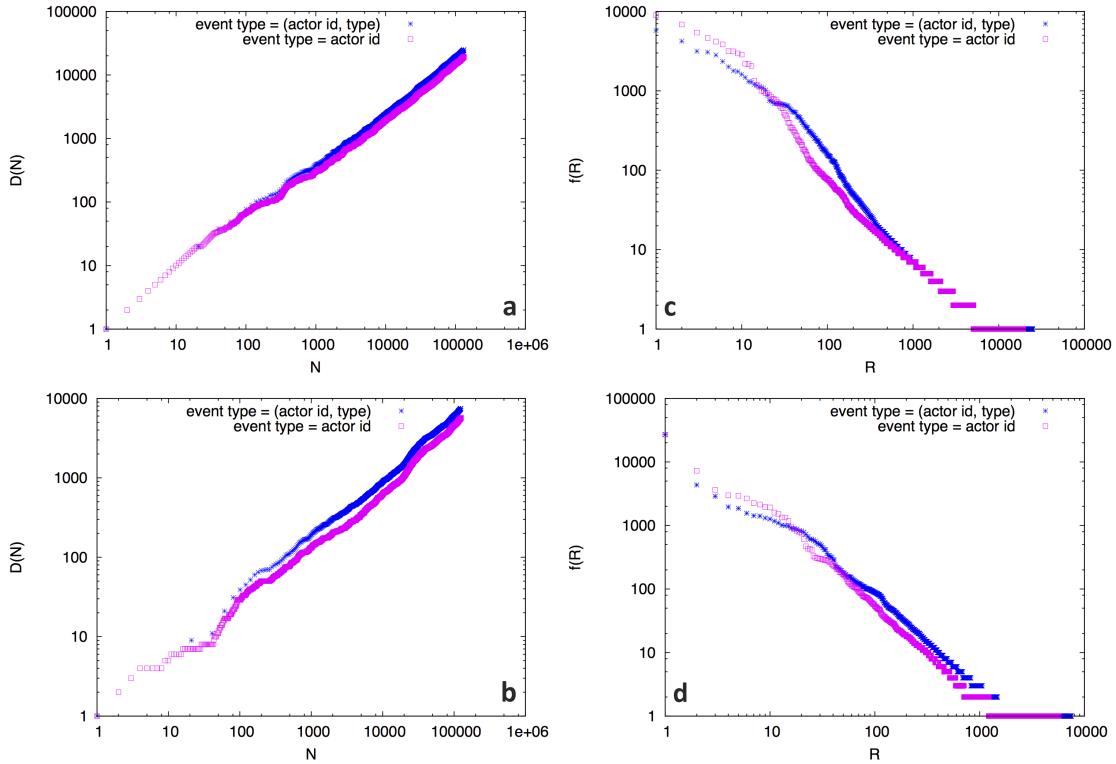


Figura 15: Numero di elementi distinti (a-b) e distribuzione frequency-rank (c-d) per le due diverse definizioni di evento. docker/docker: (a), (c); apache/spark: (b), (d).

innanzi tutto la sotto-seguenza S_E che ha inizio con la prima occorrenza di E. Poi, dividiamo tale sotto-seguenza in k intervalli uguali e in ognuno di essi calcoliamo il numero di occorrenze f_i di E. Definiamo quindi l'entropia di E come

$$S_E(k) = - \sum_{i=1}^k \frac{f_i}{k} \log \frac{f_i}{k} \quad (1)$$

Ogni $S_E(k)$ viene quindi normalizzata dividendola per $S_E^{\max}(k) = \log k$, e l'entropia totale $S(k)$ viene calcolata mediando sulle entropie di tutti gli elementi che occorrono k volte nella sequenza.

Per verificare che il programma per il calcolo dell'entropia funzionasse nel modo giusto, è stato prima testato su alcuni testi letterari scaricati da Project Gutenberg.

Una volta verificato il corretto funzionamento del programma, sono stati analizzati i dati relativi all'attività di alcune repository relativi al 2015, ordinati temporalmente. Sono state di volta in volta adottate diverse definizioni per l'etichetta: all'inizio è stata scelta la coppia nome utente-tipo di evento, poi l'etichetta è stata identificata con il nome utente e infine è stata effettuata un'ultima analisi selezionando eventi di un solo tipo ed identificando nuovamente l'etichetta con il nome utente.

I grafici ottenuti sono poi stati confrontati con quelli relativi a sequenze rimescolate. Sono stati effettuati due tipo di rimescolamento: un rimescolamento *locale*, in cui solo la sotto-seguenza viene rimescolata; e un rimescolamento *globale*, in cui l'intera sequenza viene rimescolata.

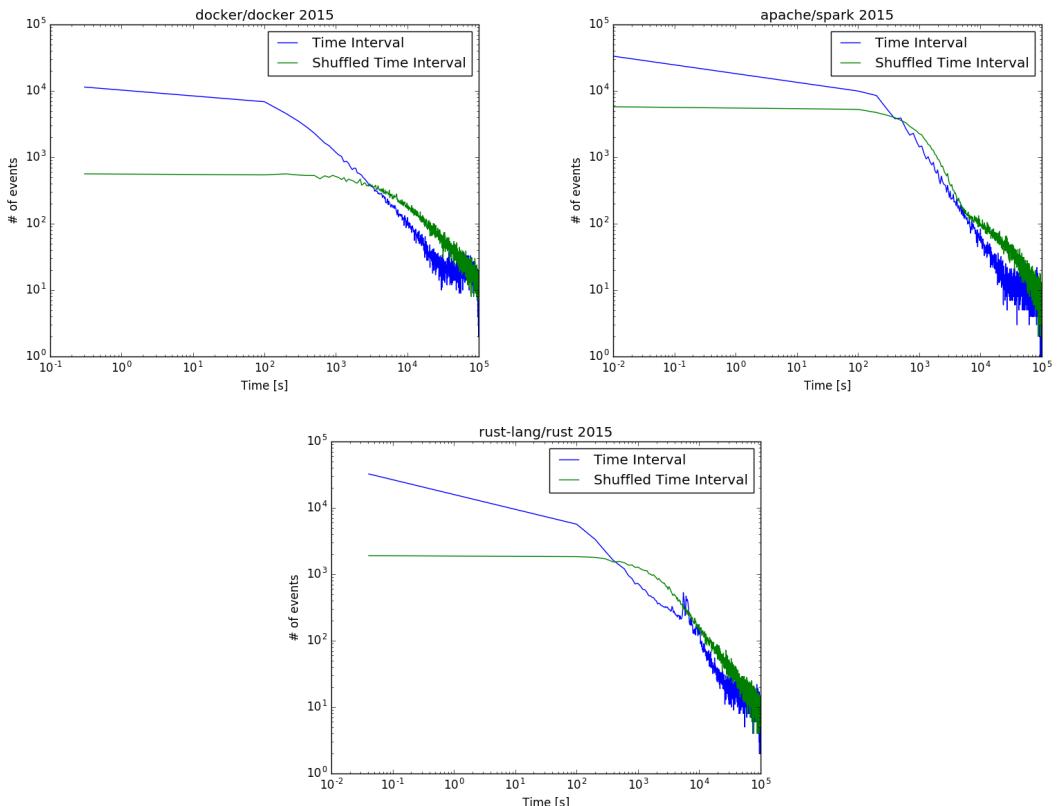


Figura 16: **Distribuzione degli intervalli temporali.** I dati mostrano un picco per brevi intervalli di tempo, maggiore di quello nel caso random: questo suggerisce una correlazione tra gli eventi e una conseguente clusterizzazione di essi nel tempo. Parametri: larghezza dei bin: 1000 s. L’istogramma è mostrato solo per intervalli temporali minori di 10^5 s.

Come ci si attende, rimescolando la sequenza si ha un significativo aumento dell'entropia, soprattutto per bassi valori di k . Possiamo dunque concludere che l'attività degli utenti non avviene in modo random, ma soprattutto che vi sono correlazioni tra gli eventi, come già osservato per gli intervalli temporali.

Si osserva che l'andamento dell'entropia calcolata per un solo tipo di evento è qualitativamente simile a quello dell'entropia calcolata prendendo in considerazione tutti i tipi di evento. Non si nota inoltre in nessuno dei casi una sostanziale differenza tra rimescolamento locale e globale.

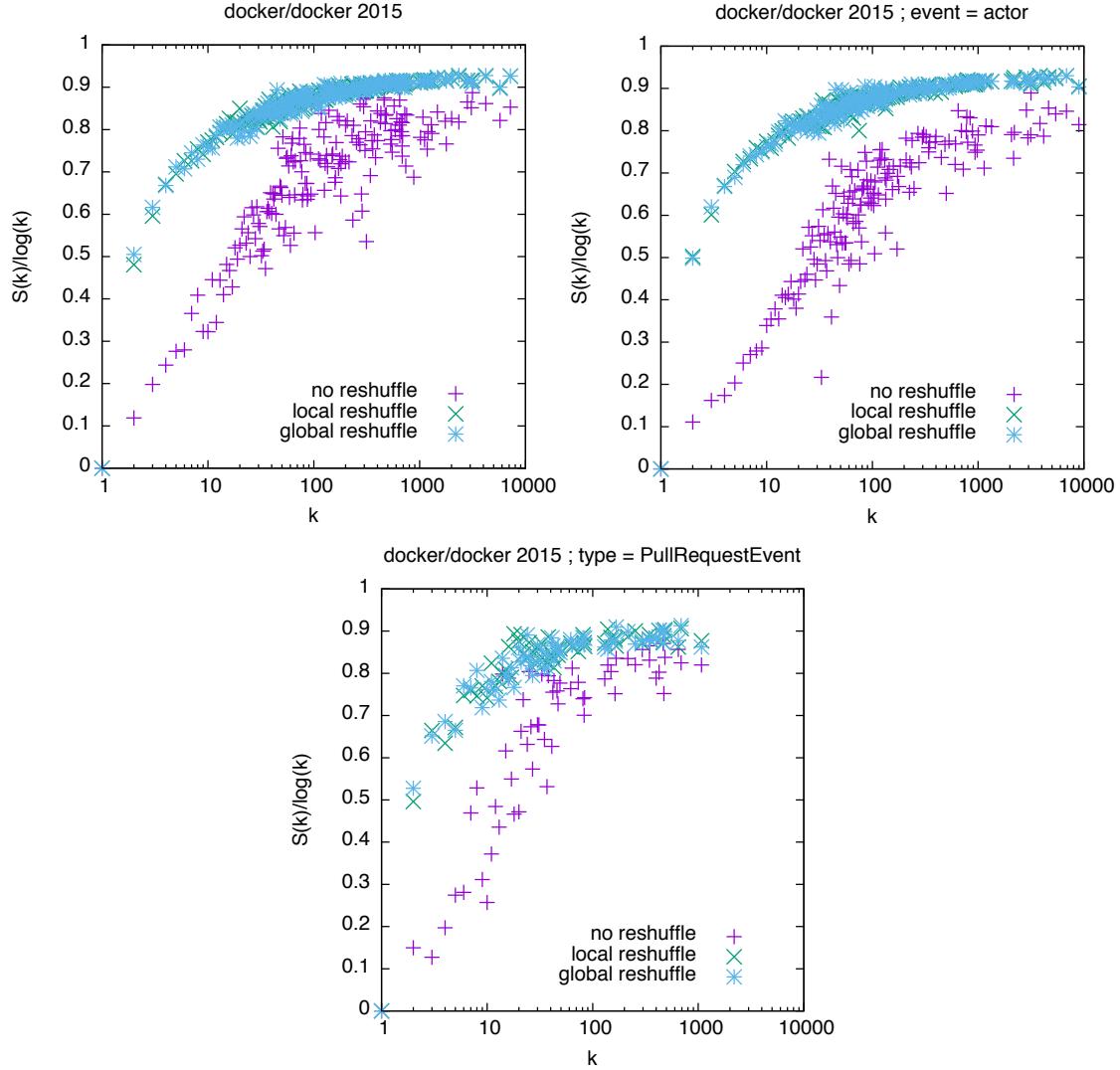


Figura 17: **Entropia della repository docker/docker** (a): Entropia relativa alla repository docker/docker, anno 2015. Etichetta = (attore, tipo di evento). (b): Entropia relativa alla repository docker/docker, anno 2015. Etichetta = attore. (c): Entropia relativa alla repository docker/docker, anno 2015. In questo caso abbiamo eliminato dalla sequenza tutti gli eventi che non fossero di tipo Pull Request. Etichetta = attore.

4.2 Analisi estesa a più repository

Tutte le analisi appena descritte sono state replicate prendendo in considerazione tutti i primi 10^8 eventi avvenuti sull'intero GitHub nel 2015. Questo tipo

di analisi non evidenzia più le correlazioni interne ad una repository, mettendo insieme tutto quello che succede su GitHub, e permette di valutare quanto già descritto da un punto di vista complessivo.

I risultati ottenuti sono solamente parziali, in quanto per esplorare bene le code delle distribuzioni servirebbero quantità di dati ancora maggiori, analizzare i quali al momento è risultato computazionalmente proibitivo.

Utilizzando come definizione di evento nome utente-tipo di evento per tutti i dati scaricati, sono stati analizzati la distribuzione frequency-rank (figura 18) e l'andamento del numero di elementi distinti con la lunghezza della sequenza (figura 19).

La distribuzione frequency-rank segue un andamento compatibile in prima analisi con una legge a potenza con esponente $b = 0.72$. Per quanto riguarda il numero di elementi distinti $D(N)$, non siamo riusciti a determinare con sufficiente sicurezza se seguia asintoticamente un andamento a potenza o lineare. Notiamo che un andamento lineare sarebbe consistente con quanto previsto dal modello (vedi Appendice oppure [4]), visto che l'esponente di $f(R)$ risulta essere in modulo minore di 1.

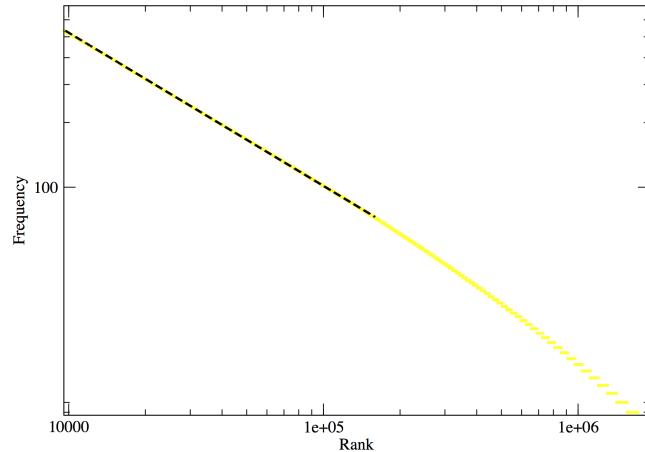


Figura 18: Frequency-rank per 10^8 eventi di tutto GitHub. L'andamento è compatibile con una legge a potenza con esponente $b = 0.72$

Per quanto riguarda la misura degli intervalli temporali tra due eventi dello stesso utente si trova di nuovo un picco di eventi a tempi brevi (figura 20), ad indicare una correlazione tra eventi successivi maggiore che nel caso di confronto in cui i tempi degli eventi vengono riassegnati casualmente con probabilità uniforme.

Infine, è stata calcolata l'entropia anche per un sottoinsieme di questi eventi ($5 \cdot 10^5$), scegliendo come etichetta la coppia nome utente-tipo di evento. L'andamento è qualitativamente compatibile con quello riscontrato nello studio delle singole repository e conferma quanto ottenuto per gli intervalli temporali, ovvero la presenza di correlazioni temporali dell'attività degli utenti.

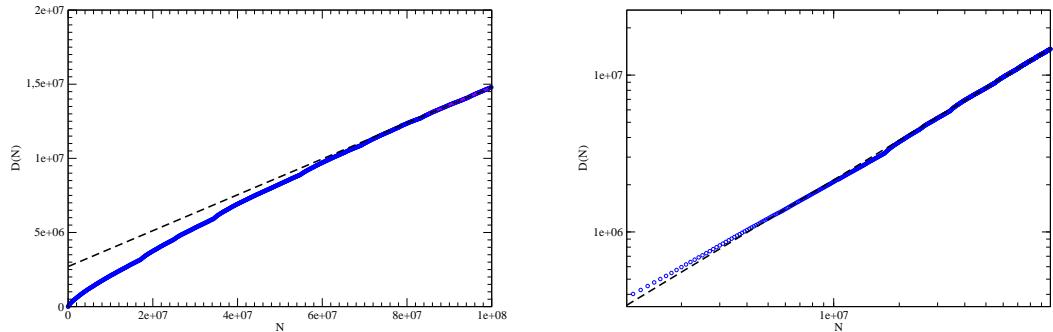


Figura 19: $D(N)$ per 10^8 eventi di tutto GitHub. Sinistra: fit lineare. Destra: fit con legge a potenza con esponente $a = 0.80$.

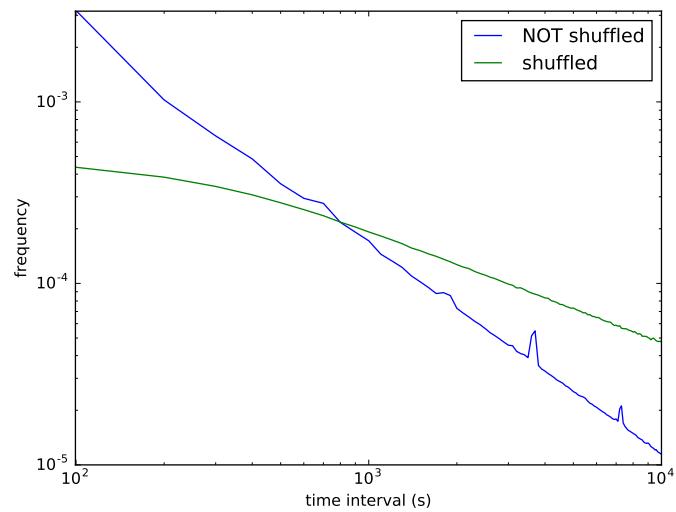


Figura 20: Distribuzione degli intervalli temporali per 10^8 eventi di tutto GitHub. Per grandi valori di N non siamo riusciti a determinare con certezza se l'andamento sia a legge di potenza o lineare.

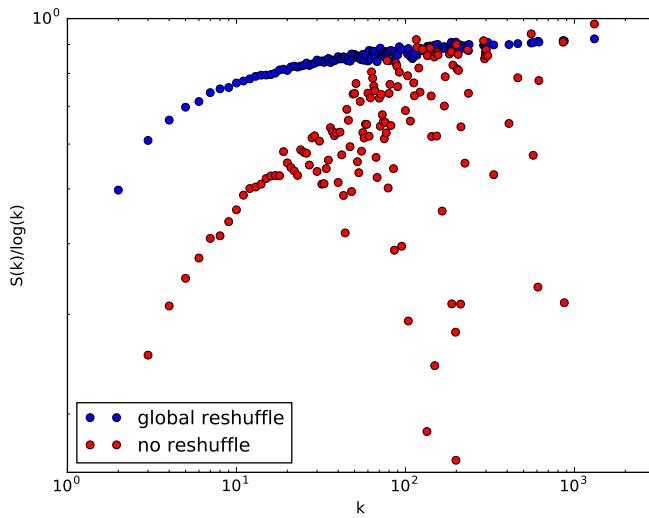


Figura 21: Entropia per $5 \cdot 10^5$ eventi di tutto GitHub. Etichetta: nome utente-tipo di evento. Il reshuffle locale non è riportato in quanto si sovrappone quasi totalmente con quello globale.

5 Conclusioni

In questo lavoro sono state studiate alcune caratteristiche della dinamica di GitHub tipiche dei sistemi complessi in presenza di innovazione.

A partire da una prima analisi qualitativa per caratterizzare i vari tipi di evento messi a disposizione da GitHub, si è passati prima ad un’analisi di rete, analizzando caratteristiche come la distribuzione del grado e il clustering coerenti con quanto si aspetta ad un sistema complesso, e infine alla caratterizzazione della dinamica di innovazione, attraverso il calcolo del frequency-rank, del numero di elementi diversi in una sequenza, dell’entropia e della distribuzione degli intervalli di tempo tra due eventi di innovazione. Queste misure sono in larga parte compatibili con le predizioni dei modelli di dinamica delle innovazioni ([4]).

Se pure tali analisi non si distaccano da quanto già analizzato in altri sistemi analoghi, conferma l’intuizione che lo sviluppo di OSS sia guidato da dinamiche di innovazione misurabili e apre la porta ad analisi successive.

Riferimenti bibliografici

- [1] M. W. Godfrey and Q. Tu, *Evolution in Open Source Software: A Case Study*. In: Proceedings International Conference on Software Maintenance (2000) pp 131-142. DOI: 10.1109/ICSM.2000.883030. URL: <https://plg.uwaterloo.ca/migod/papers/2000/icsm00.pdf>
- [2] A. Israeli, D. G. Feitelson, *The Linux kernel as a case study in software evolution*. In: The Journal of System and Software, 83 (2010) pp. 485-501. DOI: 10.1016/j.jss.2009.09.042
- [3] K. Nakakoji et al., *Evolution Patterns of Open-Source Software Systems and Communities*. International workshop on Principles of Software Evolution (2002)
- [4] F. Tria et al. *The dynamics of correlated novelties*. In: Scientific Reports (2014). DOI: 10.1038/srep05890
- [5] Réka Albert and Albert-László Barabási, *Statistical mechanics of complex networks*. In: Rev. Mod. Phys. 74, 47 – Published 30 January 2002.
- [6] Github: <https://github.com>
- [7] Google Query <https://bigquery.cloud.google.com>
- [8] Gephi: <https://gephi.org>
- [9] NetworkX <https://networkx.github.io>

Appendice A

Modelli matematici di dinamica delle innovazioni

Uno dei modelli matematici più semplici di dinamica delle innovazioni è quello basato sull'urna di Polya ([4]).

In questo modello, un'urna contiene palline di diversi colori; ad ogni passo temporale, una pallina a caso viene estratta e ispezionata e reinserita nell'urna insieme a ρ copie. In questo modo si rende più probabile l'estrazione di eventi che sono comparsi più volte in passato (un'idea che ha funzionato spesso nel passato verrà probabilmente riutilizzata più volte). Se è la prima volta che una certa pallina viene estratta, allora inseriamo nell'urna anche $\nu + 1$ elementi distinti completamente nuovi. Questo procedimento serve a modellizzare l'ampliamento dello spazio delle possibilità che una nuova scoperta o una nuova invenzione comporta³.

Il numero di elementi distinti in funzione della lunghezza N della sequenza, $D(N)$, ha i seguenti andamenti a seconda del valore dei parametri ν e ρ :

$$D(N) \propto \begin{cases} N^{\nu/\rho} & \nu < \rho \\ \frac{N}{\log N} & \nu = \rho \\ N & \nu > \rho \end{cases}$$

Corrispondentemente, la distribuzione frequency-rank ha un andamento del tipo $f(R) \propto R^{-\rho/\nu}$ (legge di Zipf). Questo modello spiega dunque contemporaneamente sia la legge di Heaps che quella di Zipf, senza che la validità di una delle due debba essere postulata per ricavare l'altra.

È possibile modificare il modello inserendovi la nozione di *semantica*, creando cioè dei "gruppi semanticci" di palline tramite l'assegnazione di etichette. In questo modello modificato, la probabilità di estrazione dipende dall'etichetta dell'ultima pallina estratta. L'etichetta viene conservata nel processo di rinforzo in cui vengono inserite ρ copie dell'ultima estratta; inoltre le $\nu + 1$ nuove palline che vengono inserite quando un nuovo elemento compare nella sequenza viene assegnata una nuova etichetta.

Questo modello riproduce anche il comportamento dell'entropia e della distribuzione degli intervalli temporali che si incontrano nell'analisi dei dati sperimentali. Per maggiori dettagli, si veda [4].

³Corrisponde al concetto di Kauffman del possibile adiacente per cui ciascuna novità, o più in generale qualsiasi innovazione, non si realizza da sola ma porta con se uno spazio di nuove possibilità.