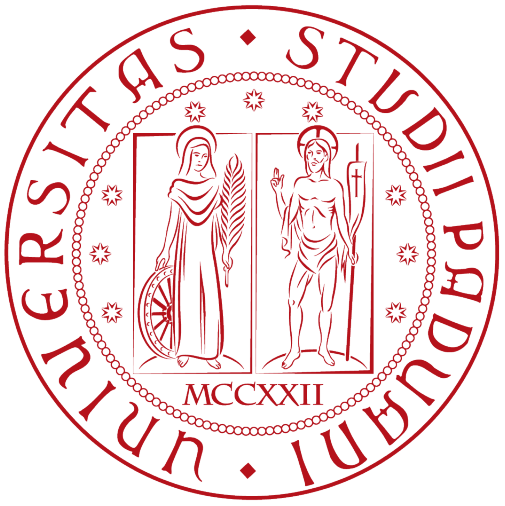


Università degli Studi di Padova

Ingegneria del Software 1 - 20/21

Tribuo

Analisi di un progetto open source

Professori:
Mauro Migliardi
Alberto Sillitti

Gruppo:
Alberto Castagnaro
Antonio Franzoso
Christian Marchiori
Francesco Visentin

Indice

1 Introduzione

Il progetto che abbiamo scelto di analizzare è Tribuo.

Tribuo è una libreria open source scritta in Java da un team di ricercatori facenti parte dell'Oracle Labs Machine Learning Research Group. É stato utilizzato internamente ai laboratori Oracle per numerosi anni, per poi essere rilasciato in versione open source da Oracle nell'agosto 2020 su GitHub, sotto una licenza Apache 2.0.

Tribuo nasce con l'obbiettivo di fornire una libreria di Machine Learning in ambiente Java che sia in linea con le funzionalità e i bisogni richiesti per lo sviluppo di grandi sistemi software.

La prima versione 1.0.0 di Tribuo viene sviluppata nel 2016 e rilasciata nell'autunno dello stesso anno. Le versioni open source, rilasciate a partire da agosto 2020, sono:

- v4.0.0 (13 agosto 2020 First OS release)
- v4.0.1 (1 settembre 2020 Patch release)
- v4.0.2 (5 novembre 2020 Patch release)
- v4.1.0 (26 maggio 2021 Minor release)

2 Analisi produzione

L'analisi del processo di produzione del software si è basata sulle informazioni ricavate dalla repository oracle/tribuo di GitHub, dal sito, dalla documentazione ufficiale di Tribuo e da alcuni articoli relativi al progetto.

Abbiamo inoltre contattato Adam Pocock, lead developer del progetto, che ha fornito ulteriori informazioni riguardo l'organizzazione del team e l'evoluzione di Tribuo.

2.1 Team e organizzazione

Il team di sviluppo è formato da un gruppo di ricercatori nel campo del Machine Learning che hanno come focus primario research & data science.

Tribuo nasce inizialmente come progetto di due persone e cresce nel tempo coinvolgendo altri membri vista la sua utilità all'interno degli altri progetti seguiti dal gruppo di ricerca.

L'ultima release di Tribuo ha coinvolto 6 sviluppatori interni e un paio di sviluppatori esterni. Molti sviluppatori di Tribuo lavorano contemporaneamente ad altri progetti interni ad Oracle, perciò il lavoro all'interno del gruppo è suddiviso sia sulla base delle aree di maggior competenza di ogni membro, sia considerando gli impegni legati ai diversi progetti di ricerca che questi stanno attualmente seguendo.

I meeting giornalieri sono previsti solo nei momenti in cui il team sta lavorando al design ed implementazione di features che interessano una grossa parte del progetto. Non sono previsti meetings ricorrenti ma solo in caso di necessità. I vari membri della squadra si coordinano utilizzando Slack.

I contributors esterni possono collaborare al progetto aprendo issues su GitHub, segnalando bug, proponendo bug-fixes, features e scrivendo documentazione aggiuntiva.

2.2 Sviluppo software

Lo sviluppo di Tribuo è motivato dalla necessità di avere uno strumento a supporto dei lavori di data science all'interno di Oracle e il progredire del progetto non è quindi legato a regolari deadline. Per questo Tribuo non segue rigorosamente un processo standard di sviluppo software.

2.2.1 Requirements elicitation

Le priorità e gli obiettivi di Tribuo sono quindi definiti sulla base di:

- necessità interne dei ricercatori di Oracle
- ciò che la community richiede
- ciò che gli sviluppatori ritengono sia utile

- elementi presenti nel backlog del progetto (gestito tramite JIRA e GitHub)

Tribuo aderisce al concetto di *semantic versioning*. Si distinguono pertanto:

- **Major release**: sono le release più sostanziose, nelle quali le modifiche apportate riguardano solitamente i pacchetti che stanno alla base della libreria e possono quindi compromettere la retro compatibilità del codice con le versioni precedenti. La struttura di Tribuo è stata volutamente resa il più modulare possibile così da poter offrire la possibilità ai fruitori della libreria di usare unicamente i pacchetti di cui effettivamente hanno bisogno. Questa scelta di design ha portato però con sé ulteriori difficoltà nello sviluppo delle funzionalità riguardanti il core della libreria, rendendo questo tipo di release il meno frequente.
- **Minor release**: garantiscono la retro compatibilità del codice e si concentrano sull'implementazione o sul rework di una singola funzionalità principale (scelta effettuate per mantenere ridotta la complessità) accompagnata da una serie di piccoli bug fixes e features minori presenti nel backlog del progetto.
- **Patch release**: retro compatibili, risolvono eventuali bug e problemi di sicurezza quando questi vengono scoperti.

La feature principale e le features minori da implementare durante lo sviluppo della prossima major o minor release, vengono selezionate a seguito di meeting coordinativi iniziali e seguendo una priorità decisa dal team. In questi meeting viene steso un documento (solitamente in linguaggio informale) contenente le linee guida per la progettazione della prossima release.

L'implementazione delle features avviene su branch separati del progetto, uniti previa approvazione del lead developer, al branch principale quando lo sviluppo e la validazione di queste si è concluso.

2.2.2 Code development

Non seguendo in maniera precisa una metodologia di sviluppo software, né Agile né Plan-based, lo sviluppo non è scandito in fasi ben delineate, ma segue il documento (accessibile a tutto il gruppo) sulla base del quale viene effettuato lo sviluppo software. Esso viene gestito ed aggiornato tramite Slack, software di collaborazione aziendale utilizzato per scambiare messaggi e documenti in modo istantaneo all'interno del team, in modo da fornire una panoramica riguardante lo stato corrente del progetto.

Il compito di ogni membro del team, a differenza di un modello Plan-based, può includere mansioni diverse all'interno del progetto, potenzialmente anche variabili nel tempo. Queste vengono definite in relazione, oltre che alle competenze individuali, anche agli impegni legati ad altri progetti di ricerca.

L'assegnamento e la coordinazione generale di questi compiti viene supervisionata dal lead developer del progetto, il quale detenendone la responsabilità si assicura, tramite un

costante rapporto diretto tra i membri del team, che il lavoro venga svolto correttamente e efficientemente.

Nel caso uno dei membri si trovi in difficoltà riguardo una sezione particolarmente problematica del codice, egli ha modo di contattare immediatamente gli altri e, se necessario, viene organizzato un meeting al fine di trovare una soluzione adeguata. Durante la progettazione di alcuni degli elementi più complessi all'interno del progetto, gli sviluppatori hanno organizzato sessioni di **pair programming**, durante la quale due membri del team lavorano contemporaneamente all'implementazione di una stessa parte di codice scambiandosi regolarmente il ruolo di "*driver*" (colui che scrive il codice e espone una possibile implementazione funzionante) e "*navigator*" (colui che lo controlla e propone soluzioni alternative).

Nonostante si faccia largo uso di test all'interno del progetto, Tribuo non aderisce rigorosamente al modello del **test driven development**. Infatti, a seconda della natura della feature da implementare, in alcuni casi i test vengono scritti *prima* dell'effettiva implementazione di una specifica funzionalità (come previsto dal modello TDD), mentre in altri casi i test vengono ideati solamente *dopo* che gran parte del codice relativo alla feature è stato già scritto, al fine di validare il suo comportamento. In altri casi ancora può capitare di scrivere test per una libreria già esistente (e già testata in precedenza) al fine di estendere la *test coverage* di quella porzione di codice, sebbene ciò avvenga solitamente in preparazione ad un imminente *refactoring* dello stesso.

2.2.3 Software validation & delivery

Gli sviluppatori utilizzano coverage tools (es. JaCoCo, per misurare la test coverage) nativi degli IDEs usati (la maggior parte del team usa IntelliJ) ed eseguono analisi statiche per trovare eventuali bug prima della release del prodotto.

3 Analisi metriche

In questa sezione si analizza l'evoluzione del codice di Tribuo, prendendo in considerazione le metriche LOC, CK, CC.

I dati analizzati vanno a confrontare tra loro le quattro releases open source di Tribuo partendo dalla versione 4.0.0 fino alla 4.1.0.

Le metriche delle versioni 4.0.1 e 4.0.2, essendo patch release, evidenzieranno una variazione minima tra le due versioni mentre quelle delle minor release 4.0.0 e 4.0.1 saranno invece caratterizzate da differenze maggiori.

	Number of classes	Number of packages	Number of external packages	Number of external classes
Tribuo 4.0.0	782	101	23	90
Tribuo 4.0.1	787	101	23	90
Tribuo 4.0.2	788	101	23	90
Tribuo 4.1.0	873	109	31	116

Tabella 1: Numero di classi e pacchetti per ogni release di Tribuo.

A causa dell'elevato numero di dati ottenuti, il confronto tra le metriche delle varie versioni è stato effettuato principalmente per via grafica in modo da poterlo rendere più visibile e immediato.

3.1 Lines Of Code (LOC)

Le linee di codice sono state misurate utilizzando l'estensione *VS Code Counter* dell'editor Visual Studio Code. Vengono riportate le linee di codice di tutti i linguaggi presenti all'interno del progetto, differenziate tra **LOC** (*Lines Of Code*), **CLOC** (*Comment Lines Of Code*), **ELOC** (*Empty Lines Of Code*) e **TLOC** (*Total Lines Of Code*).

language	files	code	comment	blank	total
Java	683	52,956	25,543	12,162	90,661
XML	70	4,148	1,109	324	5,581
Markdown	14	1,359	0	302	1,661
Python	5	240	111	58	409
JSON	1	91	0	1	92
Shell Script	3	22	3	12	37
YAML	1	21	3	4	28

(a) v4.0.0

language	files	code	comment	blank	total
Java	685	53,522	25,691	12,252	91,465
XML	70	4,150	1,109	324	5,583
Markdown	14	1,360	0	302	1,662
Python	5	240	111	58	409
JSON	1	93	0	1	94
Shell Script	3	22	3	12	37
YAML	1	21	3	4	28

(b) v4.0.1

language	files	code	comment	blank	total
Java	687	54,135	27,191	12,360	93,686
XML	71	4,349	1,152	330	5,831
Markdown	16	1,498	1	337	1,836
JSON	4	558	0	6	564
Python	5	240	111	58	409
YAML	3	63	9	12	84
Shell Script	3	22	3	12	37

(c) v4.0.2

language	files	code	comment	blank	total
Java	752	61,188	32,016	13,836	107,040
JSON	6	31,445	0	8	31,453
XML	77	4,918	1,257	364	6,539
Markdown	18	1,613	1	364	1,978
Python	4	178	61	37	276
YAML	3	63	9	12	84
Shell Script	3	22	3	12	37
HTML	1	16	0	1	17

(d) v4.1.0

Figura 1: Andamento delle *Lines Of Code* lungo lo sviluppo di Tribuo.

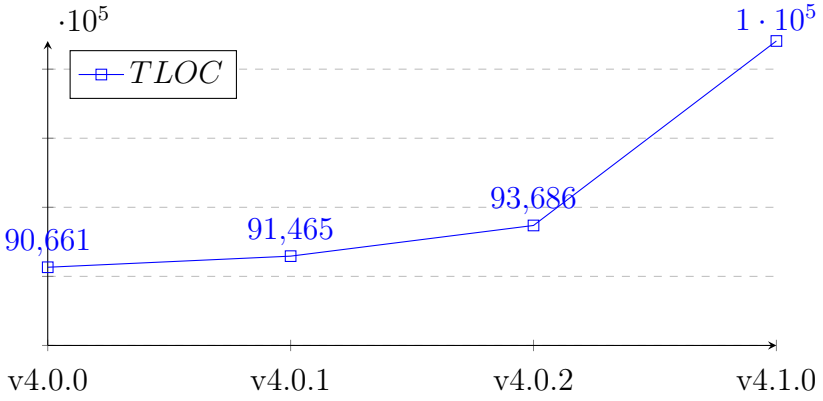


Figura 2: Andamento della metrica *TLOC* lungo lo sviluppo di Tribuo.

3.2 Chidamber and Kemerer's metrics suite (CK)

Una della suite di metriche più conosciute ed utilizzate in ambito Object Oriented è la **CK Suite** (*Chidamber and Kemerer's metrics suite*). Essa fornisce delle misure numeriche per ogni classe del progetto, facili da ottenere e utili per valutare la qualità del codice. Queste metriche, ottenute attraverso il plugin *CodeMR* dell'IDE IntelliJ IDEA, comprendono:

- *Weighted Methods per Class* (**WMC**)
- *Depth of Inheritance Tree* (**DIT**)
- *Number Of Children* (**NOC**)
- *Coupling Between Objects* (**CBO**)
- *Response For a Class* (**RFC**)
- *Lack of Cohesion in Methods* (**LCOM**)[1]

Essendo questa una misura numerica effettuata a livello di classe e non di progetto, si è deciso di riportare, per ogni release di Tribuo, i valori di tali metriche in due formati differenti.

Il primo (figura 3) rappresenta ogni metrica di CK sottoforma di grafico a torta, nel quale il colore e la grandezza di ogni fetta è proporzionale alla quantità di classi del progetto aventi quella determinata metrica a quel determinato livello (si veda la legenda di tabella 2).

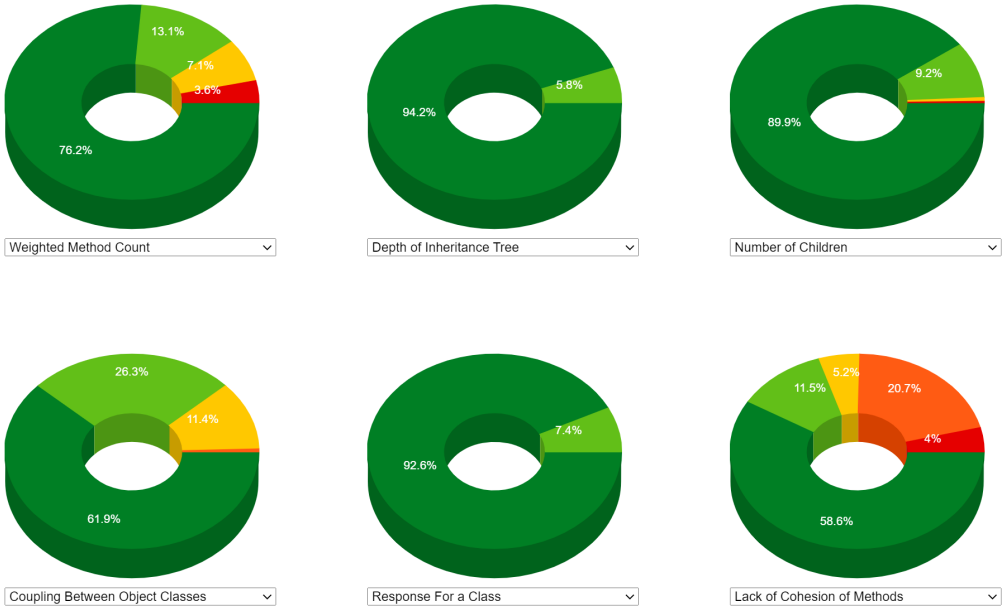
Il secondo (figura 4) confronta, attraverso un istogramma, le varie metriche in modo da poterne tracciare la progressione durante lo sviluppo delle differenti releases del progetto.

	Low	Low-Medium	Medium-High	High	Very High
WMC	0-20	20-50	50-101	101-120	>120
DIT	0-1	1-3	3-10	10-20	>20
NOC	0-1	1-5	5-10	10-15	>15
CBO	0-5	5-10	10-20	20-30	>30
RFC	0-50	50-100	100-150	150-200	>200
LCOM	0-0.5	0.5-0.7	0.7-0.8	0.8-1	>1

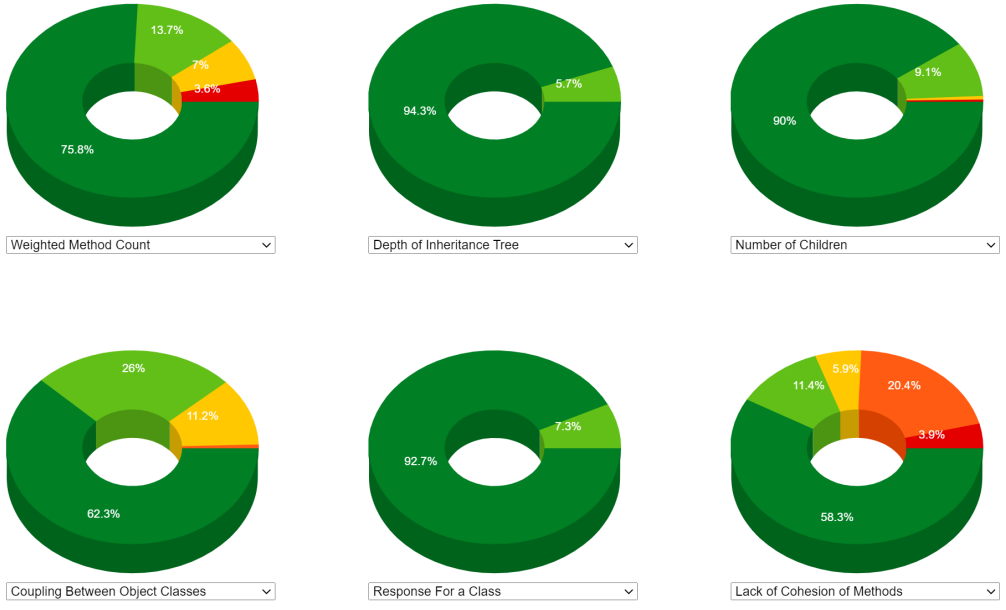
Tabella 2: Legenda di grafici a torta in figura 3.

[1] $LCOM = \frac{m - \frac{\sum mA}{a}}{m-1}$ dove:
- m, numero di metodi della classe
- a, numero di variabili della classe
- mA, numero di metodi che accedono a una variabile
- $\sum mA$, sommatoria degli mA per ogni variabile della classe

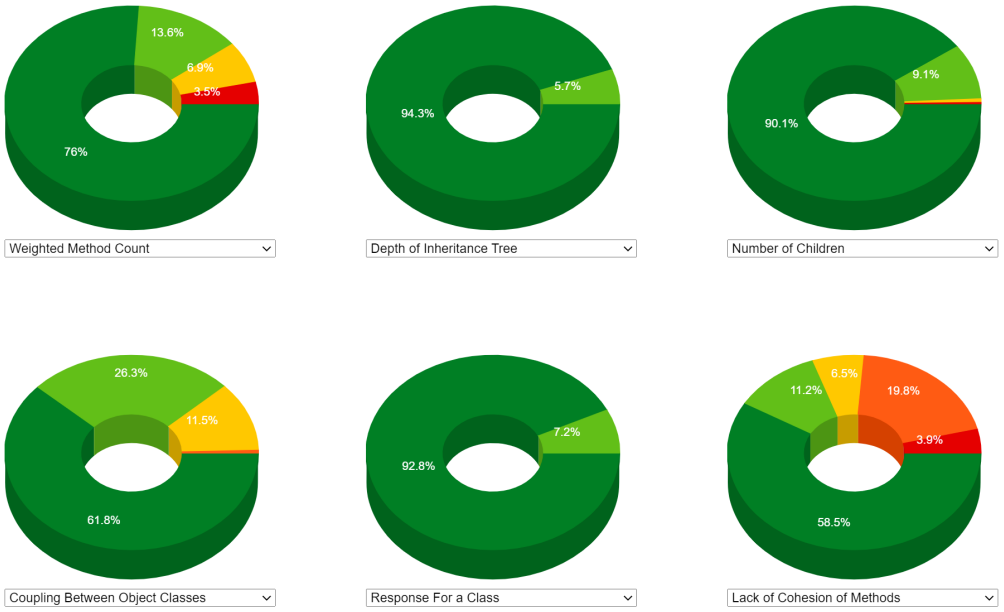
$LCOM$ è compreso tra 0 e 2, dove valori superiori a 1 sono considerati allarmanti poiché indicano una mancanza di coesione critica.


13.1%
7.1%
3.6%
76.2%
Weighted Method Count
5.8%
94.2%
Depth of Inheritance Tree
9.2%
89.9%
Number of Children
26.3%
11.4%
61.9%
Coupling Between Object Classes
7.4%
92.6%
Response For a Class
11.5%
5.2%
20.7%
4%
58.6%
Lack of Cohesion of Methods

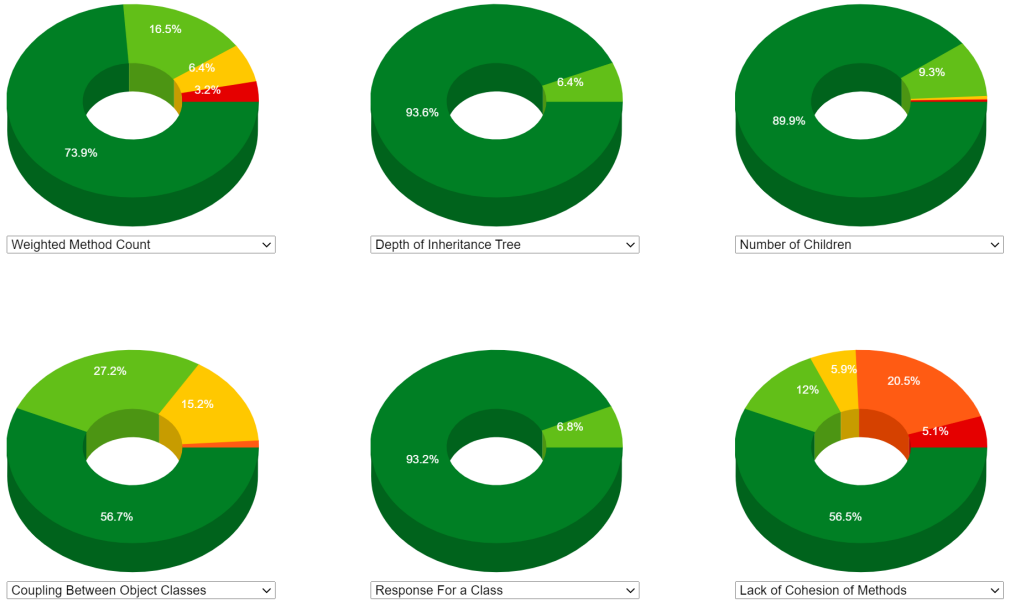
(a) v4.0.0


13.7%
7%
3.6%
75.8%
Weighted Method Count
5.7%
94.3%
Depth of Inheritance Tree
9.1%
90%
Number of Children
26%
11.2%
62.3%
Coupling Between Object Classes
7.3%
92.7%
Response For a Class
11.4%
5.9%
20.4%
3.9%
58.3%
Lack of Cohesion of Methods

(b) v4.0.1

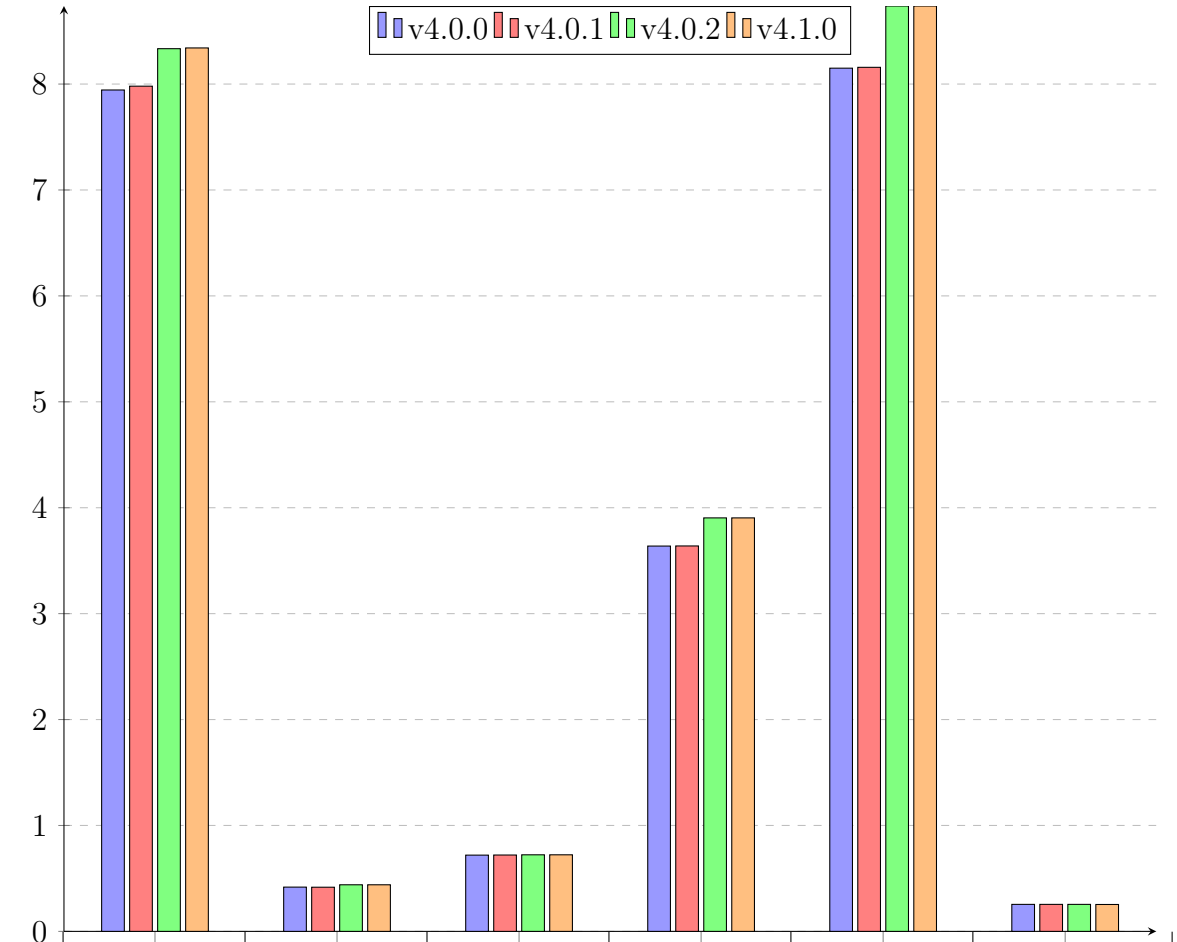


(c) v4.0.2



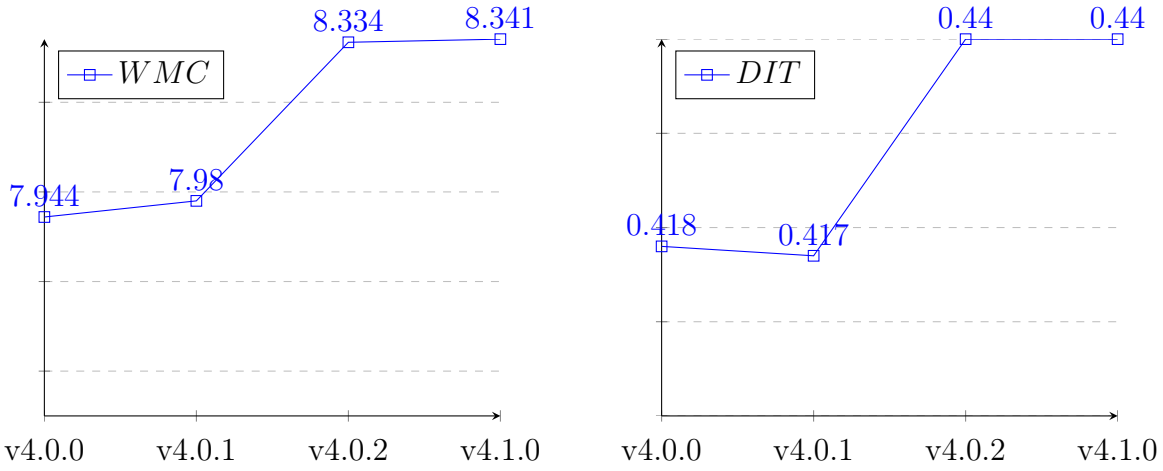
(d) v4.1.0

Figura 3: Metriche *CK* relative alle varie release di Tribuo.



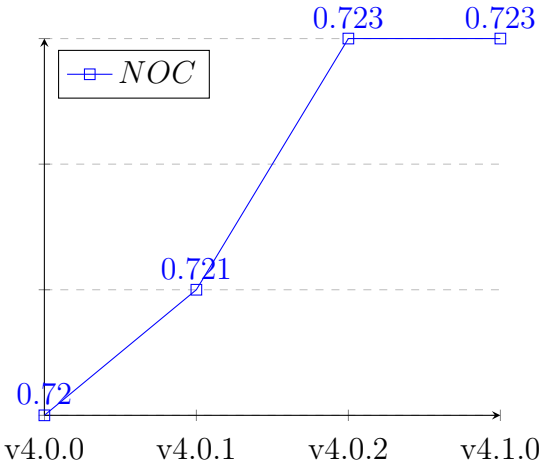
	WMC	DIT	NOC	CBO	RFC	LCOM
	0.952	0.932	0.972	0.952	0.932	0.972
	0.969	0.987	0.988	0.952	0.932	0.972
	0.969	0.997	0.997	0.952	0.932	0.972
	0.917	0.950	0.976	0.952	0.932	0.972

Figura 4: Andamento delle metriche *CK* lungo lo sviluppo di Tribuo.

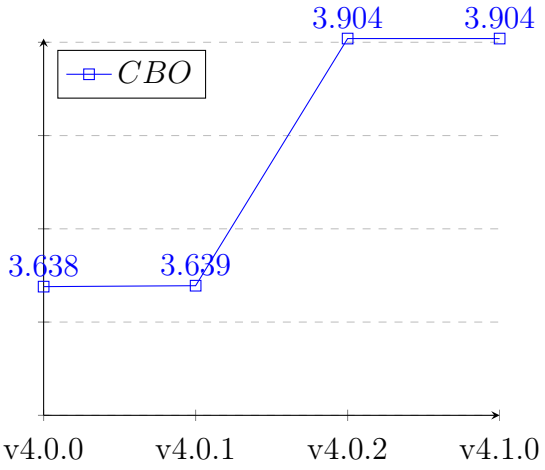


(a) WMC

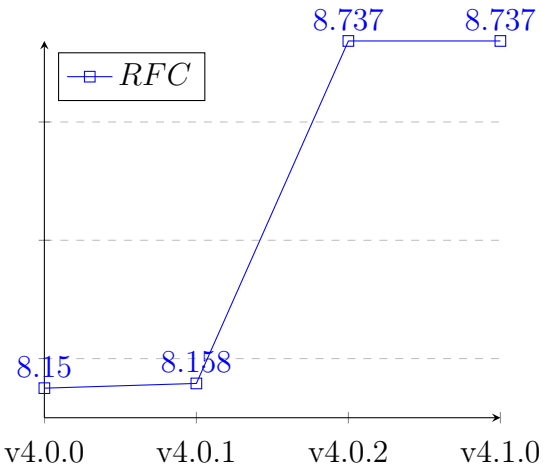
(b) DIT



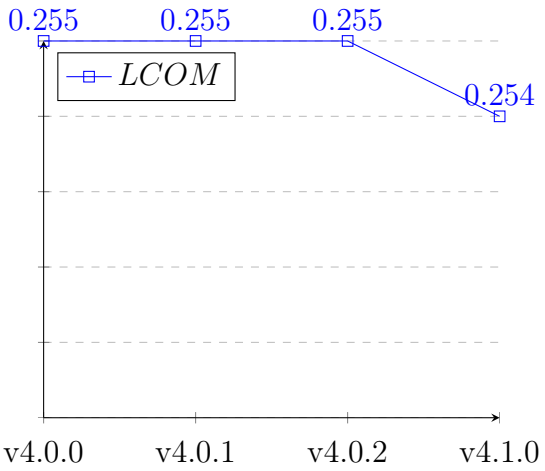
(c) NOC



(d) CBO



(e) RFC



(f) LCOM

Figura 5: Andamento delle metriche *CK* lungo lo sviluppo di Tribuo.

3.3 McCabe Cyclomatic Complexity (CC)

L'analisi della complessità ciclomatica viene effettuata attraverso il software JArchitect al fine di valutare l'andamento della complessità del codice di Tribuo all'interno delle varie releases.

La complessità ciclomatica è definita per i singoli metodi come:

$$CC_{method} = 1 + \{decision\ points\ found\ in\ the\ body\ of\ the\ method\} \tag{1}$$

È possibile estendere la complessità ciclomatica di una singola classe con la seguente formula:

$$CC_{class} = \sum_{\forall method\ \in\ class} CC_{method} \tag{2}$$

Perciò, al fine di ottenere una metrica di progetto, si considera la metrica ottenuta dalla somma della complessità ciclomatica delle singole classi facenti parti del progetto:

$$CC_{project} = \sum_{\forall class\ \in\ project} CC_{class} \tag{3}$$

Viene di seguito riportato un grafico raffigurante l'andamento delle CC risalenti alle quattro differenti releases di Tribuo.

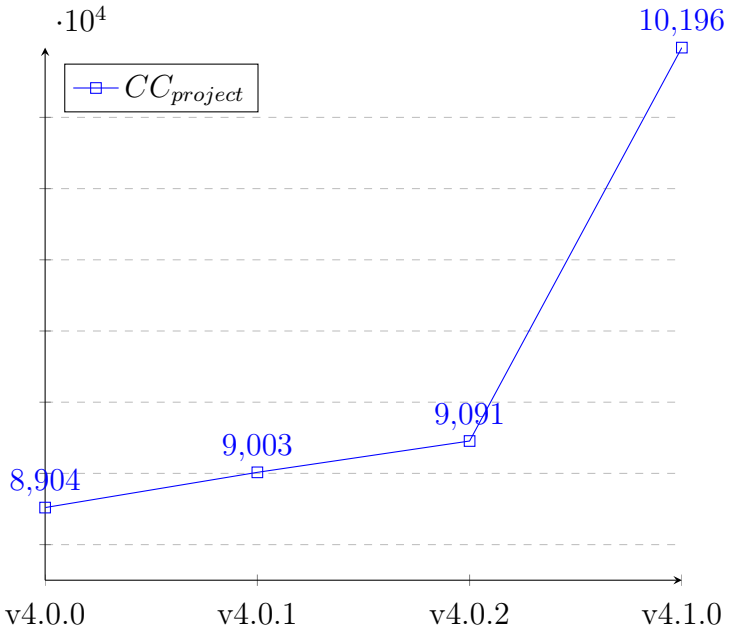


Figura 6: Andamento della *CC* lungo lo sviluppo di Tribuo.

4 Conclusione

L'analisi del codice ha evidenziato dei lievi cambiamenti, dettati dal fatto che Tribuo, essendo un software anche "giovane" (prima release OS settembre 2020) non ha ancora avuto major releases, ma solo patch e minor releases. I plugin utilizzati per l'analisi delle metriche hanno quindi evidenziato questi cambiamenti, sebbene non ci siano state rivoluzioni e modifiche strutturali importanti all'interno del codice: (TODO da vedere come strutturare)

LOC Incremento del parametro dalla v4.0.0 all v4.1.0 del 18

CK WMC : leggera diminuzione percentuale delle classi con un medium-high WMC DIT: leggere incremento delle classi con un low-medium DIT in seguito ad una leggera diminuzione delle classi con un low DIT

NOC: praticamente invariato CBO : aumento (circa 5 percento) delle classi con medium-high CBO between classes, diminuzione delle classi con low CBO

RFC: : leggere incremento delle classi con un low-medium RFC in seguito ad una leggera diminuzione delle classi con un low RFC (simile a cambiamento DIT)

LCM: leggero incremento (1%)classi con high LCM