UNIVERSITÀ DEGLI STUDI DI PADOVA

SCUOLA DI INGEGNERIA

Corso di laurea in INGEGNERIA INFORMATICA

Insegnamento di INGEGNERIA DEL SOFTWARE

HOMEWORK 2

Analisi dell'evoluzione del codice del progetto Jenkins

Gruppo di lavoro: Bolzonello Enrico (matr. 1216351)
De Nat Marco (matr. 1223742)
Lo Conte Riccardo (matr. 1201464)
Moschetta Daniele (matr. 1216411)

Data: 14/07/2021

Anno accademico 2020-2021

SOMMARIO

1. Cos'è Jenkins

Jenkins[1] è un automation server open-source scritto in linguaggio Java. È concepito per automatizzare i processi di sviluppo software e ottimizzare i tempi di scambio e di integrazione di codice nei team di programmazione. Nasce nel 2011 dal codice sorgente di Hudson dopo alcuni contrasti tra il team di sviluppo di questo e Oracle.

Sfruttando la Continuous Integration, pratica che prevede l'integrazione frequente del codice dagli ambienti di lavoro degli sviluppatori verso l'ambiente condiviso, Jenkins permette di evitare il cosiddetto "integration hell", cioè le varie problematiche di integrazione di porzioni di codice sviluppati su lunghi periodi di tempo e indipendenti dalla mainline, e quindi di compiere più velocemente tasks relative alla scrittura del codice, alle fasi di testing e alla distribuzione.

Inoltre, questo software viene eseguito lato server all'interno di un server web che supporta la tecnologia Servlet e può quindi essere utilizzato da remoto all'interno di un web browser.

Principali features:

- è di facile installazione ed è compatibile con i maggiori sistemi operativi del mercato: Windows, Linux, MacOS, Unix, etc.;
- sono disponibili migliaia di plugins che permettono di estendere la funzionalità di base;
- essendo distribuito su più piattaforme, facilita la collaborazione e comunicazione all'interno di un network di più macchine, indipendentemente dalla diversità dei sistemi operativi.

Le release di Jenkins vengono pubblicate periodicamente in base a due "release line":

- Settimanale: rilasci frequenti che includono tutte le nuove caratteristiche, miglioramenti e correzioni di bug;
- Long-Term Support: versione meno recente che viene periodicamente aggiornata tramite backport di correzioni di bug.

2. Analisi dell'evoluzione del codice

2.1 Come verrà analizzato il codice

Il codice di ogni versione del progetto presa in esame verrà analizzata tramite:

- Tabella delle seguenti metriche, prodotta con i tool o3smeasures[2] e cloc[3]:

Metrica	Descrizione
Physical LOC	Numero di righe di codice totali (codice, commenti, vuote)
Logical LOC	Numero delle righe di codice effettive
Number of Classes	Numero totale di classi
Number of Methods	Numero totale di metodi
Number of Attributes	Numero totale di attributi
Cyclomatic Complexity	Numero di percorsi indipendenti attraverso il grafo di controllo di flusso, misura la complessità logica di un certo modulo
Weight Methods per Class (CK)	Misura i metodi di una classe pesati con le loro CC
Depth of Inheritance Tree (CK)	Profondità di una determinata classe nell'albero delle sottoclassi
Number of Children (CK)	Numero di sottoclassi dirette di una classe
Coupling between Objects (CK)	Numero di classi referenziate da una classe
Response for Class (CK)	Numero dei metodi invocati in risposta ad una chiamata ai metodi di una classe
Lack of Cohesion of Methods (CK)	Numero di metodi con basso indice di collaborazione in una classe

  Per ogni parametro verranno riportati: il valore totale, il valore medio per classe, il valore minimo, il valore massimo e la risorsa a cui corrisponde il valore massimo.

- Grafico a torta della ripartizione percentuale delle classi in base a complessità, accoppiamento e mancanza di coesione, prodotto con il tool CodeMR[4]. Questo tool permette di suddividere le classi in diverse fasce in base ai seguenti range di parametri:

	Complexity	Coupling	Lack of Cohesion
Low	$WMC \leq 5, RFC \leq 50, DIT \leq 1$	$CBO \leq 5$	$LCAM \leq 0{,}6$
Low-Medium	$WMC \leq 50, RFC \leq 100, DIT \leq 3$	$CBO \leq 10$	$LCAM \leq 0{,}7$
Medium-High	$WMC \leq 101, RFC \leq 150, DIT \leq 10$	$CBO \leq 20$	$LCAM \leq 0{,}8$
High	$WMC \leq 120, RFC \leq 200, DIT \leq 20$	$CBO \leq 30$	$LCAM \leq 0{,}9$
Very High	$WMC > 120, RFC > 200, DIT > 20$	$CBO > 30$	$LCAM > 0{,}9$

  Questo grafico è utile per determinare se un valore critico di una determinata metrica del progetto sia dovuto ad una bassa percentuale di classi con un valore molto elevato oppure ad una percentuale consistente di classi con un valore medio.

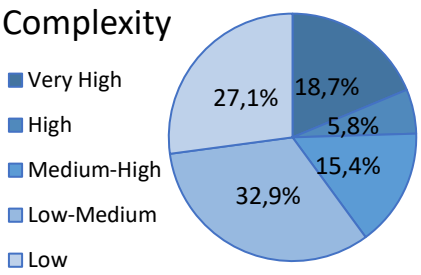
2.2 Analisi delle versioni del progetto selezionate

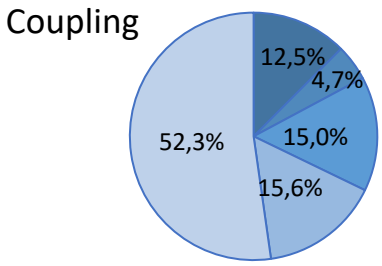
Le versioni del progetto Jenkins che verranno analizzate (scelte tra quelle stabili):

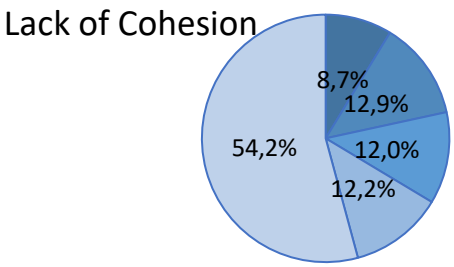
1. Jenkins stable-1.625 pubblicata il 09/12/2015, la meno recente disponibile su GitHub[5];
2. Jenkins stable-2.32 pubblicata il 01/03/2017.
   Principali novità dalla 1.625: retrocompatibilità con le versioni 1.x, introduzione Pipeline as Code, nuova interfaccia grafica, patch problemi di sicurezza SECURITY-360 / CVE-2016-9299[6];
3. Jenkins stable-2.150 pubblicata il 14/02/2019.
   Principali novità dalla 2.32: richiede Java 8, introduzione whitelist delle classi ammesse nel codice[7], nuova interfaccia di setup, patch problemi di sicurezza SECURITY-412 a 420 / CVE-2017-1000356[8];
4. Jenkins stable-2.263 pubblicata il 10/02/2021, la più recente disponibile su GitHub.
   Principali novità dalla 2.150: supporta Java 11, introduzione del dark theme.

1. JENKINS STABLE-1.625 PUBBLICATA IL 9/12/2015

Metrica	Totale	Medio	Minimo	Massimo	Risorsa con valore massimo
Physical LOC	202.400	93,5	1	4.378	Jenkins.java
Logical LOC	109.716	50,7	1	2.373	Jenkins.java
Number of Classes	2.164	-	1	20	Queue.java
Number of Methods	9.121	4,2	4	260	Jenkins.java
Number of Attributes	3.662	1,7	0	91	Jenkins.java
Cyclomatic Complexity	19.729	9,1	1	20	FormFieldValidator.java
Weight Methods per Class	31.002	14,3	1	23	Util.java
Depth of Inheritance Tree	2.143	0,9	1	6	FreeStyleBuild.java
Number of Children	1.071	0,5	0	55	MasterToSlaveCallable.java
Coupling between Objects	2.365	1,1	1	52	IOUtils.java
Response for Class	19.933	9,2	8	853	FilePath.java
Lack of Cohesion of Methods	3.075	1,4	0	155	Jenkins.java

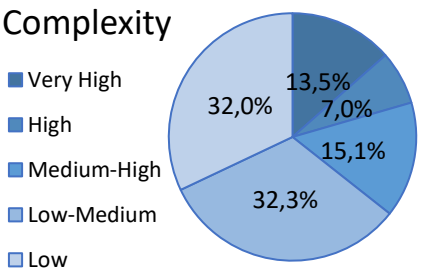


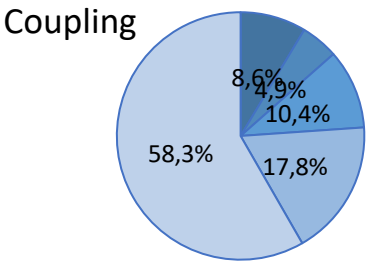


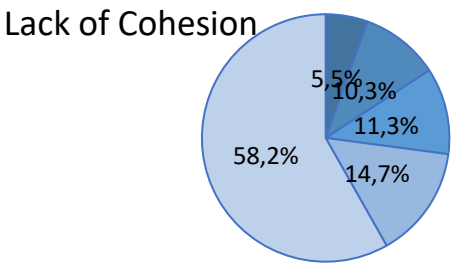


2. JENKINS STABLE-2.32 PUBBLICATA IL 01/03/2017

Metrica	Totale	Medio	Minimo	Massimo	Risorsa con valore massimo
Physical LOC	220.550	97,2	1	5.339	Jenkins.java
Logical LOC	121.571	53,6	1	2.943	Jenkins.java
Number of Classes	2.268	-	1	19	Queue.java
Number of Methods	9.840	4,3	4	290	Jenkins.java
Number of Attributes	3.846	1,7	0	106	Jenkins.java
Cyclomatic Complexity	20.598	9,0	1	20	FormFieldValidator.java
Weight Methods per Class	35.680	15,7	1	27	Jenkins.java
Depth of Inheritance Tree	2.213	0,9	0	6	FreeStyleBuild.java
Number of Children	1.044	0,4	0	56	MasterToSlaveCallable.java
Coupling between Objects	2.549	1,1	1	52	IOUtils.java
Response for Class	21.209	9,3	8	901	FilePath.java
Lack of Cohesion of Methods	3.470	1,5	0	185	Jenkins.java

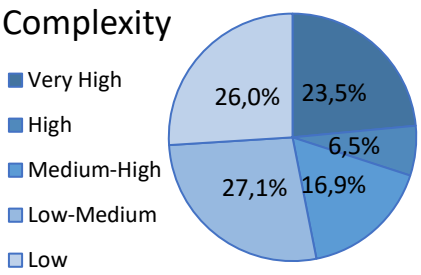


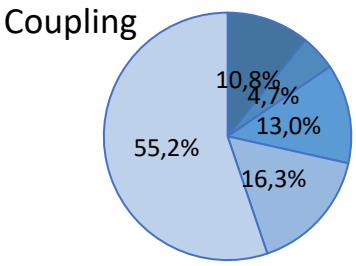


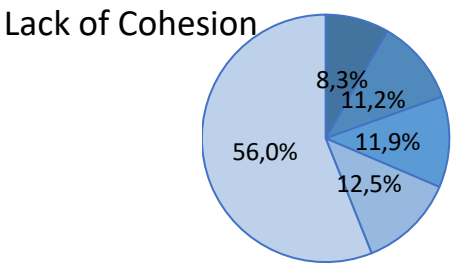


3. JENKINS STABLE-2.150 PUBBLICATA IL 14/02/2019

Metrica	Totale	Medio	Minimo	Massimo	Risorsa con valore massimo
Physical LOC	259.425	93,6	1	5.312	Jenkins.java
Logical LOC	147.205	53,1	1	3.014	Jenkins.java
Number of Classes	2.772	-	1	58	FilePath.java
Number of Methods	11.324	4,0	4	300	Jenkins.java
Number of Attributes	4.650	1,6	2	123	FilePath.java
Cyclomatic Complexity	27.096	9,8	1	56	FilePath.java
Weight Methods per Class	42.890	15,5	1	29	Jenkins.java
Depth of Inheritance Tree	2.556	0,9	1	7	FingerprintCleanupThread...
Number of Children	1.182	0,4	0	63	MasterToSlaveCallable.java
Coupling between Objects	2.869	1,0	1	52	IOUtils.java
Response for Class	24.037	8,7	5	864	FilePath.java
Lack of Cohesion of Methods	4.003	1,4	0	190	Jenkins.java

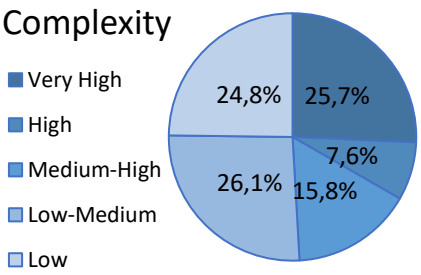


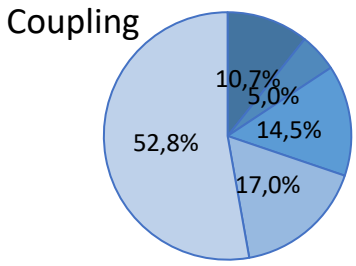


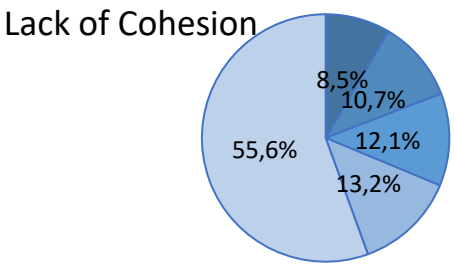


4. JENKINS STABLE-2.263 PUBBLICATA IL 10/02/2021

Metrica	Totale	Medio	Minimo	Massimo	Risorsa con valore massimo
Physical LOC	275.064	94,7	1	5.317	Jenkins.java
Logical LOC	158.002	54,4	1	3.054	Jenkins.java
Number of Classes	2.903	-	1	64	FilePath.java
Number of Methods	12.160	4,2	5	300	Jenkins.java
Number of Attributes	5.009	1,7	2	143	FilePath.java
Cyclomatic Complexity	29.114	10,0	1	53	FilePath.java
Weight Methods per Class	46.756	16,1	1	29	Jenkins.java
Depth of Inheritance Tree	2.617	0,9	1	6	DiagnosedStreamCorrupt...
Number of Children	1.281	0,4	0	57	MasterToSlaveCallable.java
Coupling between Objects	2.930	1,0	1	52	IOUtils.java
Response for Class	25.737	8,9	5	893	Jenkins.java
Lack of Cohesion of Methods	4.378	1,5	0	191	Jenkins.java







3. Confronto tra le versioni e conclusioni

La seguente tabella compara le metriche ottenute per le diverse versioni prese in esame:

	Versioni			
Metrica	**stable-1.625↓**	**stable-2.32→**	**stable-2.150→**	**stable-2.263→**
Physical LOC	202.400	+9%	+18%	+6%
Logical LOC	109.716	+11%	+21%	+7%
Number of Classes	2.164	+5%	+22%	+5%
Number of Methods	9.121	+8%	+15%	+7%
Number of Attributes	3.662	+5%	+21%	+8%
Cyclomatic Complexity	19.729	+4%	+32%	+7%
Weight Methods per Class	31.002	+15%	+20%	+9%
Depth of Inheritance Tree	2.143	+3%	+15%	+2%
Number of Children	1.071	-3%	+13%	+8%
Coupling between Objects	2.365	+8%	+13%	+2%
Response for Class	19.933	+6%	+13%	+7%
Lack of Cohesion of Methods	3.075	+13%	+15%	+9%

Da questi dati è possibile notare come la crescita del progetto sia stata continua e costante, con un maggiore incremento tra la versione stable-2.32 e la stable-2.150. La continuità e l'uniformità della crescita sono dei fattori dovuti anche al contributo che qualsiasi sviluppatore può offrire al progetto Jenkins tramite GitHub.

La seguente tabella[9] riporta i coefficienti di correlazione di Pearson dell'andamento delle metriche nelle diverse versioni del codice che sono state prese in esame:

	PLOC	LLOC	NC	NM	NA	CC	WMC	DIT	NOC	CBO	RPC	LCOM
PLOC	-											
LLOC	1,000	-										
NC	0,993	0,993	-									
NM	0,998	0,998	0,990	-								
NA	0,994	0,994	0,997	0,995	-							
CC	0,989	0,990	0,998	0,988	0,998	-						
WMC	0,998	0,998	0,984	0,998	0,988	0,979	-					
DIT	0,991	0,990	0,999	0,985	0,992	0,996	0,980	-				
NOC	0,931	0,934	0,949	0,943	0,965	0,964	0,920	0,938	-			
CBO	0,993	0,993	0,985	0,986	0,977	0,975	0,991	0,986	0,888	-		
RPC	0,997	0,998	0,989	1,000	0,996	0,988	0,997	0,983	0,949	0,982	-	
LCOM	0,995	0,995	0,977	0,997	0,985	0,973	0,998	0,972	0,922	0,984	0,996	-

- $C \geq 0{,}999$

Il coefficiente di correlazione di Pearson è un valore, compreso tra -1 e $+1$, utile a determinare se tra due variabili esiste una relazione lineare. $C = +1$ indica la massima correlazione proporzionale, $C = -1$ la massima correlazione inversamente proporzionale e $C = 0$ l'assenza di correlazione.

Da questi dati è possibile notare che ogni $C \geq 0{,}888$, il che indica una forte correlazione diretta esistente tra gli andamenti di tutte le metriche prese in esame. In particolare per quanto riguarda le relazioni:

- Physical LOC & Logical LOC: correlazione dovuta al fatto che i LOC logici siano un sottoinsieme di quelli fisici;
- Number of Classes & Depth of Inheritance Tree: correlazione dovuta al fatto che aumentando il numero delle classi, aumenti il numero di classi "figlio" e di conseguenza anche la profondità dell'albero delle sottoclassi;
- Number of Methods & Response for Class: correlazione dovuta al fatto che aumentando il numero di metodi aumenti anche il numero di metodi che possono essere invocati da una determinata classe in risposta alla chiamata ad un proprio metodo.

4. Note e riferimenti

[1] Jenkins (jenkins.io).

[2] O3smeasures (github.com/mariazevedo88/o3smeasures-tool) è uno strumento di analisi statica del codice che permette di analizzarne la qualità e le metriche.

[3] Cloc (github.com/AlDanial/cloc) è uno strumento multi-linguaggio di analisi del codice che fornisce un report sulla suddivisione delle righe di codice (LOC) in base al linguaggio e alla tipologia (commenti, righe vuote, codice.

[4] CodeMR (codemr.co.uk) è uno strumento multi-linguaggio di analisi statica del codice che permette di misurare le metriche e di visualizzarne l'andamento qualitativo anche dal punto di vista grafico.

[5] GitHub (github.com) è servizio di hosting di repository basato sul sistema di controllo versioni distribuito Git.

[6] SECURITY-360 / CVE-2016-9299 era un problema critico di sicurezza di Jenkins che permetteva ad utenti non privilegiati di eseguire codice, bypassando i meccanismi di sicurezza esistenti. (jenkins.io/security/advisory/2016-11-16)

[7] Inizialmente Jenkins utilizzava una blacklist delle classi e dei packages non ammessi stilata in base a exploit noti o sospettati. Dalla versione 2.107 in poi è stato introdotto un sistema basato su una whitelist delle classi e dei packages ammessi nel codice. (github.com/jenkinsci/jep/blob/master/jep/200/README.adoc#concern-about-the-single-whitelist-approach, jenkins.io/blog/2018/03/15/jep-200-lts)

[8] SECURITY-412 a 420 / CVE-2017-1000356 erano dei problemi di sicurezza di Jenkins che permettevano ad utenti malintenzionati di eseguire azioni da amministratore inducendo la vittima ad aprire una pagina web malevola. (jenkins.io/security/advisory/2017-04-26/#cli-unauthenticated-remote-code-execution)

[9] La tabella è simmetrica, per semplicità ne è stata riportata solo la prima metà. I coefficienti di correlazione tra una metrica e sé stessa, ovviamente uguali a $+1$, non sono stati riportati.