Raport finalny

ProjectS

1.Informacje podstawowe

Nazwa zespołu:	2016_L_G2_Project-S
Grupa dziekańska:	2
Członkowie zespołu:	• Paweł Zych • Janusz Sawicki • Michał Mitros • Patryk Cholewa • Jakub Bączek
Link Trello:	https://trello.com/b/YppElUEr/2016-l-g2-project-s
Repozytorium Github:	https://github.com/MichalMitros/ProjectS

2. Motywacja

Uznaliśmy że botnet jest ciekawym projektem, ponieważ umożliwi nam poznanie działania protokołów sieciowych/komunikacyjnych, oraz zrozumienie podstaw działania szkodliwego oprogramowania. Po za tym chcieliśmy być oryginalni i dlatego nie tworzyliśmy kolejnych implementacji sklepów/banków itp.

3. Zgrubny opis aplikacji

Aplikacja kliencka zostaje pobrana na jakąś liczbę komputerów, następnie zostaje odpalona, nawiązuje połączenie z serwerem i oczekuje na komendy - przeprowadzenie DDOS/pobranie pliku z podanego linku/odpalenie innego programu .jar/test połączenia “ping-pong”. Komendy mogą zostać wprowadzone po zalogowaniu się na serwer IRC, lub poprzez skorzystanie ze specjalnie do tego przeznaczonej aplikacji z interfejsem graficznym. Główną ideą jest przeprowadzanie ataków na strony internetowe oraz rozprzestrzenianie kolejnych szkodliwych programów.

4. Opis użytych technologii

4.1. Oprogramowanie

Projekt powstał w obiektowym języku programowania Java, przy pomocy środowiska programistycznego IntelliJ IDEA. Zdecydowaliśmy się także na system kontroli wersji Git. Kod programu przechowywany jest w serwisie Github. Wybór tych narzędzi był spowodowany głównie ich popularnością, dobrymi opiniami oraz własnym doświadczeniem. Do komunikacji i zarządzania projektem wykorzystana była aplikacja Trello.

4.2. Kod

W celu możliwości uzyskania połączenia internetowego z serwerem IRC wykorzystano pakiet java.net. Do przeprowadzenia testów jednostkowych wykorzystano narzędzie JUnit. Projekt posiada także graficzny interfejs użytkownika dla aplikacji Master, wykonany za pomocą biblioteki graficznej Swing.

4.3. Tabela użytych technologii

Język programowania:	Java
Środowisko programistyczne:	IntelliJ IDEA
Kontrola wersji:	Git
Repozytorium:	Github
Zarządzanie projektem:	Trello
Programowanie internetowe:	java.net
Testy jednostkowe:	JUnit
Interfejs użytkownika:	Swing
Server:	digitalocean.com - Debian
Serwer IRC:	inspIRCd

5. Szacowanie pracochłonności

5.1. Metoda COCOMO II

Aby użyć metody COCOMO została wyznaczona ilość tysięcy linii kodu (1684 KLOC), a także odpowiednie wartości współczynników.

Pmnom	A	size	E	
5,204427818	2,94	1,684	1,0958	
Pmadj	Pmnom	PI EM		
8,999420509	5,20442782	1,729185383		
E	B	SIGMA SF		
1,0958	0,91	18,58		
A	2,94		EM	
B	0,91		RELY	0,92
			DATA	0,9
size(KLSOM	1,684		CPLX	0,87
			RUSE	0,95
SF			DOCU	0,81
PREC	3,72		TIME	1
FLEX	0		STOR	1
RESL	4,24		PVOL	0,87
TEAM	4,38		ACAP	1,42
PMAT	6,24		PCAP	1,15
SIGMA	18,58		PCON	0,81
			APEX	1,22
			PLEX	1,19
			LTEX	1,2
			TOOL	1,17
			SITE	0,93
			SCED	1,43
			PI	1,729185383

Otrzymano pracochłonność nominalną na poziomie 5,2 osobomiesięcy oraz wartość dostosowaną na poziomie 9 osobomiesięcy.

5.2. Metoda punktów przypadków użycia

Do zastosowania tej metody trzeba najpierw przygotować kilka przypadków użycia.

Przypadek użycia 1: **Atakowany serwer**

Zakres	System
Poziom	Cel użytkownika
Uczestnicy i interesy	
Master (atakujący)	Chce uniemożliwić działanie wybranego serwera
Odbiorca pliku - klienta	Świadomie lub nieświadomie udostępnia zasoby komputera atakującemu
Atakowany serwer	Odpowiada na odebrane zapytania
Aktor główny	Master (atakujący)
Warunek początkowy	brak
Wyzwalacz	Master postanawia uniemożliwić poprawne działanie serwera

Scenariusz

1. Atakujący rozprzestrzenia plik – klient
2. Atakujący łączy się z serwerem IRC
3. Odbiorcy pliku - klienta uruchamiają go i świadomie lub nieświadomie łączą się z serwerem IRC
4. Połączone z serwerem IRC komputery tworzą „armię botów"
5. "Armia botów" wysyła zapytania na atakowany serwer
6. Atakowany serwer odpowiada na zapytania wielu komputerów jednocześnie co grozi zmniejszeniem jego wydajności

Przypadek użycia 2: **Klient IRC**

Zakres	System
Poziom	Cel użytkownika
Uczestnicy i interesy	
Master	Chce wysłać lub odebrać wiadomość z serwera IRC
Użytkownik IRC	Chce wysłać lub odebrać wiadomość z serwera IRC
Aktor główny	Master (atakujący)
Warunek początkowy	brak
Wyzwalacz	Master postanawia wysłać lub odebrać wiadomość z serwera IRC

Scenariusz

1. Master łączy się z serwerem IRC za pomocą Master GUI
2. Użytkownik IRC łączy się z serwerem za pomocą dowolnego klienta IRC
3. Master wysyła lub odbiera wiadomość z serwera IRC
4. Użytkownik wysyła lub odbiera wiadomość z serwera IRC

Przypadek użycia 3: **Test połączenia**

Zakres	System
Poziom	Cel użytkownika
Uczestnicy i interesy	
Master	Chce sprawdzić stan połączenia botów z serwerem
Bot	Jeśli połączony, odpowiada na słowo klucz
Aktor główny	Master (atakujący)
Warunek początkowy	Bot poprawnie analizuje odebraną wiadomość
Wyzwalacz	Master wykonuje test połączenia

Scenariusz

1. Master rozprzestrzenia plik – klient
2. Master łączy się z serwerem IRC za pomocą Master GUI
3. Master wybiera opcję „TESTINOUT", a następnie wysyła wiadomość o treści „ping"
4. Bot (jeśli jest połączony z serwerem) czyta i analizuje wiadomości z IRC
5. Bot (jeśli jest połączony z serwerem) odpowiada słowem kluczowym „pong" jeśli Master skorzystał z opcji „TESTINOUT" oraz wysłał wiadomość o treści „ping"

Przypadek użycia 4: **Pobieranie pliku**

Zakres	System
Poziom	Cel użytkownika
Uczestnicy i interesy	
Master (atakujący)	Chce pobrać plik na komputer klienta
Odbiorca pliku - klienta	brak
Aktor główny	Master (atakujący)
Warunek początkowy	brak
Wyzwalacz	Master postanawia pobrać plik na komputer klienta

Scenariusz

1. Master rozprzestrzenia plik – klient
2. Master łączy się z serwerem IRC za pomocą Master GUI
3. Odbiorca pliku klienta uruchamia go i łączy się z serwerem IRC
4. Master (przy użyciu odpowiedniej komendy wraz z argumentami: lokalizacja docelowa, adres URL treści pobieranej) pobiera treść na komputer klienta

Teraz na ich podstawie, szacując parametry można obliczyć pracochłonność.

ECF	=	1,34	
	Waga	Ocena	Iloczyn
E1	1,5	1	1,5
E2	0,5	2	1
E3	1	1	1
E4	0,5	1	0,5
E5	1	2	2
E6	2	1	2
E7	-1	5	-5
E8	-1	1	-1
SIGMA			2

UUCP	UAW	UUCW	
17	9	8	
UCP	UUCP	TCF	ECF
9,5676	17	0,42	1,34
PF	Mnożnik	UCP	
191,352	20	9,5676	

TCF	=	0,42	
	Waga	Ocena	Iloczyn
T1	2	4	8
T2	1	1	1
T3	1	1	1
T4	1	3	3
T5	1	0	0
T6	0,5	2	1
T7	0,5	4	2
T8	2	4	8
T9	1	1	1
T10	1	2	2
T11	1	3	3
T12	1	3	3
T13	1	3	3
SIGMA			36

LINK:	http://wazniak.mimuw.edu.pl/images/3/30/Zio-13-wyk.pdf

UAW	9
Aktor	Złożoność
Master	3
Bot/Klient	2
Atakowany	2
Odb. Pliku	2

UUCW	8
Use Case	Złożoność
Nr 1	2
Nr 2	2
Nr 3	2
Nr 4	2

Projekt posiada 9,57 punktów przypadków użycia, co daje mu pracochłonność na poziomie 191,35 roboczogodzin, przy obranym współczynniku pracochłonności wynoszącym 20.

6. Podział ról

Sprint:	Zadania:

02.03	• Wszyscy
	  ○ Podszkolenie umiejętności z Javy
	  ○ Analiza i nauka potrzebnych bibliotek

16.03	• Patryk Cholewa
	  ○ Metody i testy jednostkowe do nich:
	  ■ DownloadFileFromUrl()
	  ■ RunJarFile()
	• Jakub Bączek
	  ○ ScrumMaster
	  ○ Postawienie serwera - DigitalOcean, skonfigurowanie serwera irc - insIRCd
	  ○ Klasa SlaveClient
	  ■ tworzy obiekt klasy Bot - generuje nazwę, inicjalizuje działanie klienta.
	  ○ Klasa Bot
	  ■ łączność z serwerem
	  ■ przekazywanie poleceń do ActionFactory
	  ○ Klasa ActionFactory i interfejs Action
	  ■ ActionFactory - wzorzec projektowy fabryki - tworzy obiekty implementujące Action, które są wykonywane przez obiekt klasy Bot
	  ○ Klasa TestInOut
	  ■ przykładowa klasa implementująca interfejs Action
	  ■ wysyła na serwer wiadomość “pong” w odpowiedzi na “ping”

30.03	• Patryk Cholewa ○ Reimplementacja metod i testów pod nowy interfejs Action

20.04	• Paweł Zych
	  ○ Graficzny projekt GUI, dobór i rozmieszczenie elementów.
	  ○ Analiza pakietu Swing, zaznajomienie z kreatorem.

04.05	● Patryk Cholewa ○ Wyliczanie pracochłonności: ■ Metodą COCOMO II ■ Punktów przypadków użycia ○ Obliczanie złożoności cyklometrycznej McCabe'a: ■ dla metody DownloadFileFromUrlAction() ● Paweł Zych ○ Tworzenie właściwego GUI: ■ Klasa Gui ○ Rozszerzenie logiki bota dla Gui: ■ Klasa Master ● Janusz Sawicki ○ Tworzenie diagramów UML ■ 4 diagramy przypadków użycia wraz z opisami ■ 1 diagram aktywności ● Michał Mitros ○ Klasa zawierająca metodę ataku DDOS przez adres URL ■ Rozpoczynanie/kończenie ataku ■ Wielowątkowość
18.05	● Patryk Cholewa ○ Opis w sprawozdaniu: ■ wyliczania pracochłonności ■ miar oprogramowania ■ swoich testów jednostkowych ● Paweł Zych ○ Zamiana przycisków akcji na oddzielne okienka oraz dodanie okna help. ■ Poszczególne klasy Dialogów ● Janusz Sawicki ○ Dodanie szyfrowania nazw ■ Klasa Crypto ○ Opis w sprawozdaniu ■ Dokumentacja użytkownika

7. Dokumentacja użytkownika

7.1. Wstęp

Stworzony system umożliwia przeprowadzanie ataków DDoS na wybrane serwery. Poniższa instrukcja ma na celu przedstawienie jego funkcjonalności głównej oraz pobocznej.

7.2. Funkcjonalność główna

7.2.1. Przeprowadzanie ataków DDoS:

Na początku użytkownik musi w dowolny sposób udostępnić plik-klient jak największej ilości osób, które muszą dany plik uruchomić. Dzięki temu użytkownik zyska dostęp do zasobów i mocy obliczeniowej wielu komputerów. Aby połączyć się z serwerem oraz zacząć wysyłać zapytania należy wybrać opcję *DDOS Start* z pola *Action*, a następnie w oknie podać adres URL witryny oraz liczbę wątków, których chcemy używać w czasie ataku, a na koniec wysłać polecenie przyciskiem *Send*. Aby zakończyć atak, należy wybrać opcję *DDOS Stop* i wysłać przyciskiem *Send*.

7.3. Funkcjonalność poboczna:

7.3.1. Pobieranie plików na komputer klienta:

Aby pobrać dowolny plik na komputer, na którym uruchomiony jest plik-klient, należy wybrać opcję *Download File From Url*. Zostanie wyświetlone okno *Download File*, a użytkownik zostanie poproszony o podanie adresu URL pliku do pobrania oraz lokalizacji jego zapisu na komputerze klienta. Po ustaleniu argumentów polecenie należy wysłać przyciskiem *Send*.

7.3.2. Uruchomienie pliku wykonywalnego na komputerze klienta:

Aby uruchomić plik wykonywalny na komputerze klienta należy wybrać opcję *Run Jar File*. W oknie *Run Jar File* użytkownik zostanie poproszony o podanie ścieżki do pliku na komputerze klienta. Na koniec należy wysłać polecenie.

7.3.3. Test połączenia:

Aby sprawdzić, czy użytkownik ma dostęp do zasobów komputerów klientów (czy boty poprawnie połączyły się z serwerem IRC), należy wybrać opcję *Connection test* i wysłać polecenie. Jeśli bot jest poprawnie połączony z IRC, użytkownik otrzyma od niego odpowiedź o treści „pong”.

7.3.4. Zerwanie połączenia botów z serwerem IRC:

Aby manualnie anulować połączenie między botami a serwerem IRC należy wybrać opcję *Kill all bots* i wysłać polecenie.

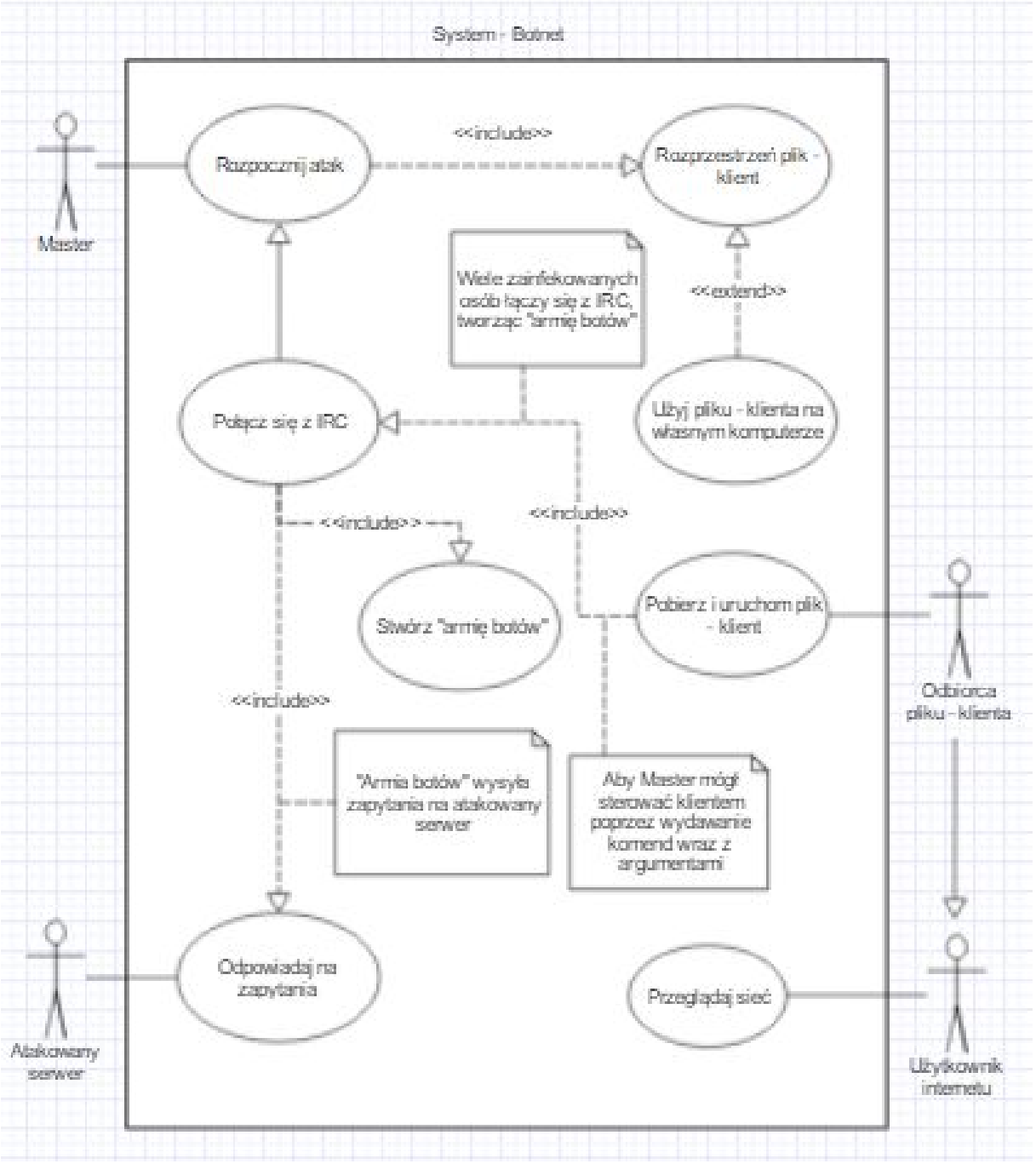
7.3.5. Wysyłanie i odbieranie wiadomości na chacie IRC:

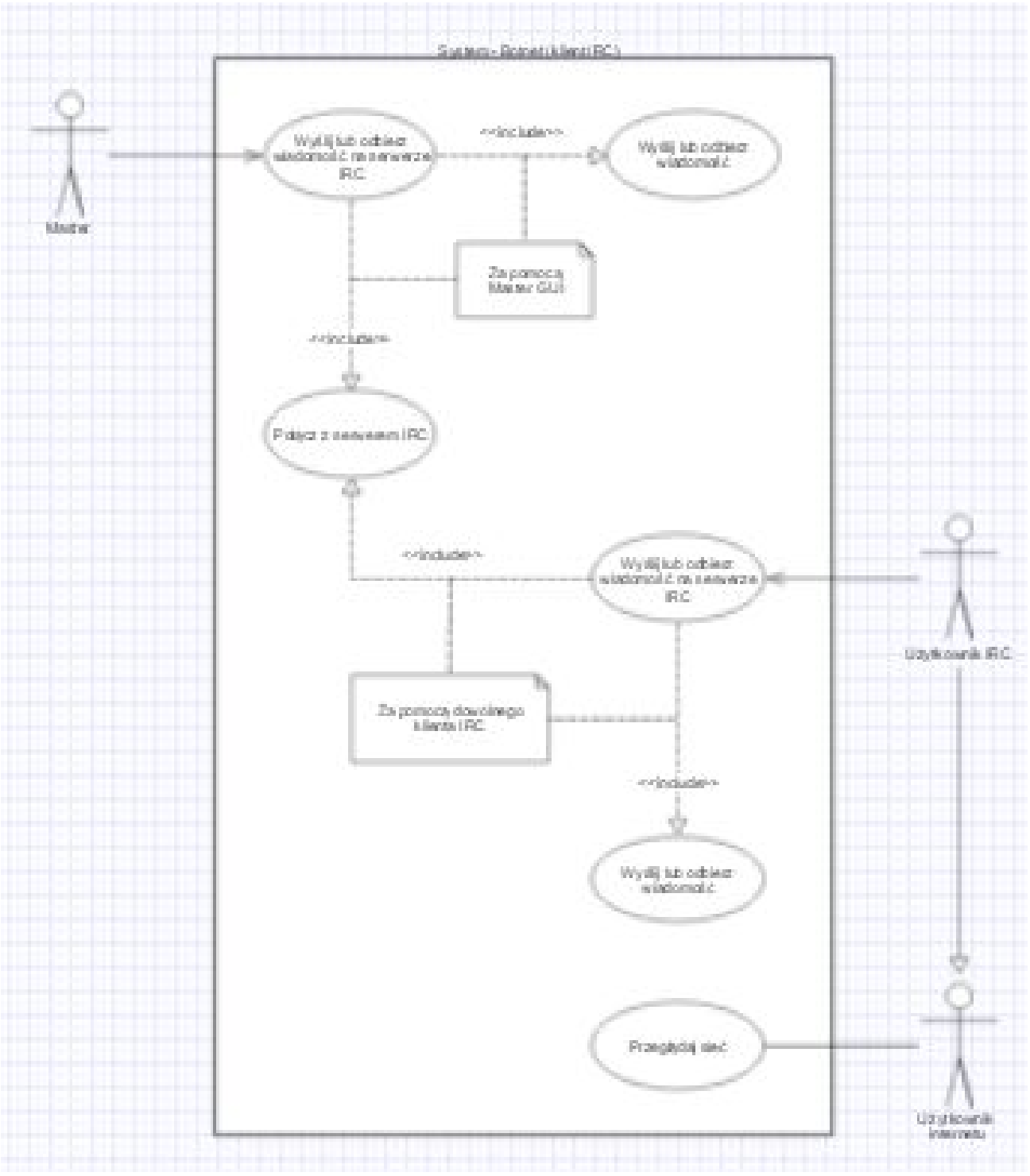
Aby wysłać dowolną wiadomość na chacie IRC, użytkownik musi wpisać treść wiadomości w polu w sekcji *Command* oraz wysłać zawartość pola przyciskiem *Send*. Jeśli użytkownik chce odczytać wiadomości, które wysłali inni użytkownicy chatu, może odczytać je w polu w sekcji *Log*.

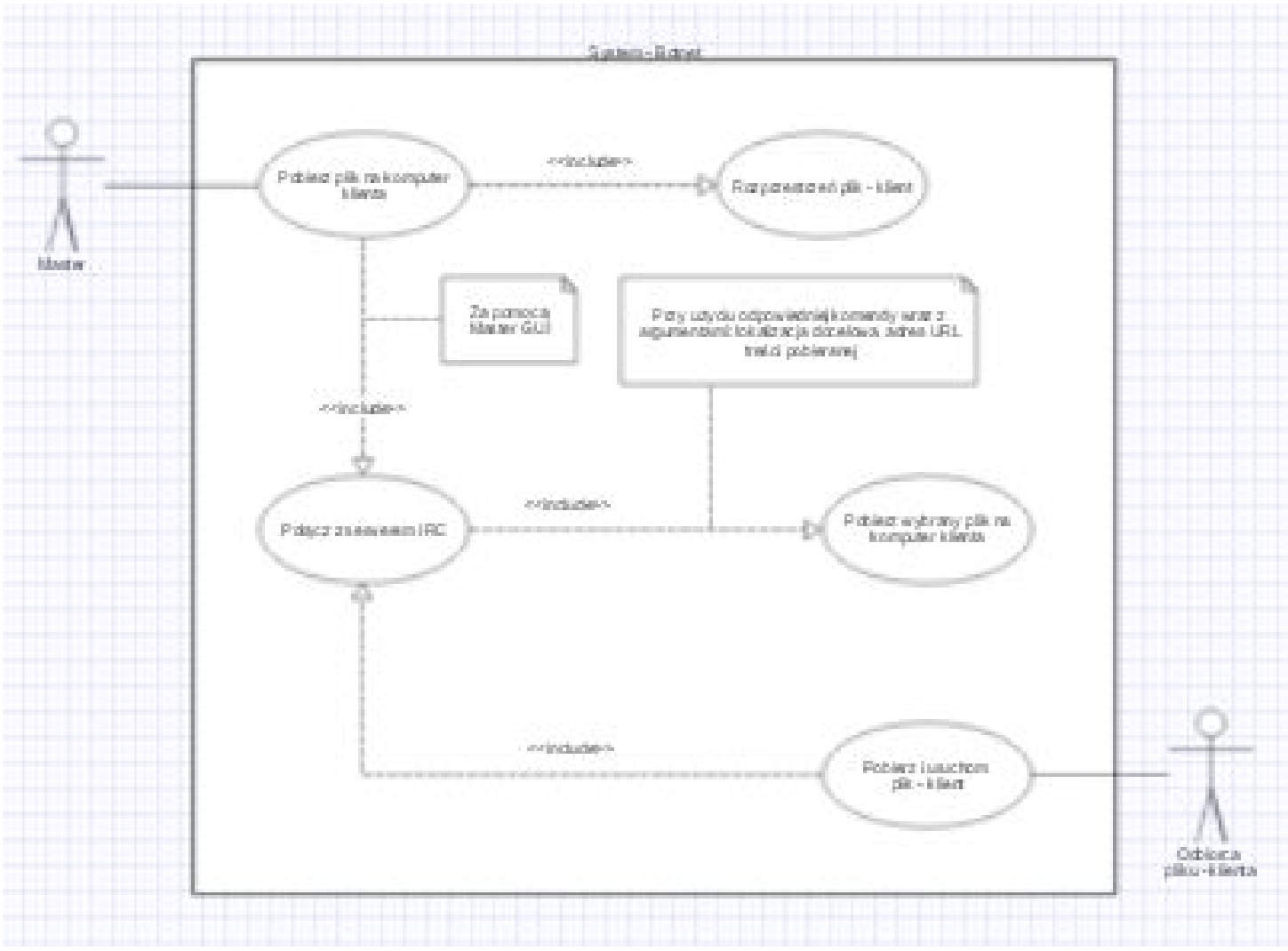
7.3.6. Wyświetlanie pomocy:

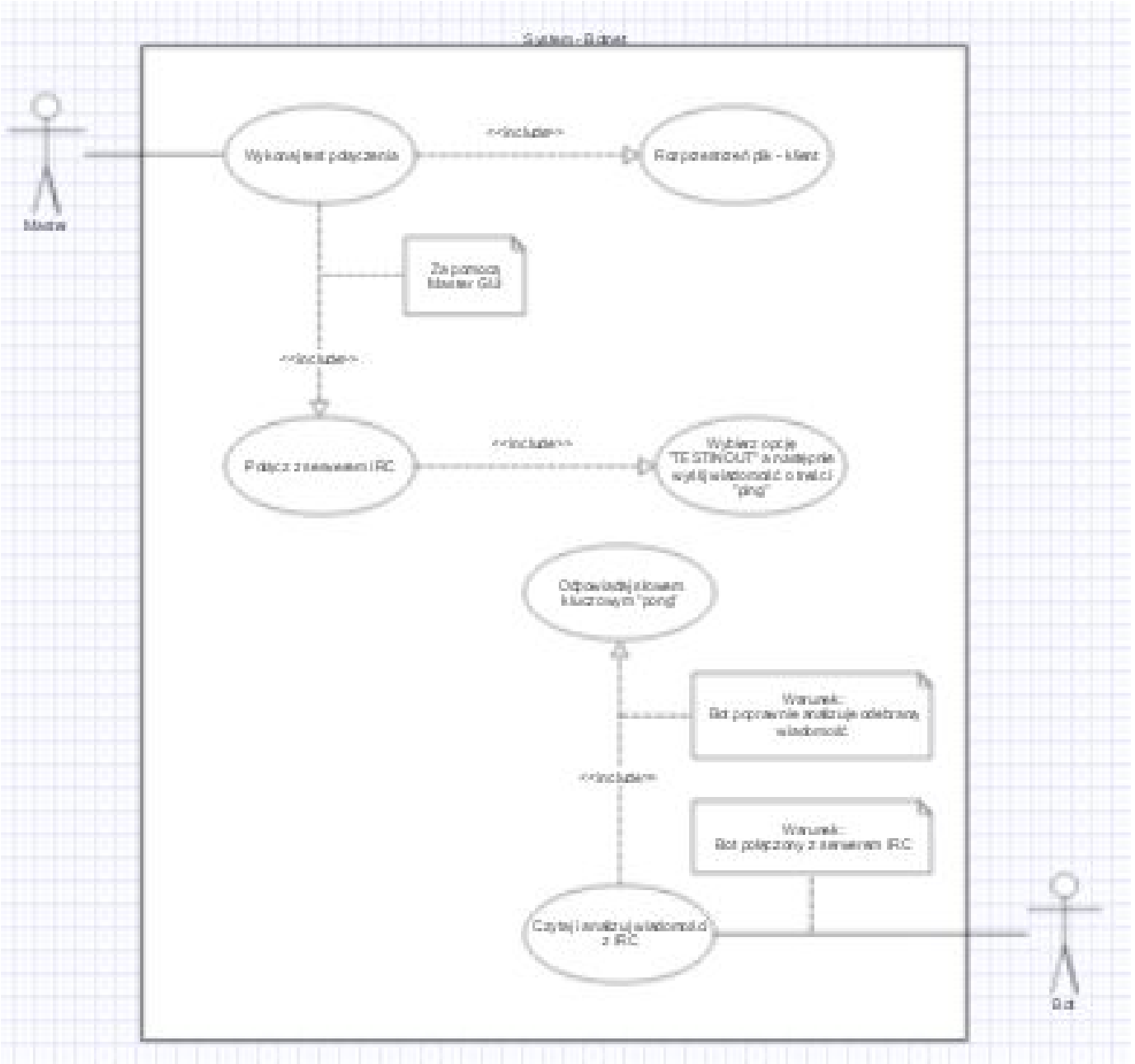
Aby wyświetlić okno pomocy należy wybrać przycisk *Help*.

8. Diagramy przypadków użycia



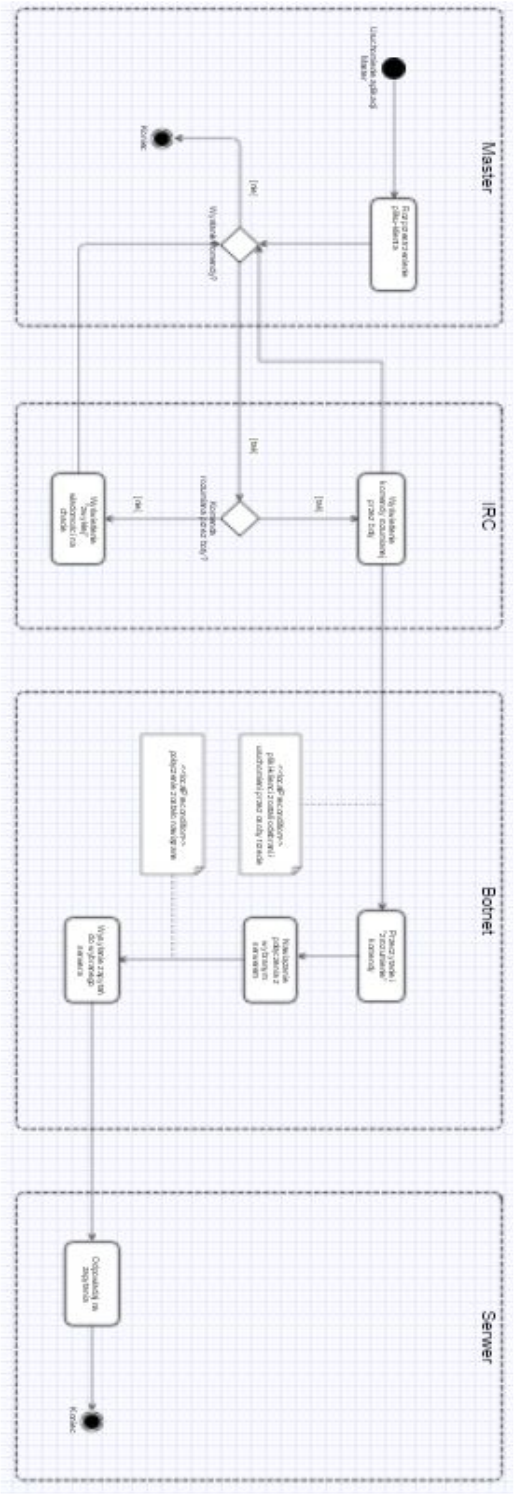

System - Bonsai (klient IRC)
Wyślij lub odbierz wiadomość na serwerze IRC
<<include>>
Wyślij lub odbierz wiadomość
Za pomocą klienta GUI
<<include>>
Połącz z serwerem IRC
<<include>>
Wyślij lub odbierz wiadomość na serwerze IRC
Za pomocą dowolnego klienta IRC
<<include>>
Wyślij lub odbierz wiadomość
Przeglądaj sieć
Użytkownik IRC
Użytkownik Internetu


System - Robot
Właściciel
Pobierz plik na komputer klienta
<<include>>
Rozpoczęcie pobierania pliku - klient
Za pomocą klienta GUI
Przy użyciu odpowiedniej komendy wraz z argumentami lokalizacja docelowa, adres URL treści pobieranej
<<include>>
Połącz z wybranym IRC
<<include>>
Pobierz wybrany plik na komputer klienta
<<include>>
Pobierz i uruchom plik - klient
Odbiorca pliku - klienta


System - Bobot
Master
Wykonaj test połączenia
<<include>>
Plik parametrów pik - klient
Za pomocą Master GUI
<<include>>
Połącz z serwerem IRC
<<include>>
Wybierz opcję "TEST/IN/OUT" a następnie wyślij wiadomość o treści "ping"
Odpowiedź słowem kluczowym "pong"
Warunek:
Bot poprawnie analizuje odebraną wiadomość
<<include>>
Warunek:
Bot połączony z serwerem IRC
Czytaj i analizuj wiadomość z IRC
Bot

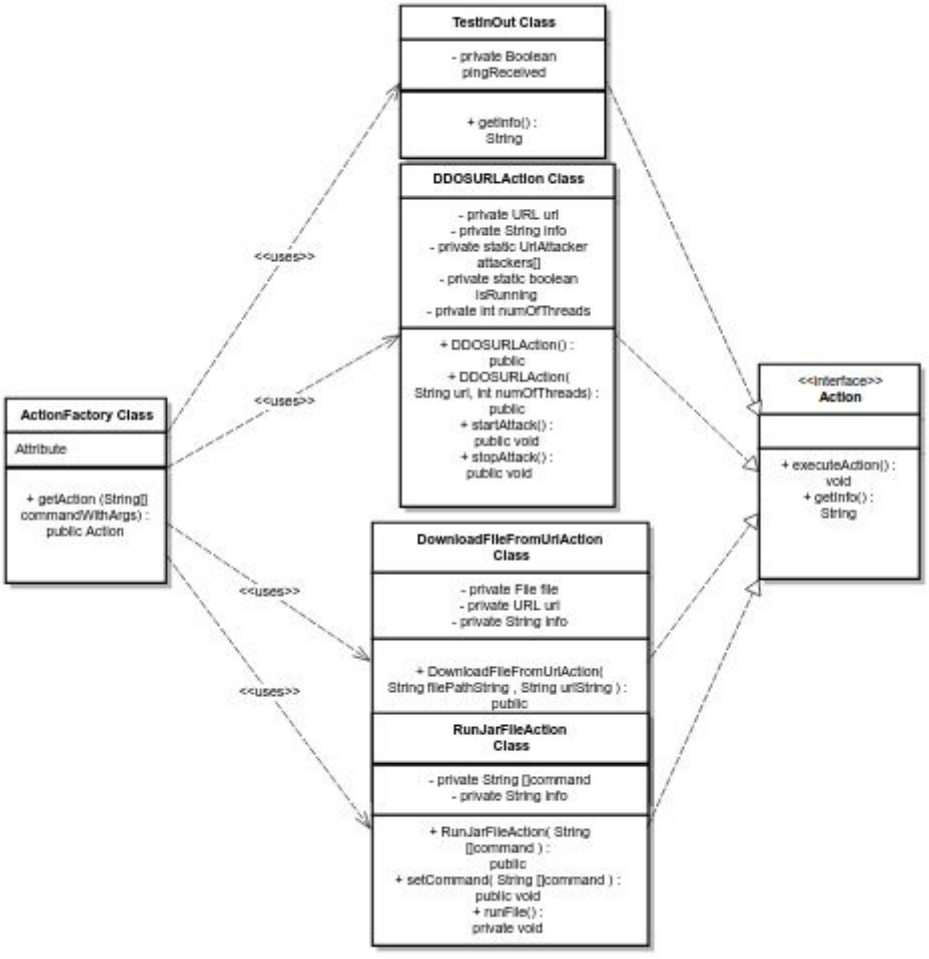
9. Diagram aktywności

Link do dysku z diagramami w lepszej jakości:
https://drive.google.com/drive/folders/0B8tFx_oWVc0rQncwMjlncXpVdWs?usp=sharing



10. Opis wzorca projektowego

Wzorzec fabryki - klasa ActionFactory zwraca obiekty implementujące interfejs Action.
Poniżej przedstawiony diagram UML:



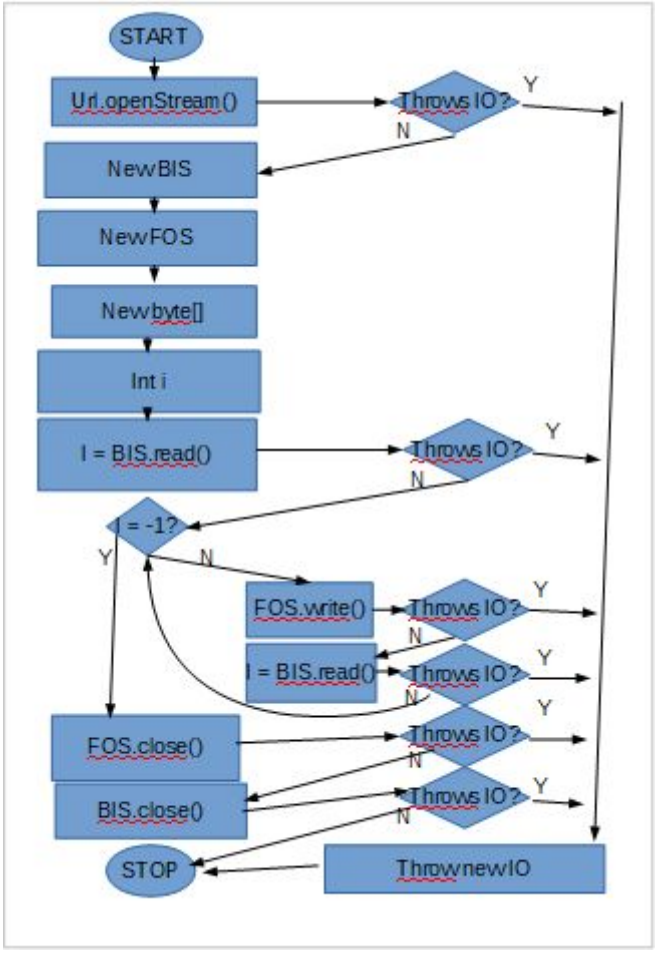
11. Miary jakości kodu

11.1. Ilość linii kodu

Po zsumowaniu wszystkich zasugerowanych przez Github ilości linii kodu dla każdego pliku otrzymano całe 1684 linii kodu [SLOC].

11.2. Złożoność obliczeniowa McCabe’a

Do pomiarów złożoności McCabe’a została wybrana metoda downloadFileFromUrl() z klasy DownloadFileFromUrlAction. Najpierw przedstawiono metodę w postaci schematu blokowego.



Obraz 11.4.1. Schemat blokowy metody

Traktując schemat blokowy jako graf, którego własności można odczytać da się już wyliczyć złożoność cyklometryczną.

CC	E	N	P
8	26	20	1

Obraz 11.4.2. Obliczanie złożoności cyklometrycznej

Złożoność sprawdzanej metody wynosi więc 8 Wobec tego jest ona na jeszcze na granicy posiadania przejrzystego kodu..

12. Przykłady testów jednostkowych

12.1. Dla klasy DownloadFileFromUrlActionTest()

12.1.1. Dla metody executeActionTest()

Metoda ma za zadanie pobrać plik z podanego adresu URL na podaną ścieżkę. Aby ją przetestować trzeba najpierw stworzyć obiekt klasy testowanej i uruchomić metodę.

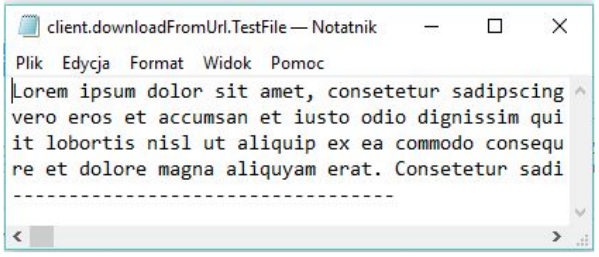
```
@Test
synchronized void executeActionTest() {

    System.out.println( "executeActionTest results depend on internet connection.");

    DownloadFileFromUrlAction action = new DownloadFileFromUrlAction(
            filePathString: "./test/testFiles/client.downloadFromUrl.tmpFile.txt" ,
            urlString: "http://websitetips.com/articles/copy/lorem/ipsum.txt" );

    action.executeAction();
```

Przygotowany został uprzednio plik na komputerze będący kopią tego na stronie, która ma służyć do testów.



Dalej kod testujący otwiera plik szablonowy i pobrany i porównuje ich zawartości.

```
File tmpfile = new File( pathname: "./test/testFiles/client.downloadFromUrl.tmpFile.txt" );
File testfile = new File( pathname: "./test/testFiles/client.downloadFromUrl.TestFile.txt" );

try {
    FileReader tmpFileReader = new FileReader(tmpfile);
    FileReader testFileReader = new FileReader(testfile);
    BufferedReader tmpFileBufferedReader = new BufferedReader(tmpFileReader);
    BufferedReader testFileBufferedReader = new BufferedReader(testFileReader);
    char[] buf = new char[2048];
    int testChar;
    int tmpChar;

    while ((testChar = testFileBufferedReader.read(buf, off: 0, len: 2048)) != -1) {

        assertNotEquals( unexpected: -1 , tmpChar = tmpFileBufferedReader.read(buf, off: 0, len: 2048), message: " Files have different le
        assertEquals(testChar , tmpChar , message: "Files do not match each other!");

    }

    assertEquals( expected: -1 , tmpFileBufferedReader.read(buf, off: 0, len: 2048), message: "Files have different lengths!");
```

Na końcu zamykane są otwarte strumienie. Jeżeli podczas testu wyskoczył jakiś niepożądany Exception, to test uznaje się za nieudany. Sprawdzane jest jeszcze, czy obiekt zwraca odpowiedni komunikat.

```
        testFileBufferedReader.close();
        testFileReader.close();
        tmpFileBufferedReader.close();
        tmpFileReader.close();

    } catch ( IOException e ){
        fail( e.getMessage() );
    }

    assertTrue( action.getInfo().contains( "correctly" ) );

}
```

12.2. Dla klasy RunJarFileAction()

12.2.1. Dla metody executeActionTest()

Aby przetestować tą metodę należało swożyć obiekt testowanej klasy, i uruchomić metodę. Problemem było jednak tu, że metoda tworzyła osobny proces, więc trzeba było stworzyć opóźnienie w testach, aby tworzony proces się wykonał. Na koniec sprawdzenie, czy uruchomiony program zadziałał i czy zwraca się odpowiedni komunikat.

```
    @Test
    synchronized void  executeActionTest() {

        String []command =
                { "./test/testFiles/client.runJarFile.TestFile.jar" , "./test/testFiles/client.run.JarFile.tmpFile" };
        RunJarFileAction action = new RunJarFileAction( command );

        //Test program creates file on given path.
        action.executeAction();

        try {
            Thread.sleep( millis: 2000);
        } catch( InterruptedException e ){
            fail( e.getMessage() );
        }

        assertTrue( new File( pathname: "./test/testFiles/client.run.JarFile.tmpFile").exists() );
        assertTrue( action.getInfo().contains( "correctly" ));

    }

}
```

Test ten wymagał jednak stworzenia osobnego programu przeznaczonego tylko do testów. Jest on bardzo prosty. Tworzy plik o zadanej w pierwszym argumencie ścieżce. Jeżeli nie przekazuje mu się argumentów, to używa ścieżki domyślnej.

```
import java.io.File;
import java.io.IOException;

public class Main {

    public static void main(String[] args) throws IOException {

        if( args.length == 0 ) {
            File file = new File( "./test/testFiles/tmpFile" );
            file.createNewFile();
        } else {
            File file = new File( args[0] );
            file.createNewFile();
        }

    }
}
```