Software Quality Assurance

Difference between two versions Java Telnet daemon

By Hua Jiang

Rivier University

CS680AH2 Software Quality Assurance

Professor : Dr. Vladimir V. Riabov

Spring 2015

Table of Contents:

Abstract:

Telnet was developed in 1969 beginning with RFC 15, extended in RFC 854, and standardized as Internet Engineering Task Force (IETF) Internet Standard. The term TELNET may also refer to the software that implements the client part of the protocol. Telnet client applications are available for virtually all computer platforms.

This is a network protocol used on the Internet or local area networks to provide a bidirectional interactive text-oriented communication facility using a virtual terminal connection.

User data is interspersed in-band with Telnet control information in an 8-bit byte oriented data connection over the Transmission Control Protocol (TCP). This is an open source to implement a telnet daemon that has compact and generic.

Some Main Features are:

Telnet protocol implementation (following specifications, support for NVT, ECHO, TTYPE, NAWS, LINEMODE,NEWENV)Terminal I/O with support for various terminal types Simple UI toolkit as OO layer on top of the basic terminal I/O (work progress)Connection management (host based access and handling of idle connections).

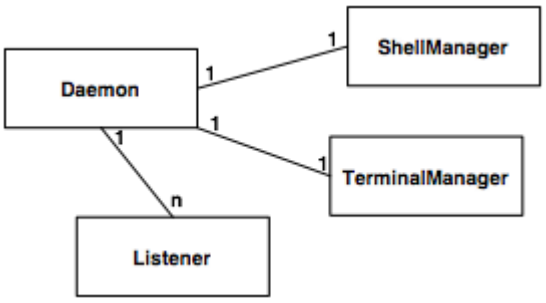
The main focus is design that is flexible and powerful, yet at the same time stable and with a small runtime footprint. Telnet users are many some of them are SUN MICROSYSTEMS, KARANET, OSCAR etc.

1.0 Introduction:

The purpose of the TELNET Protocol is to provide a fairly general, bi-directional, eight-bit byte oriented communications facility. Its primary goal is to allow a standard method of interfacing terminal devices and terminal-oriented processes to each other. It is envisioned that the protocol may also be used for terminal-terminal communication ("linking") and process communication(distributed computation).A TELNET connection is a Transmission Control Protocol (TCP) connection used to transmit data with interspersed TELNET control information.

The TELNET Protocol is built upon three main ideas: first, the concept of a "Network Virtual Terminal"; second, the principle of negotiated options; and third, a symmetric view of terminals and processes.

When deployed, telnetd will create the following set of important and configurable instances:



Telnet D:

Telnet D is a Singleton that will take care of loading and instantiating all required elements including logging.

The unified configuration file specifies all the configuration required for startup, except for logging. It is documented in more detail here.

The Shell Manager:

The Shell Manager is a singleton that will take care of loading and providing shell instances.

The configuration file specifies a special section for the shells that are available as well as their implementing classes. Once you have written an implementation for a shell, this section needs to be updated.

The Terminal Manager:

The Terminal Manager is a singleton that will take care of loading and providing instances of terminal type specific terminal implementations (e.g. ansi, vt100, xterm etc.).

The configuration file specifies in a special section the terminals that are available, their aliases as well as their implementing classes.

Listeners:

Listeners will take care about accepting and managing connections.

For each listener defined in the configuration file there should be a related configuration section that specifies connection and connection management properties.

2.0 Functionality of code:

```
package net.wimpi.telnetd.io.toolkit;

import net.wimpi.telnetd.io.BasicTerminalIO;

import net.wimpi.telnetd.io.TerminalIO;

import net.wimpi.telnetd.io.terminal.ColorHelper;

import java.io.*;

import java.util.ArrayList;

import java.util.Stack;

import java.io.IOException;

import java.io.InputStream;

 import java.io.UnsupportedEncodingException;

import java.io.OutputStream;

 import net.wimpi.telnetd.io.toolkit.Point;

import java.io.IOException;

import java.util.List;

+import java.net.Socket;
```

import net.wimpi.telnetd.io.toolkit.BufferOverflowException;

-import net.wimpi.telnetd.impl.Activator;

++import net.wimpi.telnetd.io.toolkit.Point;

+import java.text.BreakIterator;

+import net.wimpi.telnetd.service.ShellServiceManager;

+import net.wimpi.telnetd.service.TerminalManager;

+import net.wimpi.telnetd.service.TemplatesService;

+import net.wimpi.telnetd.shell.ShellService;

+import net.wimpi.telnetd.util.ThreadPool;

+import org.apache.commons.logging.Log;

+import org.apache.commons.logging.LogFactory;

+import org.osgi.framework.BundleActivator;

+import org.osgi.framework.BundleContext;

+import org.osgi.framework.ServiceReference;

+import org.xml.sax.XMLReader;

+import javax.xml.parsers.SAXParserFactory;

+import java.io.File;
+import org.osgi.framework.BundleContext;

import net.wimpi.telnetd.io.BasicTerminalIO;

import net.wimpi.telnetd.io.terminal.ColorHelper;

import org.apache.commons.logging.Log;

import org.apache.commons.logging.LogFactory;

+import java.io.FileDescriptor;

```
+import java.net.InetAddress;s
+import java.security.Permission;
+import net.wimpi.telnetd.io.toolkit.BufferOverflowException;
+import net.wimpi.telnetd.io.toolkit.Point;
+import java.io.IOException;
+import java.io.OutputStream;
+import java.io.PrintWriter;
+import java.util.ArrayList;
+import java.util.Iterator;
+import java.util.List;
+import java.io.IOException;
+import java.io.InputStream;
+import java.io.OutputStream;
+import org.apache.commons.logging.Log;
+import org.apache.commons.logging.LogFactory;
+import net.wimpi.telnetd.io.terminal.*;
+import net.wimpi.telnetd.service.TerminalManager;
+import org.osgi.framework.BundleContext;
+import java.util.HashMap;
+import java.util.Iterator;
+import java.util.Map;
+import java.util.TreeSet;
+import java.util.ListResourceBundle;
```

+import java.util.HashMap;

+import java.util.Iterator;

+import java.util.Map;

+import org.osgi.service.cm ConfigurationException;.

3.0 Metrics Examination: Release 1.0

3.1 Battle map:

A battle map is a visual representation of program's structure. The display implements the structure chat and represents the relationship between modules program. It also indicates the complexity of each module based on the color (for example, red and yellow for high complexity).

Fig.1(a) Battle map for Java Telnet Release 1.0

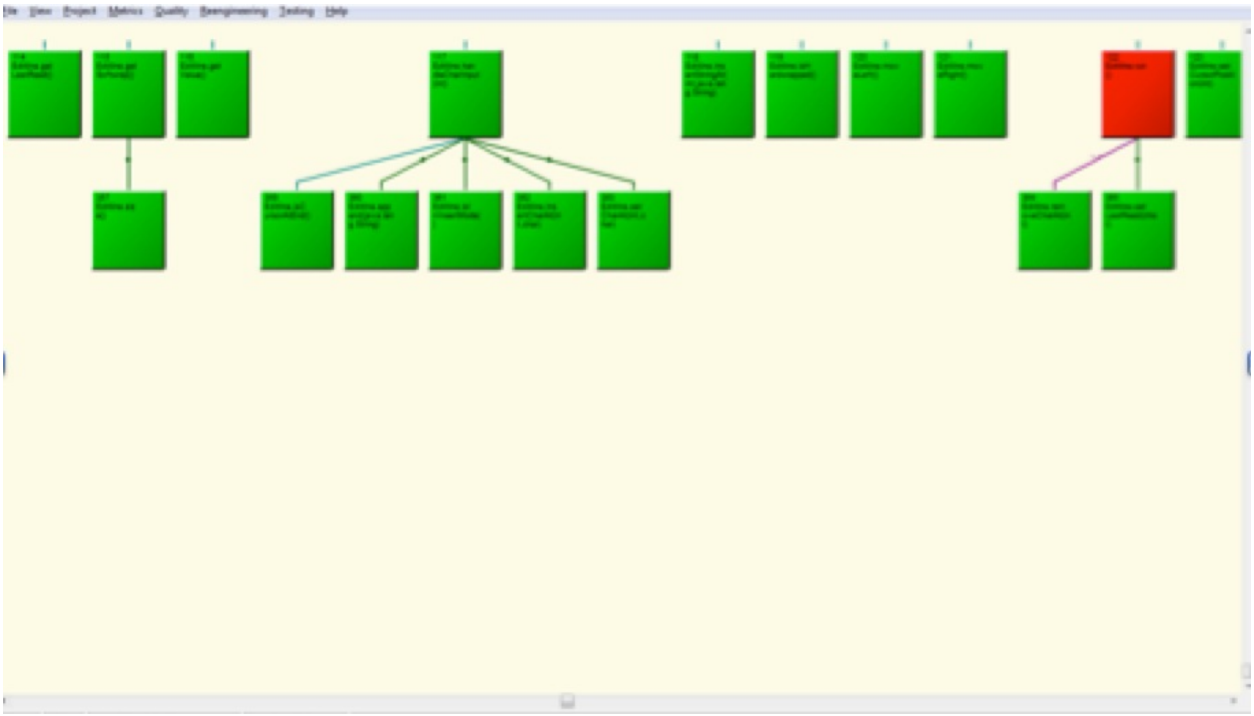


Fig.1(b) Battle map for Java Telnet Release 1.0

There are 8 in red and 3 in yellow which leaves 516 in green.

3.2 McCabe Complexity Metrics Report:

Cyclomatic complexity is defined for each module to be e-n+2 , where e and n are the number of edges and nodes in the control flow graph, respectively. Cyclomatic complexity is also known as v(G) , where v refers to the cyclomatic number in graph theory and G indicates that the complexity is a function of the graph [8]. Here in this module,

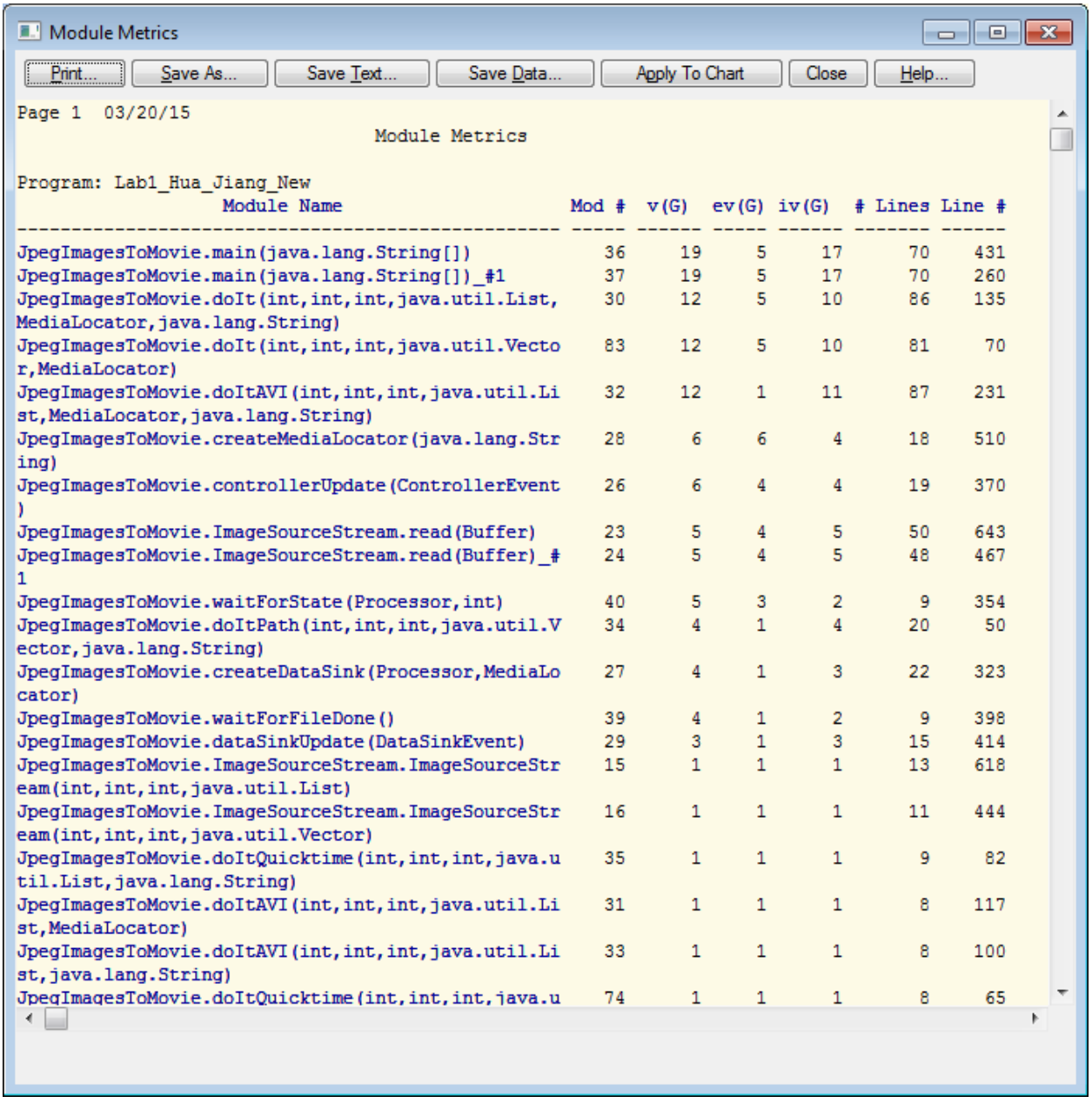
Highest v(G) = 30 for TelnetIO.IACHandler.parse(int[]).

The essential complexity also known as ev(G) of a module is calculated by first removing structured programming primitives from the module's control flow graph until the graph cannot be reduced any further, and then calculating the cyclomatic complexity . In this module,

Highest ev(G) = 20 for TelnetIO.IACHandler.parse(int[]).

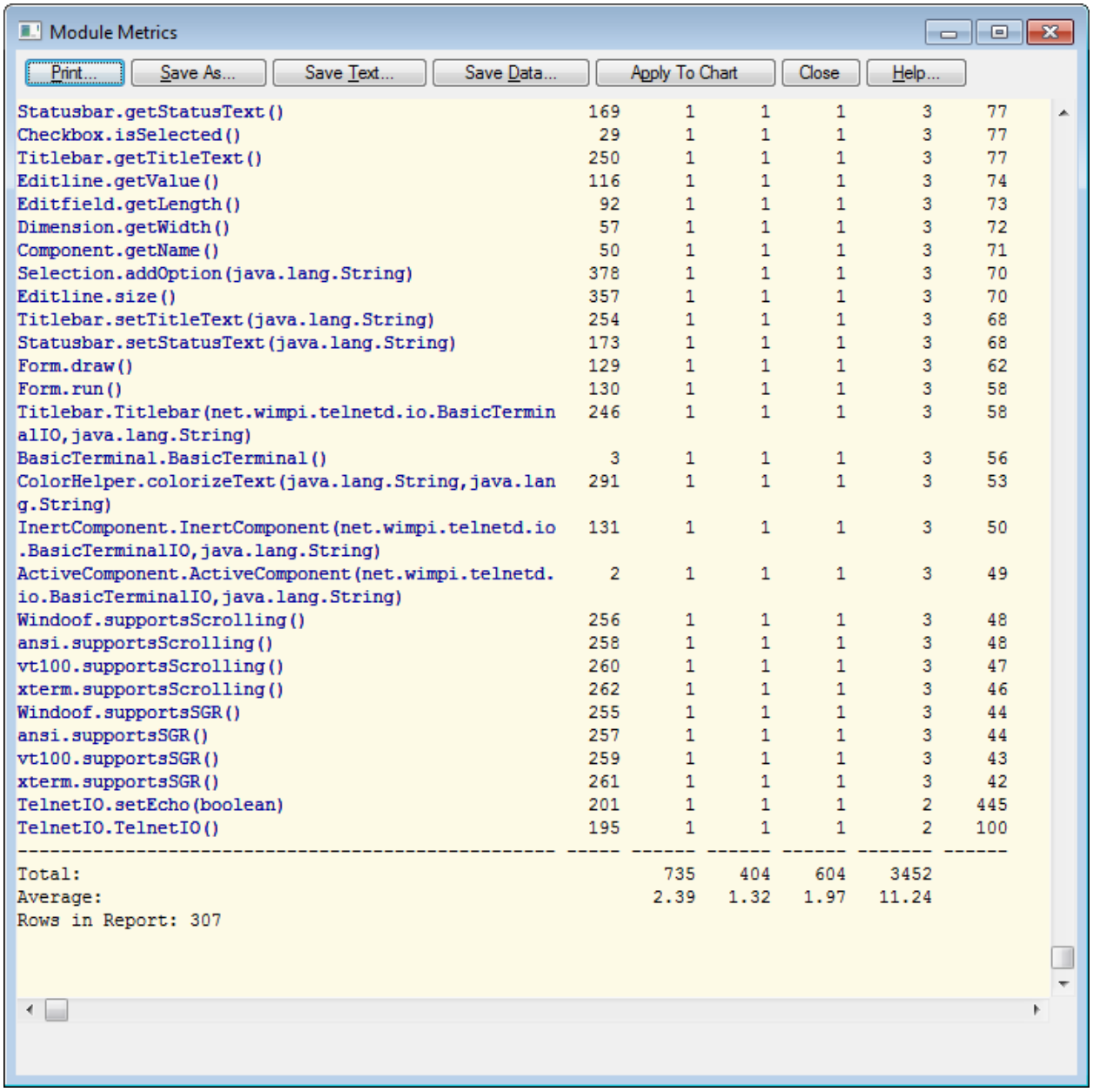
The module design complexity also known as iv(G), is defined as the cyclomatic complexity of the reduced graph after design reduction has been performed. In this module,

Highest iv(G) = 29 for TelnetIO.IACHandler.parse(int[]).



Module Name	Mod #	v(G)	ev(G)	iv(G)	# Lines	Line #
JpegImagesToMovie.main(java.lang.String[])	36	19	5	17	70	431
JpegImagesToMovie.main(java.lang.String[])_#1	37	19	5	17	70	260
JpegImagesToMovie.doIt(int,int,int,java.util.List, MediaLocator,java.lang.String)	30	12	5	10	86	135
JpegImagesToMovie.doIt(int,int,int,java.util.Vecto r,MediaLocator)	83	12	5	10	81	70
JpegImagesToMovie.doItAVI(int,int,int,java.util.Li st,MediaLocator,java.lang.String)	32	12	1	11	87	231
JpegImagesToMovie.createMediaLocator(java.lang.Str ing)	28	6	6	4	18	510
JpegImagesToMovie.controllerUpdate(ControllerEvent)	26	6	4	4	19	370
JpegImagesToMovie.ImageSourceStream.read(Buffer)	23	5	4	5	50	643
JpegImagesToMovie.ImageSourceStream.read(Buffer)_# 1	24	5	4	5	48	467
JpegImagesToMovie.waitForState(Processor,int)	40	5	3	2	9	354
JpegImagesToMovie.doItPath(int,int,int,java.util.V ector,java.lang.String)	34	4	1	4	20	50
JpegImagesToMovie.createDataSink(Processor,MediaLo cator)	27	4	1	3	22	323
JpegImagesToMovie.waitForFileDone()	39	4	1	2	9	398
JpegImagesToMovie.dataSinkUpdate(DataSinkEvent)	29	3	1	3	15	414
JpegImagesToMovie.ImageSourceStream.ImageSourceStr eam(int,int,int,java.util.List)	15	1	1	1	13	618
JpegImagesToMovie.ImageSourceStream.ImageSourceStr eam(int,int,int,java.util.Vector)	16	1	1	1	11	444
JpegImagesToMovie.doItQuicktime(int,int,int,java.u til.List,java.lang.String)	35	1	1	1	9	82
JpegImagesToMovie.doItAVI(int,int,int,java.util.Li st,MediaLocator)	31	1	1	1	8	117
JpegImagesToMovie.doItAVI(int,int,int,java.util.Li st,java.lang.String)	33	1	1	1	8	100
JpegImagesToMovie.doItQuicktime(int,int,int,java.u	74	1	1	1	8	65

Fig 2(a).First page of Module Metrics for Java Telnet Release1.0



Statusbar.getStatusText()	169	1	1	1	3	77
Checkbox.isSelected()	29	1	1	1	3	77
Titlebar.getTitleText()	250	1	1	1	3	77
Editline.getValue()	116	1	1	1	3	74
Editfield.getLength()	92	1	1	1	3	73
Dimension.getWidth()	57	1	1	1	3	72
Component.getName()	50	1	1	1	3	71
Selection.addOption(java.lang.String)	378	1	1	1	3	70
Editline.size()	357	1	1	1	3	70
Titlebar.setTitleText(java.lang.String)	254	1	1	1	3	68
Statusbar.setStatusText(java.lang.String)	173	1	1	1	3	68
Form.draw()	129	1	1	1	3	62
Form.run()	130	1	1	1	3	58
Titlebar.Titlebar(net.wimpi.telnetd.io.BasicTerminalIO,java.lang.String)	246	1	1	1	3	58
BasicTerminal.BasicTerminal()	3	1	1	1	3	56
ColorHelper.colorizeText(java.lang.String,java.lang.String)	291	1	1	1	3	53
InertComponent.InertComponent(net.wimpi.telnetd.io.BasicTerminalIO,java.lang.String)	131	1	1	1	3	50
ActiveComponent.ActiveComponent(net.wimpi.telnetd.io.BasicTerminalIO,java.lang.String)	2	1	1	1	3	49
Windoof.supportsScrolling()	256	1	1	1	3	48
ansi.supportsScrolling()	258	1	1	1	3	48
vt100.supportsScrolling()	260	1	1	1	3	47
xterm.supportsScrolling()	262	1	1	1	3	46
Windoof.supportsSGR()	255	1	1	1	3	44
ansi.supportsSGR()	257	1	1	1	3	44
vt100.supportsSGR()	259	1	1	1	3	43
xterm.supportsSGR()	261	1	1	1	3	42
TelnetIO.setEcho(boolean)	201	1	1	1	2	445
TelnetIO.TelnetIO()	195	1	1	1	2	100
Total:		735	404	604	3452	
Average:		2.39	1.32	1.97	11.24	

Rows in Report: 307

Fig2(b). Final data for Module Metrics in Java Telnet Release1.0

3.3 Scatter plot:

The scatter plot report plots the maximum and minimum values for selected metrics. A Scatter plot report is a graph of one metric value plotted against another metric value, giving a high-level view of a system's complexity by showing the distribution of the modules' metrics. Here is the scatter diagram:

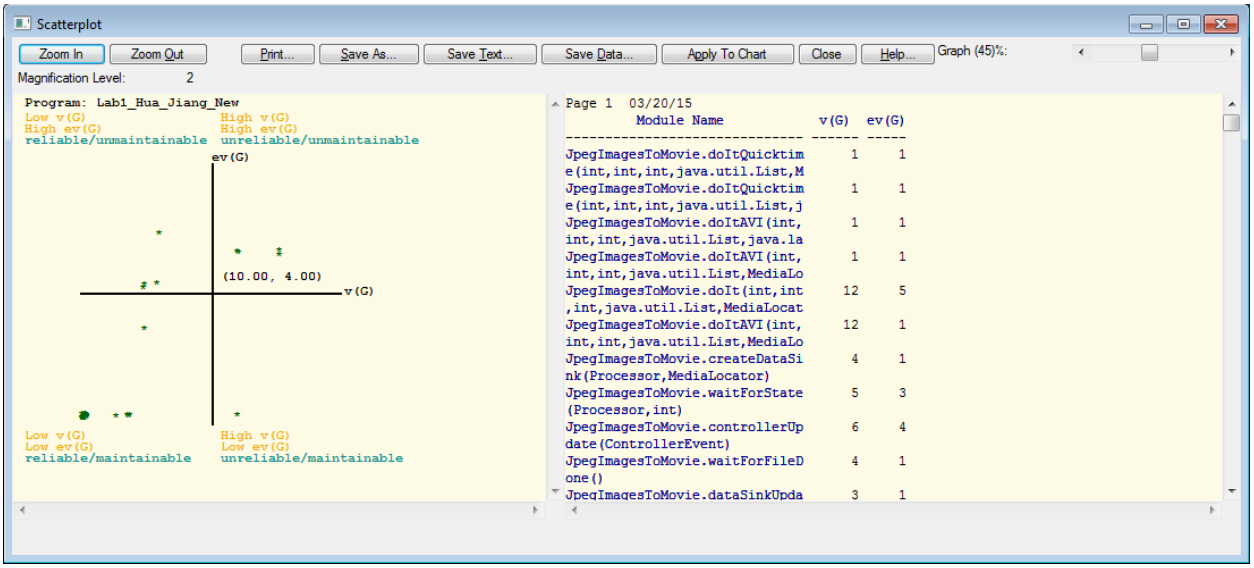


Fig 3(a) Scatter plot for Java Telnet Release 1.0

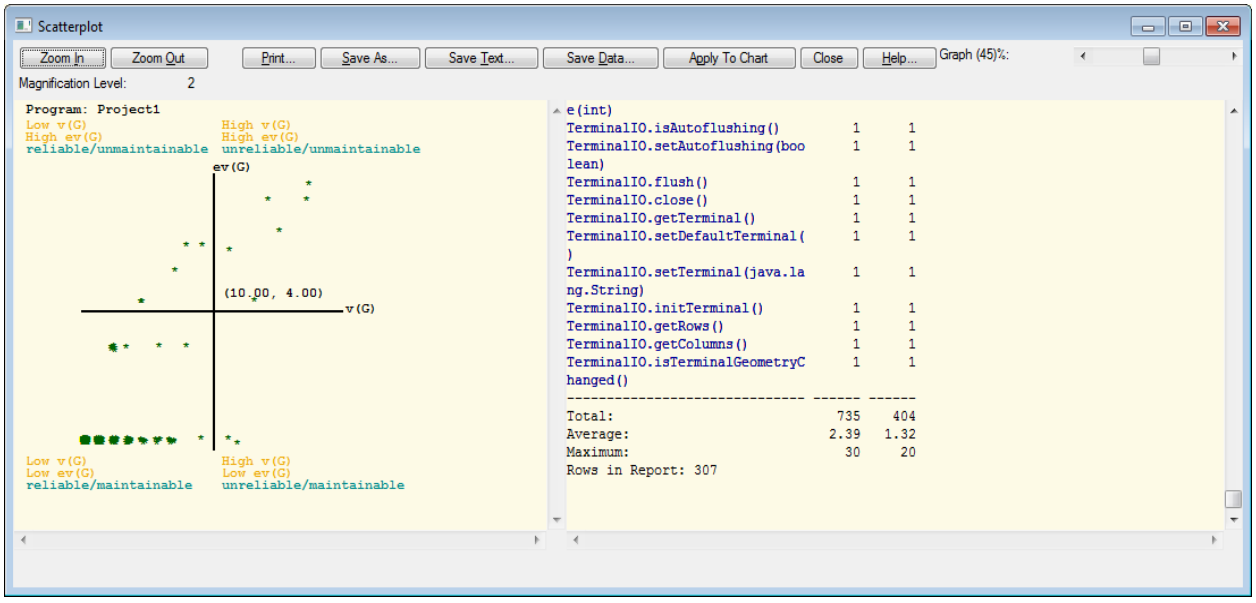


Fig 3(b) Scatter plot for Java Telnet Release 1.0

Based on the scatter plot, there are 6 modules that are unreliable and unmaintainable.

The 6 modules with the highest complexity for Version 1.0 are:

- TelnetIO.IACHandler.parse(int[ ])
- Editarea.run()
- Editarea.run()
- Editline.run()
- Pager.page(java.lang.string[])
- TelnetIO.IACHandler.setWait(int,int,boolean)

In addition, there are 4 modules that are reliable/unmaintainable, and 6 modules that are unreliable/maintainable. This means that there are 515 modules that are reliable/maintainable(subtracting from the total of 527 modules).

3.4 Flow graphs of Selected Modules:

The intention is to look at the flow graphs of Low, Medium, and High complexity to see the difference in the diagrams.

Flow graph is a visual representation of the logic structure in a module. Here is the Highest complexity

(v(G) = 30) is TelnetIO.IACHandler.parse(int[ ]).

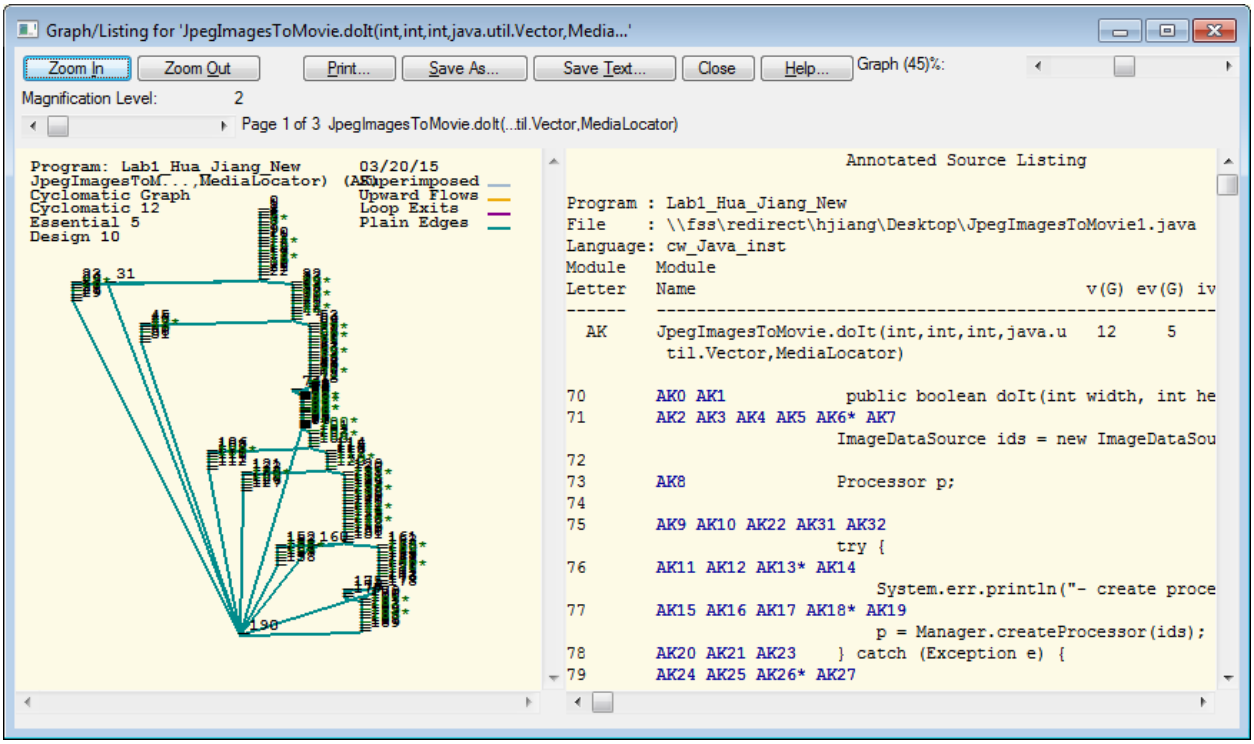


Fig 4.Flowgraph of high complexity for Java Telnet Release 1.0

For medium complexity, the Editarea.insert newline() module was selected with v(G) of 5:

Fig 5.Flowgraph of medium complexity for java Telnet Release 1.0

For Low complexity TelnetIO.IACHandler.skipToSE() module was selected:

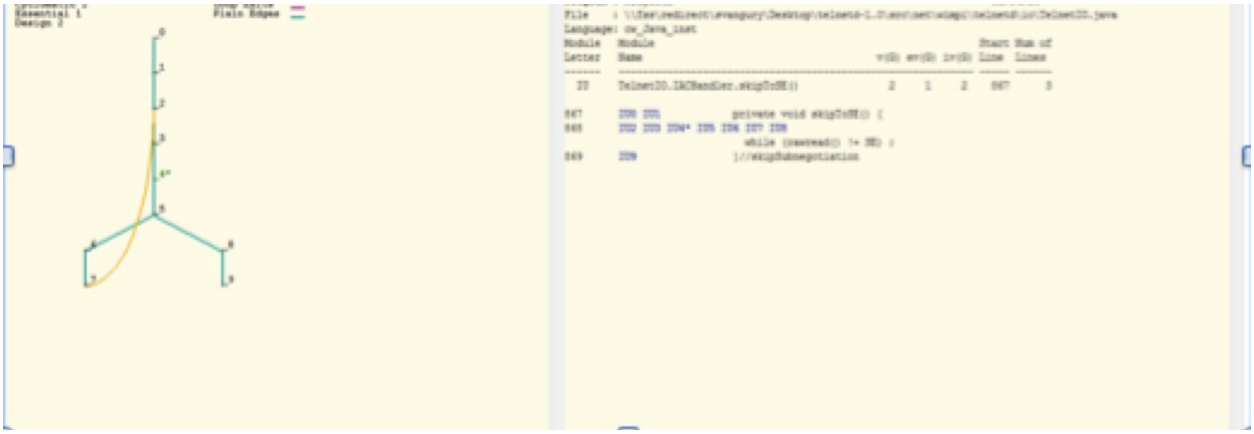


Fig 6.Flowgraph of low complexity for java Telnet Release 1.0

3.5 Independent path for Unit Tests:

As the most vulnerable module is the one with the highest complexity, there are 30 paths for the TelnetIO.IACH.....r.ParseInt( )module. Which has a complexity (v(G)) of 30.

Path 9 of 30:

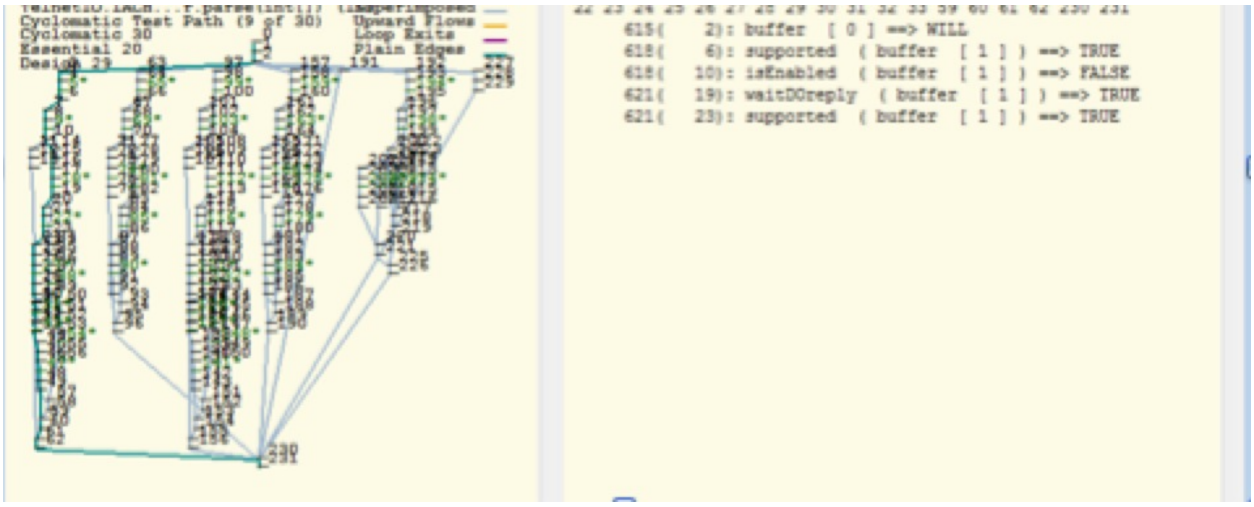


Fig 7. Path 9 of Unit Tests for high complexity for Java Telnet Release 1.0

Path 15 of 30:

Fig 8.Path 15 of Unit Tests for high complexity for Java Telnet Release 1.0

Path 30 of 30:

Fig 9.Path 30 of Unit Tests for high complexity for Java Telnet Release 1.0

3.6 Halstead's Metric Report:

Halstead complexity metrics were developed by the late Maurice Halstead as a means of determining a quantitative measure of complexity directly from the operators and operands in the module to measure a program module's complexity directly from source code. Halstead´s metrics is based on interpreting the source

code as a sequence of tokens and classifying each token to be an operator or an operand.

The following are counted:

- number of unique (distinct) operators (n1)
- number of unique (distinct) operands (n2)
- total number of operators (N1)
- total number of operands (N2).

All other Halstead's measures are derived from these four quantities using the following set of formulas.

Page 1 04/02/15

Halstead Metrics

Program: Lab2_Hua_Jiang_Metrics

Module Name	N	V	L	D	I	E	B	T
JpegImagesToMovie.main(java.lang.String[])	395	2361.03	0.02	42.15	56.01	99526.31	0.79	5529.24
JpegImagesToMovie.doIt(int,int,int,java.util.Vector,MediaLocator)	357	2256.93	0.04	23.14	97.55	52215.38	0.75	2900.85
JpegImagesToMovie.ImageSourceStream.read(Buffer)	245	1435.21	0.04	28.26	50.79	40554.80	0.48	2253.04
JpegImagesToMovie.createMediaLocator(java.lang.String)	103	510.28	0.06	16.88	30.24	8611.01	0.17	478.39
JpegImagesToMovie.createDataSink(Processor,MediaLocator)	90	454.00	0.08	12.53	36.24	5686.89	0.15	315.94
JpegImagesToMovie.controllerUpdate(ControllerEvent)	87	408.94	0.09	11.14	36.70	4556.74	0.14	253.15
JpegImagesToMovie.doItPath(int,int,int,java.util.Vector,java.lang.String)	77	394.95	0.10	10.13	39.01	3998.92	0.13	222.16
JpegImagesToMovie.dataSinkUpdate(DataSinkEvent)	54	233.38	0.10	9.78	23.87	2281.98	0.08	126.78
JpegImagesToMovie.ImageSourceStream.ImageSourceStream(int,int,int,java.util.Vector)	50	219.62	0.12	8.18	26.84	1796.86	0.07	99.83
JpegImagesToMovie.waitForState(Processor,int)	38	166.91	0.12	8.13	20.54	1356.13	0.06	75.34
JpegImagesToMovie.waitForFileDone()	31	129.27	0.14	7.00	18.47	904.87	0.04	50.27
JpegImagesToMovie.ImageDataSource.ImageDataSource(int,int,int,java.util.Vector)	26	106.27	0.18	5.63	18.89	597.79	0.04	33.21
JpegImagesToMovie.prUsage()	18	64.53	0.29	3.50	18.44	225.85	0.02	12.55
JpegImagesToMovie.ImageSourceStream.getContentDescriptor()	10	31.70	0.19	5.25	6.04	166.42	0.01	9.25
JpegImagesToMovie.ImageSourceStream.getControls()	8	24.00	0.33	3.00	8.00	72.00	0.01	4.00
JpegImagesToMovie.ImageDataSource.getControls()	8	24.00	0.33	3.00	8.00	72.00	0.01	4.00
JpegImagesToMovie.ImageDataSource.getContentType()	6	15.51	0.50	2.00	7.75	31.02	0.01	1.72
JpegImagesToMovie.ImageDataSource.getStreams()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
JpegImagesToMovie.ImageDataSource.getDuration()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
JpegImagesToMovie.ImageSourceStream.getContentLength()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
JpegImagesToMovie.ImageSourceStream.endOfStream()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
JpegImagesToMovie.ImageDataSource.getControl(java.lang.String)	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
JpegImagesToMovie.ImageSourceStream.getControl(java.lang.String)	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
JpegImagesToMovie.ImageSourceStream.willReadBlock()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
JpegImagesToMovie.ImageDataSource.getLocator()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
JpegImagesToMovie.ImageSourceStream.getFormat()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
JpegImagesToMovie.ImageDataSource.start()	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00
JpegImagesToMovie.ImageDataSource.connect()	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00
JpegImagesToMovie.ImageDataSource.setLocator(MediaLocator)	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00
JpegImagesToMovie.ImageDataSource.disconnect()	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00
JpegImagesToMovie.ImageDataSource.stop()	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Total:	1644	8908.53	8.76	213.20	551.35	222762.97	2.96	12375.75
Average:	53.03	287.37	0.28	6.88	17.79	7185.90	0.10	399.22

Rows in Report: 31

Fig 10(a).Halsted Metrics for Java Telnet Release 1.0

Totals from end of report:

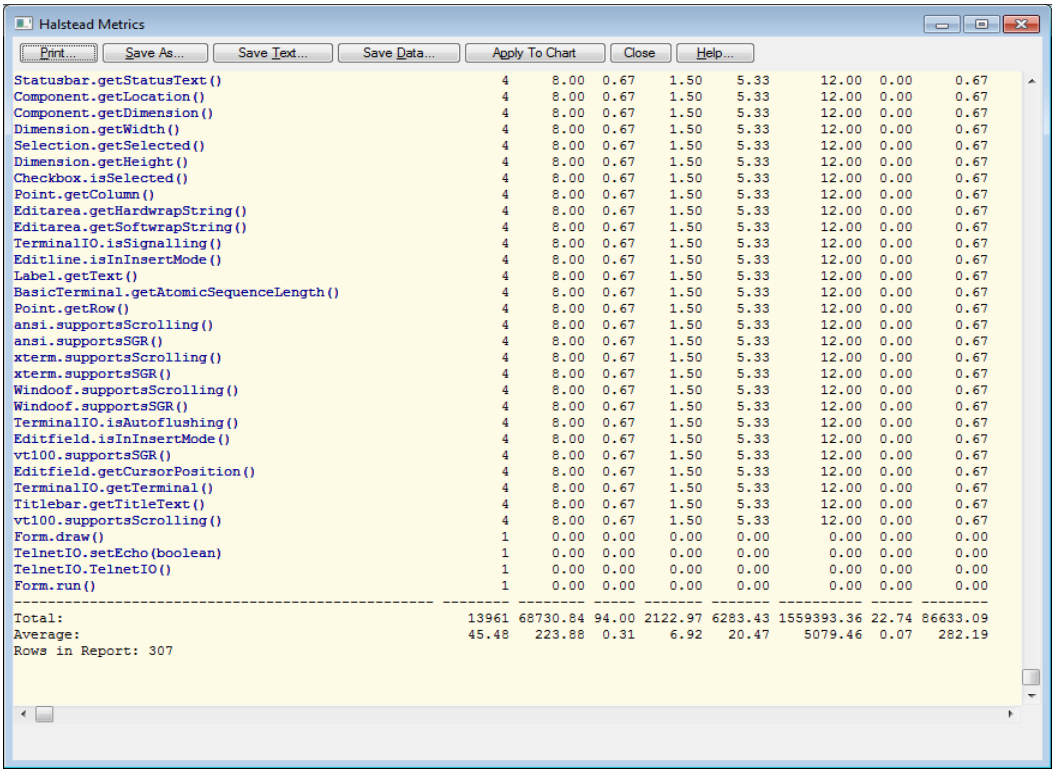


Fig 10(b). Final page of Halsted Metrics for Java Telnet Release 1.0

Program length (N)

The program length (N) is the sum of the total number of operators and operands in the program:

N = N1 + N2=9 . Average program length(N) for this version is 45.48

The module with the largest N is colorizer.main(java.main.string[7]) with a value of 843.

Program volume (V):

The program volume (V) is the information contents of the program, measured in mathematical bits. It is calculated as the program length times the 2-base logarithm of the vocabulary size (n) :

V = N * log2(n) [7]. Average program volume (V) for this version is 223.88

The module with the largest V value is colorizer.main(java.main.string[7]) with a value of 5201.25

Program level (L):

The difficulty level or error proneness (D) of the program is proportional to the number of unique operators. D is also proportional to the ration between the total number of operands and the number of unique operands (i.e. if the same operands are used many times in the program, it is more prone to errors) [7].

D = (n1 / 2) * (N2 / n2) . Here the average difficulty level (D) is 6.92.

Editarea.run( ) has the highest D value of 53.94

Effort to implement (E):

The effort to implement (E) or understand a program is proportional to the volume and to the difficulty level of the program [7].

E = V * D. Here the average effort to implement (E) is 5079.46

The modules with the highest E value is colorizer.main(java.lang.string[7]) with 229715.62

Number of delivered bugs (B):

The number of delivered bugs (B) correlates with the overall complexity of the software. Halstead gives the following formula for B:

B = (E ** (2/3)) / 3000 ** stands for "to the exponent"

Halstead's delivered bugs (B) is an estimate for the number of errors in the implementation. Delivered bugs in a file should be less than 2[7]. In this case average number of delivered bugs (B) is 0.07.

Colorizer.main(java.lang.string[7]) also has the highest value of 1.73 for B.

Time to implement (T):

The time to implement or understand a program (T) is proportional to the effort. Empirical experiments can be used for calibrating this quantity. Halstead has found that dividing the effort by 18 give an approximation for the time in seconds [7].

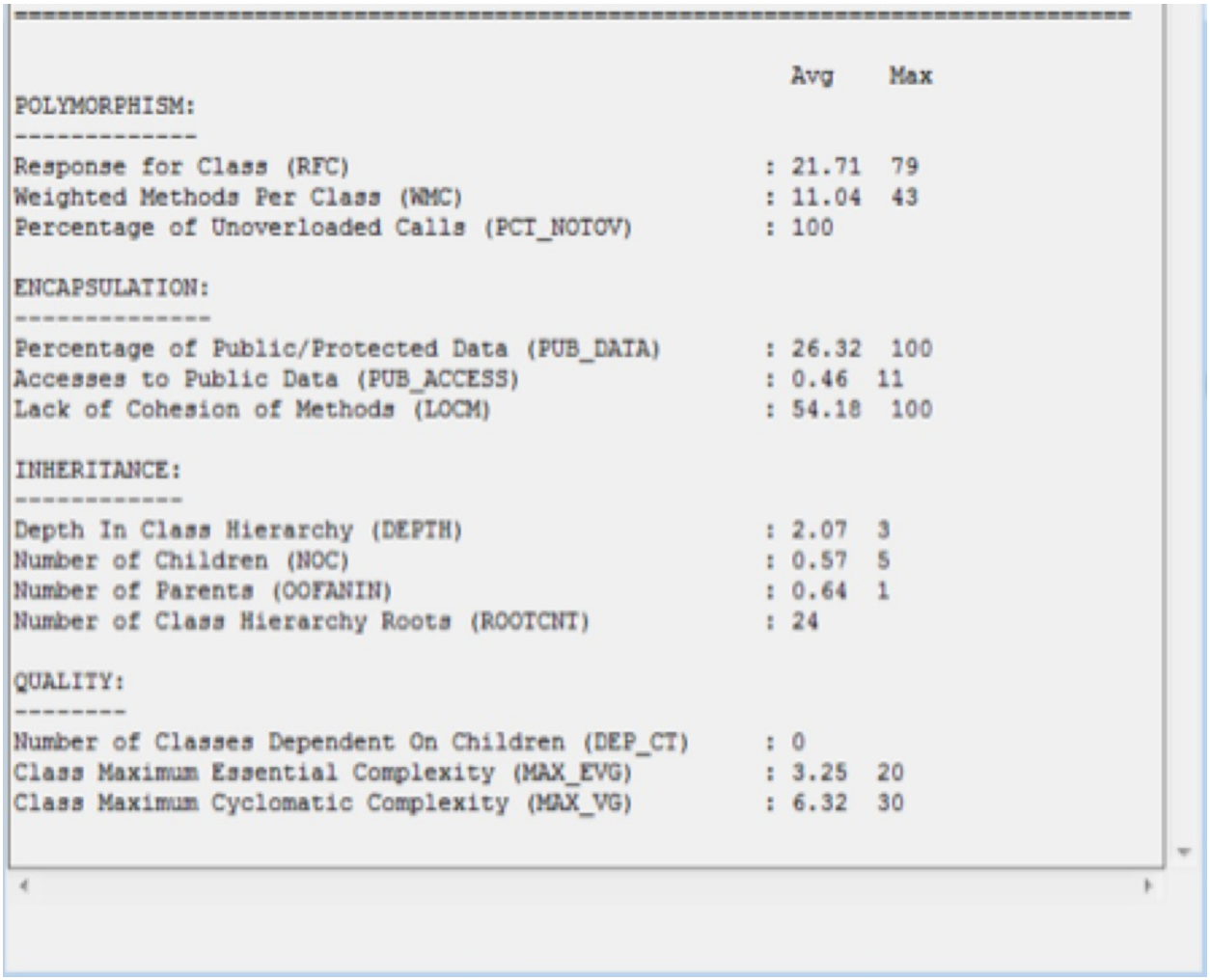
T = E / 18 . Here the average Time to implement(T) is 282.19

Colorizer.main(java.lang.string[]) has the highest value at 12763.98

3.7 Object-Oriented Metrics Report:

TelnetIO6IACHandler	124	5.39	30	20	0	1	23	23	1	100	0	0	0	
Editarea	79	3.16	30	13	0	3	35	25	0	77	1	0	0	
TerminalIO	77	1.79	4	1	0	2	79	43	0	96	1	0	88	
Editfield	67	2.48	18	7	0	3	38	27	0	75	1	0	0	
TelnetIO	58	2.76	9	3	0	1	44	21	1	97	0	0	78	
Editline	58	2.06	16	12	0	1	29	28	0	74	0	0	0	
BasicTerminal	45	3.46	8	3	4	2	28	15	0	86	1	2	100	
Pager	42	3.50	14	6	0	1	12	12	0	76	0	0	0	
Selection	34	2.83	7	3	0	3	22	12	0	71	1	0	40	
Checkbox	20	2.22	4	1	0	3	19	9	0	85	1	0	45	
Colorizer	18	1.80	5	1	0	1	10	10	0	97	0	0	0	
ColorHelper	18	1.38	3	1	0	1	13	13	0	100	0	0	100	
Titlebar	17	1.89	6	3	0	3	18	9	0	85	1	0	43	
Statusbar	17	1.89	6	3	0	3	18	9	0	85	1	0	43	
CharBuffer	12	1.20	2	1	2	1	10	10	0	40	0	0	0	
Component	8	1.13	2	1	2	1	8	8	0	75	0	11	100	
Point	8	1.00	1	1	0	1	8	8	0	25	0	0	0	
Label	6	1.20	2	1	0	3	14	5	0	40	1	0	0	
Dimension	6	1.00	1	1	0	1	6	6	0	33	0	0	0	
Form	3	1.00	1	1	0	3	13	3	0	100	1	0	100	
ansi	2	1.00	1	1	0	3	30	2	0	0	1	0	0	
xterm	2	1.00	1	1	0	3	30	2	0	0	1	0	0	
Windoof	2	1.00	1	1	0	3	30	2	0	0	1	0	0	
vt100	2	1.00	1	1	0	3	30	2	0	0	1	0	0	
ActiveComponent	1	1.00	1	1	5	2	10	2	0	0	1	0	0	
InertComponent	1	1.00	1	1	3	2	9	1	0	0	1	0	0	
Editfield$Buffer	1	1.00	1	1	0	2	11	1	0	0	1	0	0	
Editline$Buffer	1	1.00	1	1	0	2	11	1	0	0	1	0	0	
Total:	729	51.14	177	91	14	58	408	309	2	1517	18	13	737	
Average:	26.04	1.83	6.32	3.25	0.57	2.07	21.71	11.04	0.07	54.18	0.64	0.46	26.32	
Rows in Report: 28														

Fig 11.Class metrics from object-oriented metrics report for Java Telnet Release 1.0



```
================================================================================

                                                    Avg    Max
POLYMORPHISM:
-------------
Response for Class (RFC)                          : 21.71  79
Weighted Methods Per Class (WMC)                  : 11.04  43
Percentage of Unoverloaded Calls (PCT_NOTOV)      : 100

ENCAPSULATION:
--------------
Percentage of Public/Protected Data (PUB_DATA)    : 26.32  100
Accesses to Public Data (PUB_ACCESS)              : 0.46  11
Lack of Cohesion of Methods (LOCM)                : 54.18  100

INHERITANCE:
------------
Depth In Class Hierarchy (DEPTH)                  : 2.07  3
Number of Children (NOC)                          : 0.57  5
Number of Parents (OOFANIN)                       : 0.64  1
Number of Class Hierarchy Roots (ROOTCNT)         : 24

QUALITY:
--------
Number of Classes Dependent On Children (DEP_CT)  : 0
Class Maximum Essential Complexity (MAX_EVG)      : 3.25  20
Class Maximum Cyclomatic Complexity (MAX_VG)      : 6.32  30
```

Fig 12.Program Metrics from Object-Oriented Metrics Report for Java Telnet Release 1.0

The Object Oriented (OO) metrics McCabe provides are the following broken down by aspect of OO design [8]:

Polymorphism

RFC – Response for Class - A count of methods implemented within a class plus the number of methods accessible to an object of this class due to inheritance. In general, lower values indicate greater polymorphism. Here the average response for class is 21.71.

WMC – Weighted Methods per Class - A count of methods implemented within a class (rather than all methods accessible within the class hierarchy). In general,

lower values indicate greater polymorphism. Average weighted methods per class for this version is 11.04

PCT_NOTOV – Percent of Unoverloaded Calls - The percentage of calls that are not made to overloaded modules. This metric is generated for the entire program, not individual classes. In general, lower values indicate greater polymorphism. Average percent of unoverloaded calls is 100.

Encapsulation :

PUB_DATA – Percent Public Data - The percentage of data that is public and protected data in a class. In general, lower values indicate greater encapsulation. Average percent public data is26.32.

PUB_ACCESS – Access to Public Data - The amount of times that a class's public and protected data is accessed. In general, lower values indicate greater encapsulation Average access to public data is 0.46.

LOCM – Lack of Cohesion of Methods - For each data field in a class, the percentage of the methods in the class using that data field; the percentages are averaged then subtracted from 100%. The LOCM metric indicates low or high percentage of cohesion. If the percentage is low, the class is cohesive. If it is high, it may indicate that the class could be split into separate classes that will individually have greater cohesion. Average lack of cohesion of methods is 54.18.

Inheritance:

DEPTH – Depth - The level for a class. For example, if a parent class has one child, the depth for the child is 2. Depth indicates at what level a class is located within its class hierarchy. In general, inheritance increases when depth increases. Average depth is 2.07.

NOC – Number of Children - The number of classes derived from a specified class. Average number of children is 0.57.

OOFANIN – Number of Parents (fan-in) - The number of classes from which a class is derived. Average number of parents are 0.64.

ROOTCNT – Number of Class Hierarchy Roots - The sum of the class hierarchy roots within a program (that is, the number of distinct class hierarchies employed by a program). Average number of class hierarchy root is 24.

CBO – Coupling between Objects - The number of distinct non-inheritance-related classes on which a class depends. If a class that is heavily dependent on many classes outside of its hierarchy is introduced into a library, all the classes upon which it depends need to be introduced as well. This may be acceptable, especially if the classes which it references are already part of a class library and are even more fundamental than the specified class. Average Coupling between objects is 0.07.

Quality:

DEP_CT – Number of Classes Dependent on Children – The number of classes dependent on descendants. A higher value would indicate poor quality in the OO design. Average number of classes dependent on children is 0.

MAX_EVG – Class Maximum Essential Complexity - The maximum essential complexity of a module within all the classes in a program/section. Average class maximum essential complexity is 3.25.

MAX_VG – Class Maximum Cyclomatic Complexity - The maximum cyclomatic complexity of a module within all the classes in a program. Average class maximum cyclomatic complexity is 6.32.

4.0 Metrics Examination – version 2.0

4.1 Battle map:

There are too many modules here are few screenshots for Battlemap of java telnet daemon 2.0 version

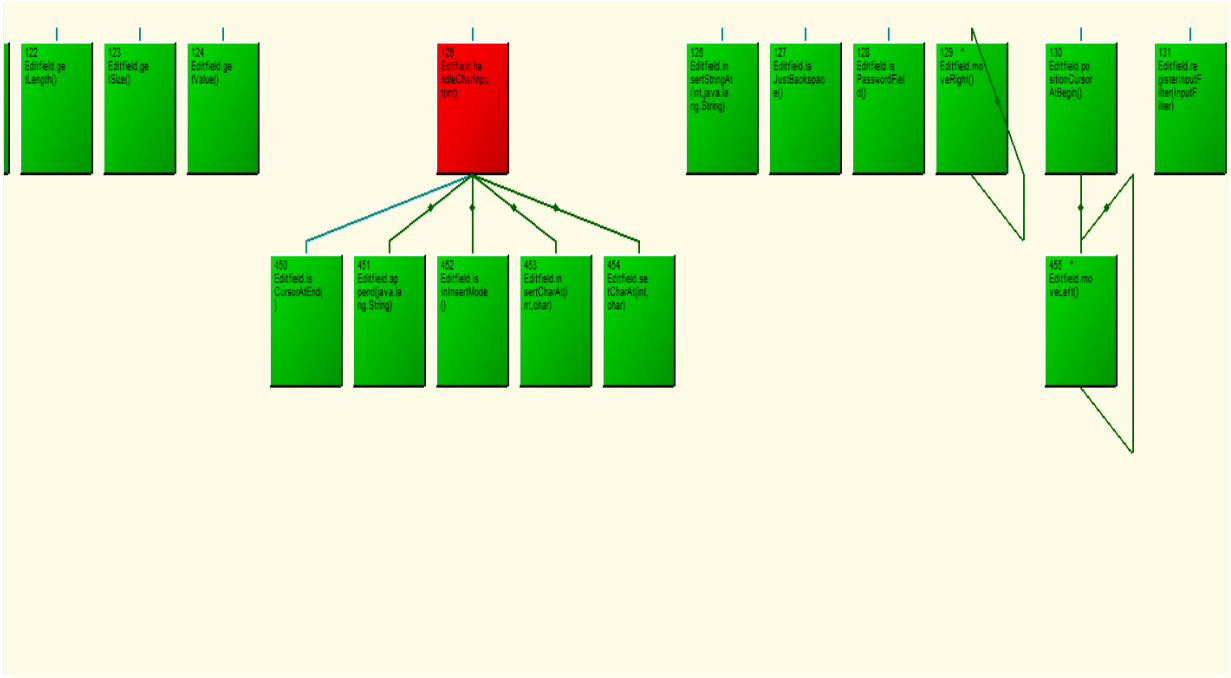


Fig no.13(a) Battle map for Java Telnet Release 2.0

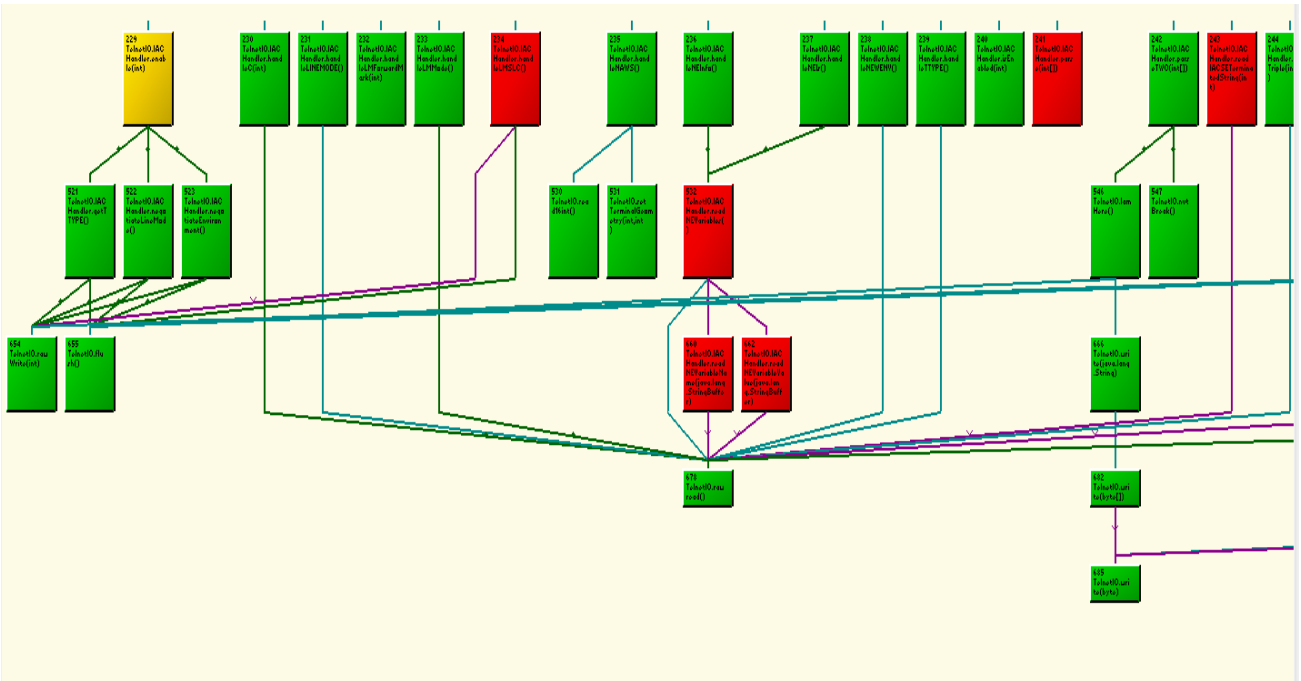


Fig 13(b). Battle map for Java Telnet Release 2.0

There are 803 modules in which 19 are red,3 yellow and the other 781 modules are green.

4.2 McCabe Complexity Metrics Report:

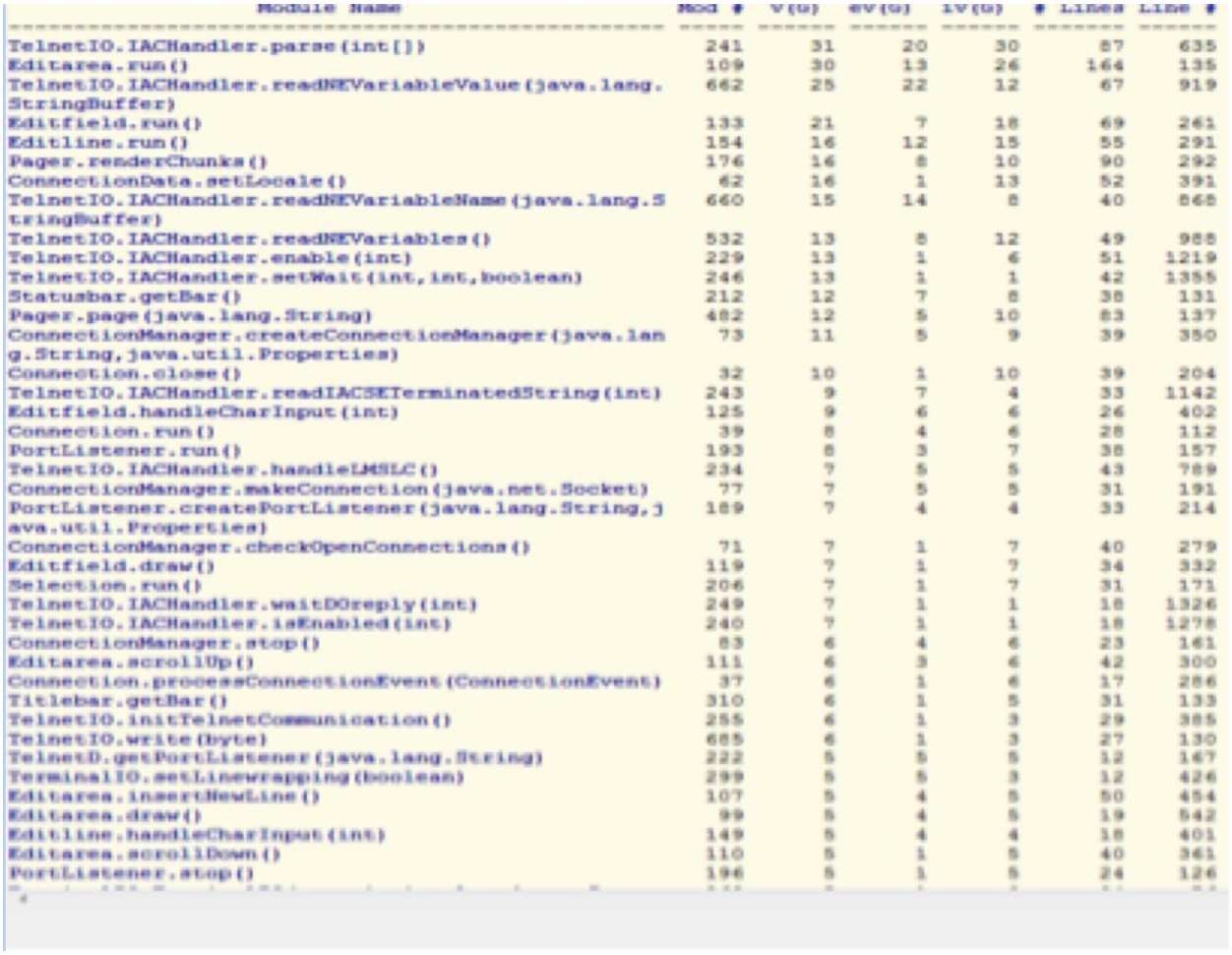
Here is the sample McCabe complexity metrics report. Highest complexity metrics are:

V (G) = 31 TelnetIO.IACHandler.parse (int [])

Ev (G) = 20 TelnetIO.IACHandler.parse (int [])

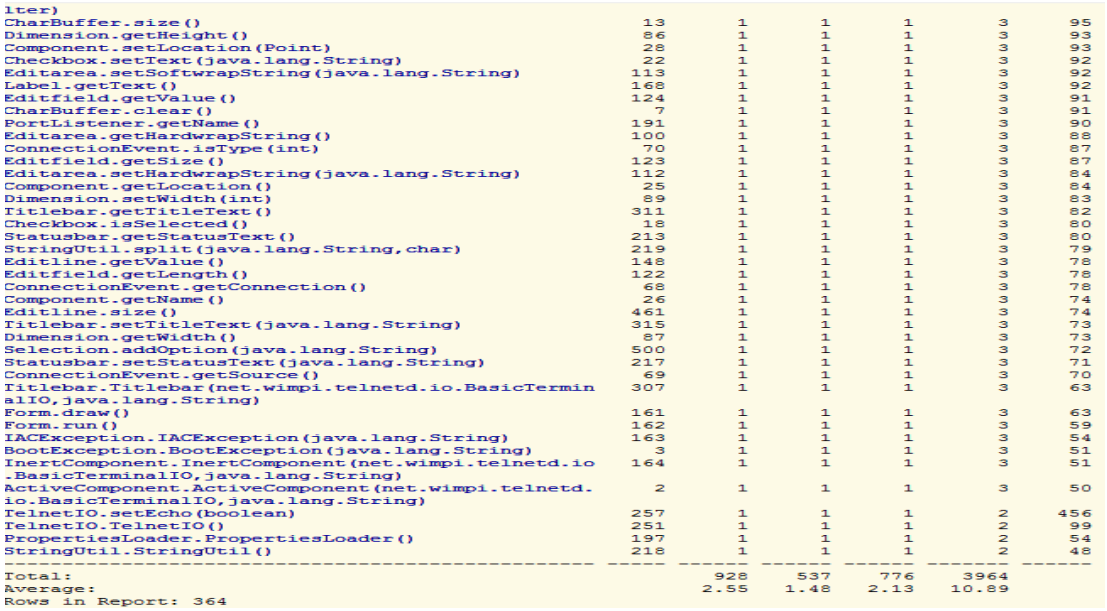
iV (G) = 30 TelnetIO.IACHandler.parse (int[])

When compared to release 1.0, there are only 15 modules where Cyclomatic Complexity (v (G)) is greater than 10. There are 5 modules where Essential Complexity (ev (G)) is greater than 10. There are 10 modules where Module Design Complexity (iv (G)) is greater than 10.



Module Name	Mod #	v(G)	ev(G)	iv(G)	# Lines	Line #
TelnetIO.IACHandler.parse(int[])	241	31	20	30	87	635
Editarea.run()	109	30	13	26	164	135
TelnetIO.IACHandler.readNEVariableValue(java.lang.StringBuffer)	662	25	22	12	67	919
Editfield.run()	133	21	7	18	69	261
Editline.run()	154	16	12	15	55	291
Pager.renderChunks()	176	16	8	10	90	292
ConnectionData.setLocale()	62	16	1	13	52	391
TelnetIO.IACHandler.readNEVariableName(java.lang.StringBuffer)	660	15	14	8	40	868
TelnetIO.IACHandler.readNEVariables()	532	13	8	12	49	988
TelnetIO.IACHandler.enable(int)	229	13	1	6	51	1219
TelnetIO.IACHandler.setWait(int,int,boolean)	246	13	1	1	42	1355
Statusbar.getBar()	212	12	7	8	38	131
Pager.page(java.lang.String)	482	12	5	10	83	137
ConnectionManager.createConnectionManager(java.lang.String,java.util.Properties)	73	11	5	9	39	350
Connection.close()	32	10	1	10	39	204
TelnetIO.IACHandler.readIACSETerminatedString(int)	243	9	7	4	33	1142
Editfield.handleCharInput(int)	125	9	6	6	26	402
Connection.run()	39	8	4	6	28	112
PortListener.run()	193	8	3	7	38	157
TelnetIO.IACHandler.handleLMSLC()	234	7	5	5	43	789
ConnectionManager.makeConnection(java.net.Socket)	77	7	5	5	31	191
PortListener.createPortListener(java.lang.String,java.util.Properties)	189	7	4	4	33	214
ConnectionManager.checkOpenConnections()	71	7	1	7	40	279
Editfield.draw()	119	7	1	7	34	332
Selection.run()	206	7	1	7	31	171
TelnetIO.IACHandler.waitDOreply(int)	249	7	1	1	18	1326
TelnetIO.IACHandler.isEnabled(int)	240	7	1	1	18	1278
ConnectionManager.stop()	83	6	4	6	23	161
Editarea.scrollUp()	111	6	3	6	42	300
Connection.processConnectionEvent(ConnectionEvent)	37	6	1	6	17	286
Titlebar.getBar()	310	6	1	5	31	133
TelnetIO.initTelnetCommunication()	255	6	1	3	29	385
TelnetIO.write(byte)	685	6	1	3	27	130
TelnetD.getPortListener(java.lang.String)	222	5	5	5	12	167
TerminalIO.setLinewrapping(boolean)	299	5	5	3	12	426
Editarea.insertNewLine()	107	5	4	5	50	454
Editarea.draw()	99	5	4	5	19	542
Editline.handleCharInput(int)	149	5	4	4	18	401
Editarea.scrollDown()	110	5	1	5	40	361
PortListener.stop()	196	5	1	5	24	126

Fig no.14(a)Fragment 1 for module metrics java telnet Daemon version 2.0



lter)					
CharBuffer.size()	13	1	1	1	3	95
Dimension.getHeight()	86	1	1	1	3	93
Component.setLocation(Point)	28	1	1	1	3	93
Checkbox.setText(java.lang.String)	22	1	1	1	3	92
Editarea.setSoftwrapString(java.lang.String)	113	1	1	1	3	92
Label.getText()	168	1	1	1	3	92
Editfield.getValue()	124	1	1	1	3	91
CharBuffer.clear()	7	1	1	1	3	91
PortListener.getName()	191	1	1	1	3	90
Editarea.getHardwrapString()	100	1	1	1	3	88
ConnectionEvent.isType(int)	70	1	1	1	3	87
Editfield.getSize()	123	1	1	1	3	87
Editarea.setHardwrapString(java.lang.String)	112	1	1	1	3	84
Component.getLocation()	25	1	1	1	3	84
Dimension.setWidth(int)	89	1	1	1	3	83
Titlebar.getTitleText()	311	1	1	1	3	82
Checkbox.isSelected()	18	1	1	1	3	80
Statusbar.getStatusText()	213	1	1	1	3	80
StringUtil.split(java.lang.String,char)	219	1	1	1	3	79
Editline.getValue()	148	1	1	1	3	78
Editfield.getLength()	122	1	1	1	3	78
ConnectionEvent.getConnection()	68	1	1	1	3	78
Component.getName()	26	1	1	1	3	74
Editline.size()	461	1	1	1	3	74
Titlebar.setTitleText(java.lang.String)	315	1	1	1	3	73
Dimension.getWidth()	87	1	1	1	3	73
Selection.addOption(java.lang.String)	500	1	1	1	3	72
Statusbar.setStatusText(java.lang.String)	217	1	1	1	3	71
ConnectionEvent.getSource()	69	1	1	1	3	70
Titlebar.Titlebar(net.wimpi.telnetd.io.BasicTermin	307	1	1	1	3	63
alIO,java.lang.String)						
Form.draw()	161	1	1	1	3	63
Form.run()	162	1	1	1	3	59
IACException.IACException(java.lang.String)	163	1	1	1	3	54
BootException.BootException(java.lang.String)	3	1	1	1	3	51
InertComponent.InertComponent(net.wimpi.telnetd.io	164	1	1	1	3	51
.BasicTerminalIO,java.lang.String)						
ActiveComponent.ActiveComponent(net.wimpi.telnetd.	2	1	1	1	3	50
io.BasicTerminalIO,java.lang.String)						
TelnetIO.setEcho(boolean)	257	1	1	1	2	456
TelnetIO.TelnetIO()	251	1	1	1	2	99
PropertiesLoader.PropertiesLoader()	197	1	1	1	2	54
StringUtil.StringUtil()	218	1	1	1	2	48
Total:		928	537	776	3964	
Average:		2.55	1.48	2.13	10.89	
Rows in Report: 364						

Figure 14(b): Fragment 2 for for module metrics java telnet Daemon version 2.0

Here the average Cyclomatic Complexity is 2.55, Essential Complexity is 1.48 and Module Design Complexity is 10.89.

4.3 SCATTER DIAGRAM:-

Here is the scatter diagram of JAVA TELNET DAEMON 2.0 version:

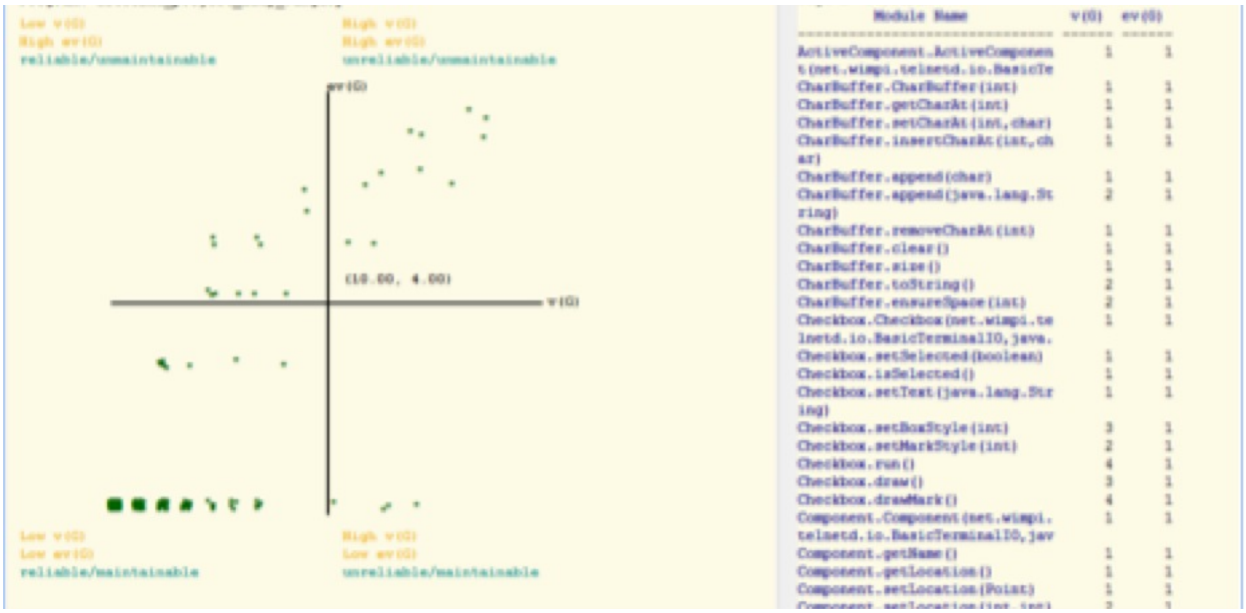


Fig 15(a). Scatter plot for Java Telnet Release 2.0

Fig no.15(b) Scatter plot for Java telnet Release 2.0

4.4 Flow graphs of Selected Modules:

GRAPH LISTING:

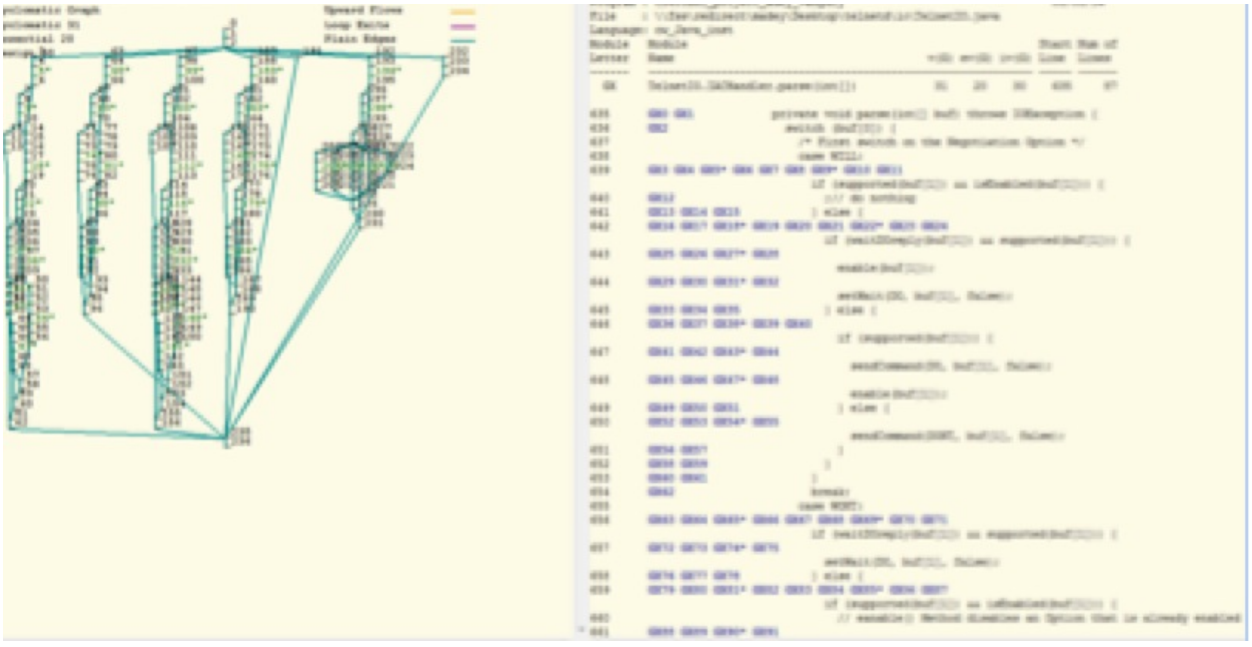


Fig no. 16 Graph listing for Java telnet Release 2.0

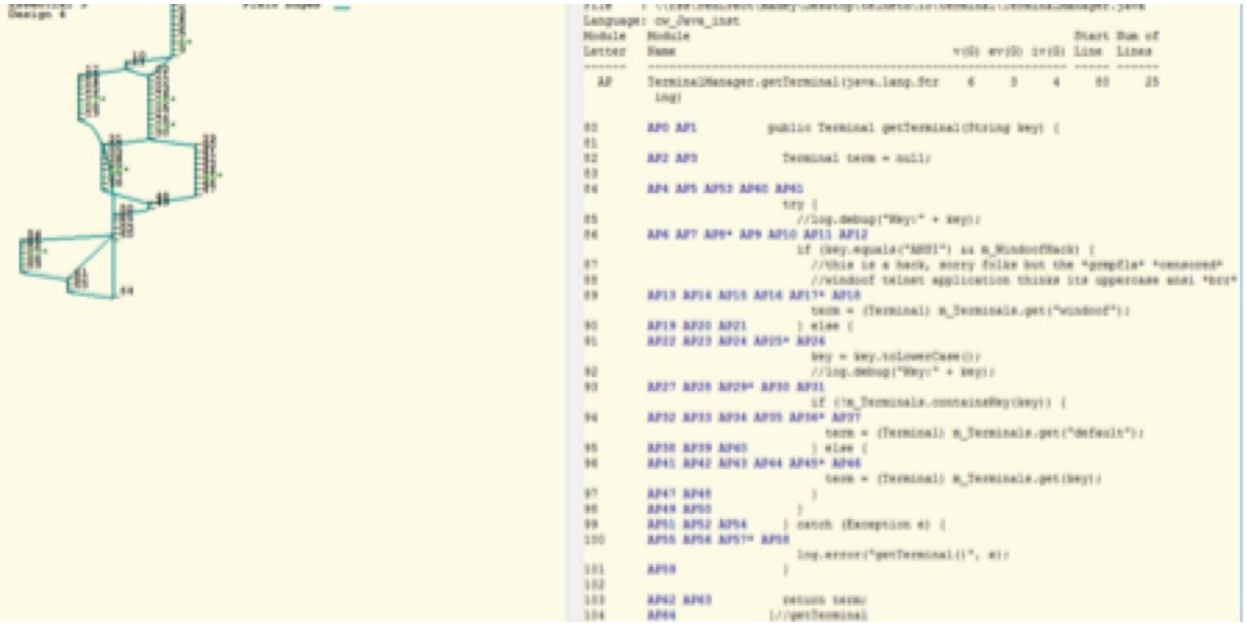


Fig no.17 Graph listing for Java Telnet Release 2.0

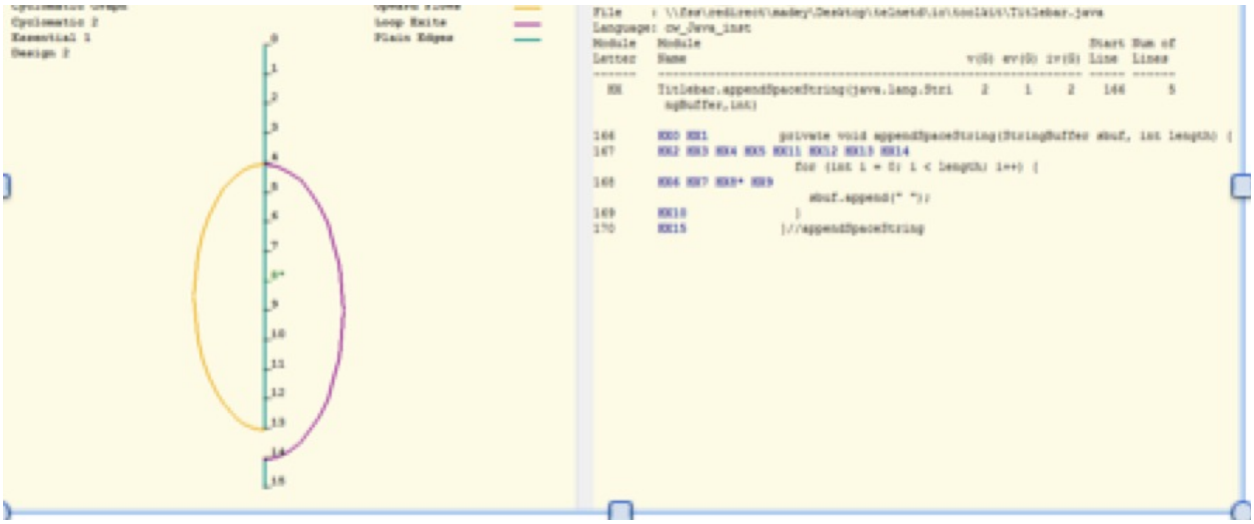


Fig no.18 Graph listing for Java Telnet Release 2.0

4.5 Independent path for Unit Tests:

Test Path 1 of 31:

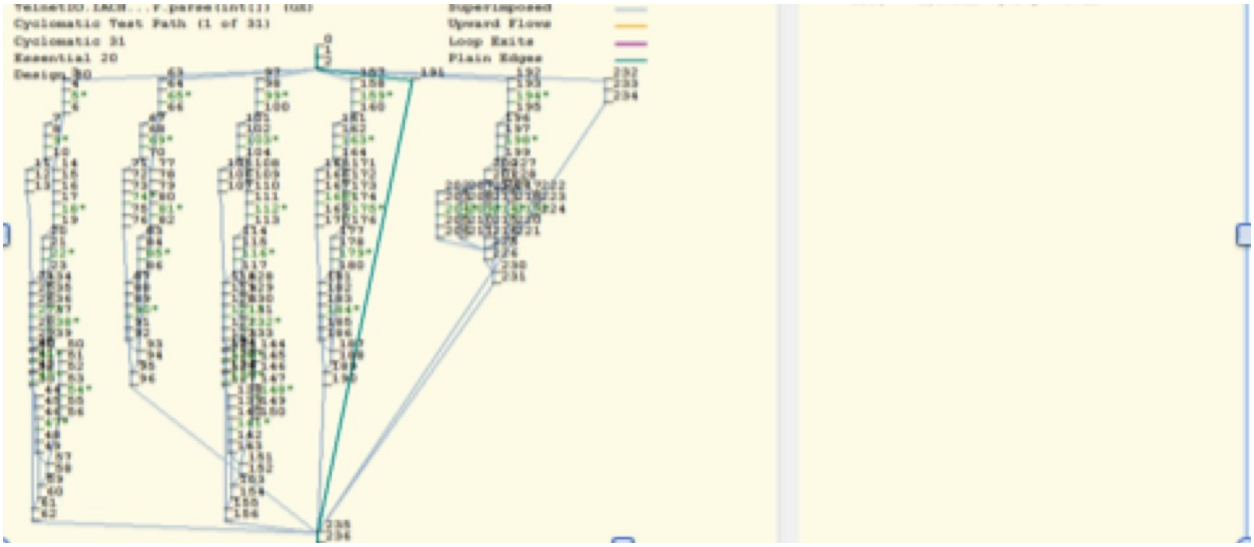


Fig no.19 path 1 of Java Telnet Release 2.0

Test Path 9 of 31:

Fig no.20 Path 9 for Java Telnet Release 2.0

Test path 22 OF 31:

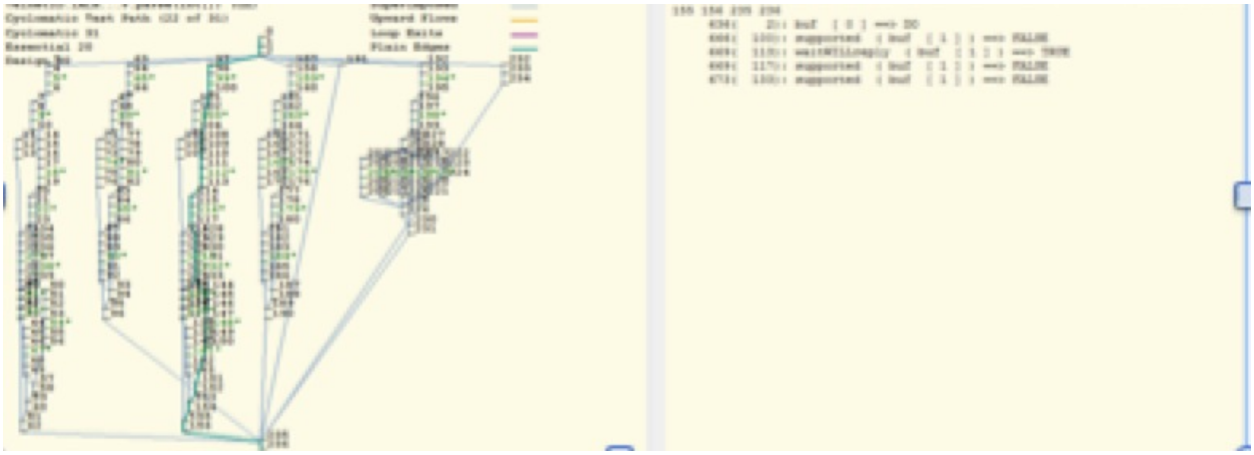


Fig no.(21) path 22 for Java Telnet Release 2.0

Test path 15 of 31:

Fig no.(22) Path 15 for Java Telnet Release 2.0

Test Path 31 of 31:

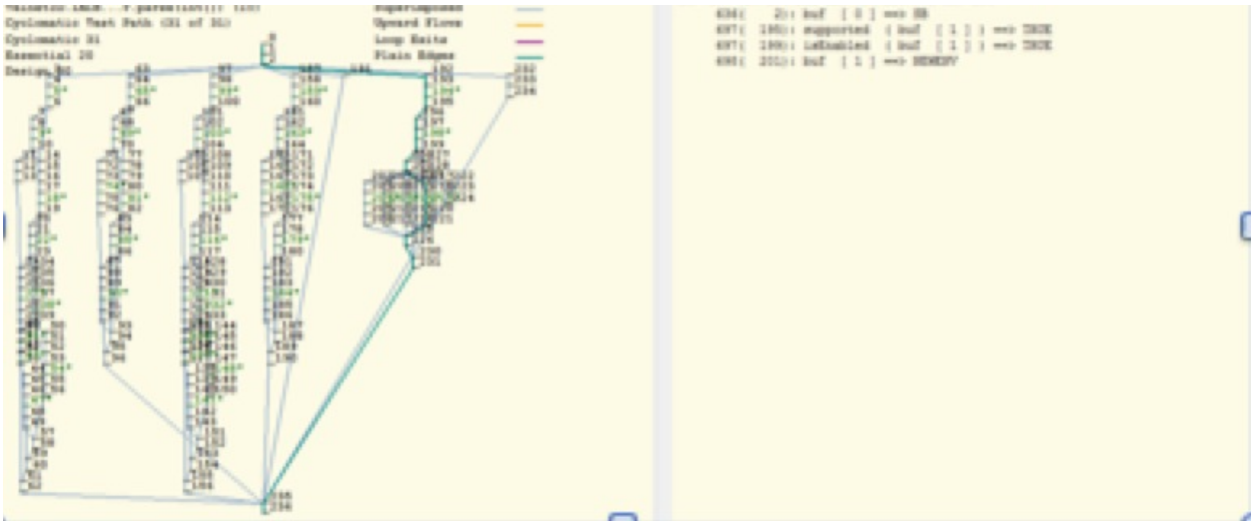


Fig no.(23) Path 31 for java Telnet Release 2.0

4.6 Halstead's Metric Report:

Fig no.23(a) Halsted metrics for Java Telnet Release 2.0

ConnectionData.getLoginShell()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
ConnectionData.isLineMode()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
Label.getText()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
Editline.isHardwrapped()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
Point.getColumn()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
Editline.isInInsertMode()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
ConnectionData.getNegotiatedTerminalType()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
ConnectionEvent.getSource()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
ConnectionEvent.getConnection()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
Editline.getLastRead()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
Editline.getCursorPosition()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
Point.getRow()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
ConnectionManager.getConnectionFilter()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
Connection.getTerminalIO()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
Connection.getConnectionData()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
Editfield.isPasswordField()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
Selection.getSelected()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
Editfield.isInInsertMode()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
Statusbar.getStatusText()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
PortListener.getConnectionManager()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
PortListener.isAvailable()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
Editfield.isJustBackspace()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
Editfield.getCursorPosition()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
Editarea.getSoftwrapString()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
PortListener.getName()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
Component.getLocation()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
Titlebar.getTitleText()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
Editarea.getHardwrapString()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
Checkbox.isSelected()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
TerminalIO.getTerminal()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
TerminalIO.isLineWrapping()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
TerminalIO.isAutoflushing()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
Dimension.getHeight()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
TerminalIO.isSignalling()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
Dimension.getWidth()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
TelnetD.getVersion()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
ConnectionData.getLastActivity()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
Component.getDimension()	4	8.00	0.67	1.50	5.33	12.00	0.00	0.67
PropertiesLoader.PropertiesLoader()	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Form.run()	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Form.draw()	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00
TelnetIO.TelnetIO()	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00
TelnetIO.setEcho(boolean)	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00
StringUtil.StringUtil()	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Total:	15423	75326.99	116.71	2338.88	7265.81	1525748.04	24.90	84764.00
Average:	42.37	206.94	0.32	6.43	19.96	4191.62	0.07	232.87

Rows in Report: 364

Fig no.23(b) Halsted metrics for Java Telnet Release 2.0

Program length (N)

The program length (N) is the sum of the total number of operators and operands in the program:

N = N1 + N2 [7]. Average program length (N) for this version is 42.37

The module with the largest N is Editarea.run () with a value of 643.

Program volume (V)

The program volume (V) is the information contents of the program, measured in mathematical bits. It is calculated as the program length times the 2-base logarithm of the vocabulary size (n) :

V = N * log2(n) [7]. Average program volume (V) for this version is 206.94

Editarea.run () also has the highest V value of 4087.91.

Program level (L)

The program level (L) is the inverse of the error proneness of the program. I.e. a low level program is more prone to errors than a high level program.

L = 1 / D [7]. Here the average program level (L) is 0.32.

For the highest L of 0.67, there are any modules with this value.

Difficulty level (D)

The difficulty level or error proneness (D) of the program is proportional to the number of unique operators. D is also proportional to the ration between the total number of operands and the number of

Unique operands (i.e. if the same operands are used many times in the program, it is more prone to errors) [7].

D = (n1 / 2) * (N2 / n2) . Here the average difficulty level (D) is 6.43.

The module Editarea.run () with the highest D is with a value of 53.94

Effort to implement (E)

The effort to implement (E) or understand a program is proportional to the volume and to the difficulty level of the program [7].

E = V * D. Here the average effort to implement (E) is 4191.72

Editarea.run () has the highest E value of 220511.08

Number of delivered bugs (B)

The number of delivered bugs (B) correlates with the overall complexity of the software. Halstead gives the following formula for B:

B = (E ** (2/3)) / 3000 ** stands for "to the exponent"

Halstead's delivered bugs (B) is an estimate for the number of errors in the implementation. Delivered bugs in a file should be less than 2[9]. In this case average number of delivered bugs (B) is 0.07.

Also, Editarea.run () the highest value of 1.36 for B.

Time to implement (T)

The time to implement or understand a program (T) is proportional to the effort. Empirical experiments can be used for calibrating this quantity. Halstead has found that dividing the effort by 18 give an approximation for the time in seconds [7].

T = E / 18. Here the average Time to implement (T) is 232.87

Lastly, Editarea.run () again has the highest T value of 12250.62

4.7 Object oriented metrics:

```
                                                    Avg    Max
POLYMORPHISM:
-------------
Response for Class (RFC)                          : 17.97  90
Weighted Methods Per Class (WMC)                  : 11.87  48
Percentage of Unoverloaded Calls (PCT_NOTOV)      : 100

ENCAPSULATION:
--------------
Percentage of Public/Protected Data (PUB_DATA)    : 20.03  100
Accesses to Public Data (PUB_ACCESS)              : 0.35   11
Lack of Cohesion of Methods (LOCM)                : 56.16  100

INHERITANCE:
------------
Depth In Class Hierarchy (DEPTH)                  : 1.84   3
Number of Children (NOC)                          : 0.39   5
Number of Parents (OOFANIN)                       : 0.58   1
Number of Class Hierarchy Roots (ROOTCNT)         : 37

QUALITY:
--------
Number of Classes Dependent On Children (DEP_CT)  : 0
Class Maximum Essential Complexity (MAX_EVG)      : 3.81   22
Class Maximum Cyclomatic Complexity (MAX_VG)      : 7.19   31
```

Fig no.24 Program metrics for Java Telnet Release 2.0

Class metrics:

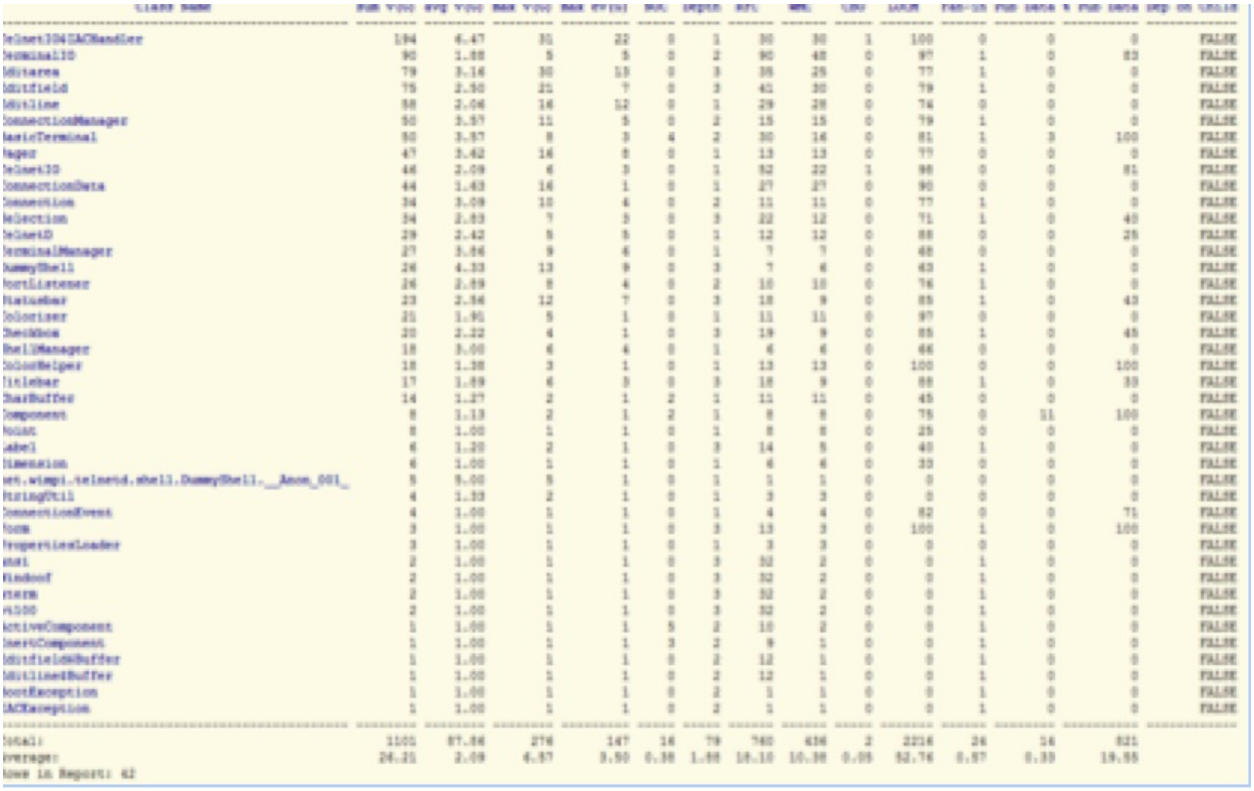


Fig no. 25 Class metrics for Java Telnet Release 2.0

The OO metrics McCabe provides are the following broken down by aspect of OO design [7]:

Polymorphism RFC – Response for Class - A count of methods implemented within a class plus the number of methods accessible to an object of this class due to inheritance. In general, lower values indicate greater polymorphism. Here the average response for class is 17.97

WMC – Weighted Methods per Class - A count of methods implemented within a class (rather than all methods accessible within the class hierarchy). In general, lower values indicate greater polymorphism. Average weighted methods per class is 11.87

PCT_NOTOV – Percent of Unoverloaded Calls - The percentage of calls that are not made to overloaded modules. This metric is generated for the entire program, not individual classes. In general, lower values indicate greater polymorphism. Average percent of unoverloaded calls is 100.

Encapsulation PUB_DATA – Percent Public Data - The percentage of data that is public and protected data in a class. In general, lower values indicate greater encapsulation. Average percent public data is 20.03.

PUB_ACCESS – Access to Public Data - The amount of times that a class's public and protected data is accessed. In general, lower values indicate greater encapsulation Average access to public data is 0.35. **LOCM – Lack of Cohesion of Methods** - For each data field in a class, the percentage of the methods in the class using that data field; the percentages are averaged then subtracted from 100%. The LOCM metric indicates low or high percentage of cohesion. If the percentage is low, the class is cohesive. If it is high, it may indicate that the class could be split into separate classes that will individually have greater cohesion. Average lack of cohesion of methods is 56.16.

Inheritance DEPTH – Depth - The level for a class. For example, if a parent class has one child, the depth for the child is 2. Depth indicates at what level a class is located within its class hierarchy. In general, inheritance increases when depth increases. Average depth is 1.84.

NOC – Number of Children - The number of classes derived from a specified class. Average number of children is 0.39.

OOFANIN – Number of Parents (fan-in) - The number of classes from which a class is derived.Average number of parents are 0.58.

ROOTCNT – Number of Class Hierarchy Roots - The sum of the class hierarchy roots within a program (that is, the number of distinct class hierarchies employed by a program). Average number of class hierarchy root is 37.

CBO – Coupling between Objects - The number of distinct non-inheritance-related classes on which a class depends. If a class that is heavily dependent on many classes outside of its hierarchy is introduced into a library, all the classes upon which it depends need to be introduced as well. This may be acceptable, especially if the classes which it references are already part of a class library and are even more fundamental than the specified class. Average Coupling between objects is 0.05.

Quality DEP_CT – Number of Classes Dependent on Children – The number of classes dependent on descendants. A higher value would indicate poor quality in the OO design. Average number of classes dependent on children is 0.

MAX_EVG – Class Maximum Essential Complexity - The maximum essential complexity of a module within all the classes in a program/section. Average class maximum essential complexity is 3.81.

MAX_VG – Class Maximum Cyclomatic Complexity - The maximum cyclomatic complexity of a module within all the classes in a program. Average class maximum cyclomatic complexity 7.19

5.0 Comparisons of Releases:

Here are the comparisons:

Module metrics:

	Java Telnet Release 1.0	**Java Telnet Release 2.0**
Modules	527	803
Total v(G)	735	928
Average v(G)	2.39	2.55
Total ev(G)	404	537
Average ev(G)	1.32	1.48
Total iv(G)	604	776
Average iv(G)	1.97	2.13

Halsted Metrics:

Halsted metrics	Release 1.0	Release 2.0	Difference
N (number of operators and operands in the function)	13961	15423	1462
V (minimum number of bits required to code the function)	68730.84	75326.99	6596.15
L (Program Level)	94.00	116.71	22.71
D (Difficulty Level)	2122.97	2338.88	215.91
I (Complexity of the algorithm)	6283.43	7265.81	982.38
E (Effort to implement the function)	1559393.35	1525748.04	33645.31
B (Predicted no of bugs)	22.74	24.90	2.16
T (Time to implement the function)	56633.09	84764.00	28130.91

Class Metrics:

Metrics	Release 1.0	Release 2.0	Difference
RFC (Response for class)	608	557	51
WMC (Weighted Methods per class)	309	368	59
PUB_DATA	13	14	1

(Percentage of Public/Protected Data)			
PUB_ACCESS(Access to Public Data)	737	821	84
LOCM (Lack of Cohesion of Methods)	1517	2216	699
DEPTH (Depth in Class Hierarchy)	58	79	21
NOC (Number of Children)	16	16	0
CBO(Coupling between Objects)	2	2	0

6.0 Recommendations:

In version 2.0 more number of modules were added when compared with 1.0 and it has the highest values for the Halsted metrics than 1.0 version. It will complicate the program and take more time to implement in order to meet the program efficiency and reduce the Halsted metric for 2.0 version.

7.0 Conclusion:

After analyzing all the metrics it is concludes that the size of the code has increased considerably and the release 1 code has been modified. This code is more error prone and more likely to have more bugs . Object oriented metrics in certain instances indicate that the code stays the same path of the principle of

inheritance.

	Java Telnet Release 1.0	Java Telnet Release 2.0
Total No. of Modules	527	803
Total v(G)	735	928
Average v(G)	2.39	2.55
Total ev(G)	404	537
Average ev(G)	1.32	1.48
Total iv(G)	604	776
Average iv(G)	1.97	2.13
Unreliable/unmaintainable according to scatter plot	6	11
N in Halstead (Number of operators and operands in the function)	843	643
Highest Weighted Methods per Class (WMC)	11.04	11.87

Therefore, Using McCabe IQ tool, codes complexity and quality based on the analysis of Battle Maps, Module Metrics, Halstead Metrics, and Object-Oriented Metrics are studied and compared. After comparing both the releases there are 276 modules increased in version 2.0 than version 1.0.Therefore due to increase in the modules there may be increase in complexity. Halstead metrics in version2.0 have high values when compared to version 1.0 and objected oriented metrics have slightly similar value .By taking all this into consideration we can conclude that release two has more complexity and it will be more prone to errors.

Glossary:

CBO = Coupling Between Objects

DEP_CT = Number of Classes Dependent on Children

HTML = Hyper Text Markup Language

LOCM = Lack of Cohesion of Methods

MAX_EVG = Class Maximum Essential Complexity

MAX_VG = Class Maximum Cyclomatic Complexity

NOC = Number Of Children

OO = Object Oriented

OOFANIN = Number Of Parents (Fan-In)

PCT_NOTOV = Percent of Unoverloaded Calls

PUB_DATA = Percent Public Data

PUB_ACCESS = Access to Public Data

RFC = Response For Class

ROOTCNT = Number of Class Hierarchy Roots

WMC = Weighted Methods per Class

XML = extensible Markup Language

References and links:

[1] Abreu, F. B. e., "The MOOD Metrics Set," presented at ECOOP '95 Workshop on Metrics, 1995.

[2] Abreu, F. B. e. and Melo, W., "Evaluating the Impact of OO Design on Software Quality," presented at Third International Software Metrics Symposium, Berlin, 1996.

[3] Boehm, B. W., "Improving Software Productivity," *IEEE Computer*, pp. 43-57, September 1987.

[4] Briand, L., Emam, K. E., and Morasca, S., "Theoretical and Empirical Validation of Software Metrics," 1995.

[5] http://sourceforge.net/projects/telnetd/?source=directory.

[6] Fowler, M., Beck, K., Brant, J., Opdyke, W., and Roberts, d., *Refactoring: Improving the Design of Existing Code*. Reading, Massachusetts: Addison Wesley, 1999.

[7] Glasberg, D., Emam, K. E., Melo, W., and Madhavji, N., "Validating Object-Oriented Design Metrics on a Commercial Java Application," National Research Council 44146, September 2000.

APPENDIX A- SORCE CODE

Source code for version 1.0 and 2.0 is attached as zip from which it can be extracted for inspection

```
import java.io.*;
import java.util.*;
import java.awt.Dimension;

import javax.media.*;
import javax.media.control.*;
import javax.media.protocol.*;
import javax.media.protocol.DataSource;
import javax.media.datasink.*;
import javax.media.format.VideoFormat;


/**
 * This program takes a list of JPEG image files and convert them into
 * a QuickTime movie.
```

```
 */
public class JpegImagesToMovie implements ControllerListener,
DataSinkListener {


        public boolean doItPath(int width, int height, int frameRate, Vector
inFiles, String outputURL) {
         // Check for output file extension.
         if (!outputURL.endsWith(".mov") && !outputURL.endsWith(".MOV")) {
             //System.err.println("The output file extension should end with a
.mov extension");
             prUsage();
         }

         // Generate the output media locators.
         MediaLocator oml;

         if ((oml = createMediaLocator("file:" + outputURL)) == null) {
             //System.err.println("Cannot build media locator from: " +
outputURL);
             System.exit(0);
         }

         boolean success = doIt(width, height, frameRate, inFiles, oml);

         System.gc();
         return success;
}
    public boolean doIt(int width, int height, int frameRate, Vector inFiles,
MediaLocator outML) {
         ImageDataSource ids = new ImageDataSource(width, height, frameRate,
inFiles);

         Processor p;

         try {
             System.err.println("- create processor for the image datasource
...");
             p = Manager.createProcessor(ids);
         } catch (Exception e) {
             System.err.println("Yikes!  Cannot create a processor from the
data source.");
             return false;
         }

         p.addControllerListener(this);

         // Put the Processor into configured state so we can set
         // some processing options on the processor.
         p.configure();
         if (!waitForState(p, p.Configured)) {
             System.err.println("Failed to configure the processor.");
             return false;
         }

         // Set the output content descriptor to QuickTime.
```

```
        p.setContentDescriptor(new
ContentDescriptor(FileTypeDescriptor.QUICKTIME));

        // Query for the processor for supported formats.
        // Then set it on the processor.
        TrackControl tcs[] = p.getTrackControls();
        Format f[] = tcs[0].getSupportedFormats();
        if (f == null || f.length <= 0) {
            System.err.println("The mux does not support the input format: " +
tcs[0].getFormat());
            return false;
        }

        tcs[0].setFormat(f[0]);

        System.err.println("Setting the track format to: " + f[0]);

        // We are done with programming the processor.  Let's just
        // realize it.
        p.realize();
        if (!waitForState(p, p.Realized)) {
            System.err.println("Failed to realize the processor.");
            return false;
        }

        // Now, we'll need to create a DataSink.
        DataSink dsink;
        if ((dsink = createDataSink(p, outML)) == null) {
            System.err.println("Failed to create a DataSink for the given
output MediaLocator: " + outML);
            return false;
        }

        dsink.addDataSinkListener(this);
        fileDone = false;

        System.err.println("start processing...");

        // OK, we can now start the actual transcoding.
        try {
            p.start();
            dsink.start();
        } catch (IOException e) {
            System.err.println("IO error during processing");
            return false;
        }

        // Wait for EndOfStream event.
        waitForFileDone();

        // Cleanup.
        try {
            dsink.close();
        } catch (Exception e) {}
        p.removeControllerListener(this);
```

```
        System.err.println("...done processing.");

        return true;
    }


    /**
     * Create the DataSink.
     */
    DataSink createDataSink(Processor p, MediaLocator outML) {

        DataSource ds;

        if ((ds = p.getDataOutput()) == null) {
            System.err.println("Something is really wrong: the processor does
not have an output DataSource");
            return null;
        }

        DataSink dsink;

        try {
            System.err.println("- create DataSink for: " + outML);
            dsink = Manager.createDataSink(ds, outML);
            dsink.open();
        } catch (Exception e) {
            System.err.println("Cannot create the DataSink: " + e);
            return null;
        }

        return dsink;
    }


    Object waitSync = new Object();
    boolean stateTransitionOK = true;

    /**
     * Block until the processor has transitioned to the given state.
     * Return false if the transition failed.
     */
    boolean waitForState(Processor p, int state) {
        synchronized (waitSync) {
            try {
                while (p.getState() < state && stateTransitionOK)
                    waitSync.wait();
            } catch (Exception e) {}
        }
        return stateTransitionOK;
    }


    /**
     * Controller Listener.
     */
    public void controllerUpdate(ControllerEvent evt) {
```

```
    if (evt instanceof ConfigureCompleteEvent ||
        evt instanceof RealizeCompleteEvent ||
        evt instanceof PrefetchCompleteEvent) {
        synchronized (waitSync) {
            stateTransitionOK = true;
            waitSync.notifyAll();
        }
    } else if (evt instanceof ResourceUnavailableEvent) {
        synchronized (waitSync) {
            stateTransitionOK = false;
            waitSync.notifyAll();
        }
    } else if (evt instanceof EndOfMediaEvent) {
        evt.getSourceController().stop();
        evt.getSourceController().close();
    }
}


Object waitFileSync = new Object();
boolean fileDone = false;
boolean fileSuccess = true;

/**
 * Block until file writing is done.
 */
boolean waitForFileDone() {
    synchronized (waitFileSync) {
        try {
            while (!fileDone)
                waitFileSync.wait();
        } catch (Exception e) {}
    }
    return fileSuccess;
}


/**
 * Event handler for the file writer.
 */
public void dataSinkUpdate(DataSinkEvent evt) {

    if (evt instanceof EndOfStreamEvent) {
        synchronized (waitFileSync) {
            fileDone = true;
            waitFileSync.notifyAll();
        }
    } else if (evt instanceof DataSinkErrorEvent) {
        synchronized (waitFileSync) {
            fileDone = true;
            fileSuccess = false;
            waitFileSync.notifyAll();
        }
    }
}
```

```
    public static void main(String args[]) {

        if (args.length == 0)
            prUsage();

        // Parse the arguments.
        int i = 0;
        int width = -1, height = -1, frameRate = 1;
        Vector inputFiles = new Vector();
        String outputURL = null;

        while (i < args.length) {

            if (args[i].equals("-w")) {
                i++;
                if (i >= args.length)
                    prUsage();
                width = new Integer(args[i]).intValue();
            } else if (args[i].equals("-h")) {
                i++;
                if (i >= args.length)
                    prUsage();
                height = new Integer(args[i]).intValue();
            } else if (args[i].equals("-f")) {
                i++;
                if (i >= args.length)
                    prUsage();
                frameRate = new Integer(args[i]).intValue();
            } else if (args[i].equals("-o")) {
                i++;
                if (i >= args.length)
                    prUsage();
                outputURL = args[i];
            } else {
                inputFiles.addElement(args[i]);
            }
            i++;
        }

        if (outputURL == null || inputFiles.size() == 0)
            prUsage();

        // Check for output file extension.
        if (!outputURL.endsWith(".mov") && !outputURL.endsWith(".MOV")) {
            System.err.println("The output file extension should end with a
.mov extension");
            prUsage();
        }

        if (width < 0 || height < 0) {
            System.err.println("Please specify the correct image size.");
            prUsage();
        }
```

```
        // Check the frame rate.
        if (frameRate < 1)
            frameRate = 1;

        // Generate the output media locators.
        MediaLocator oml;

        if ((oml = createMediaLocator(outputURL)) == null) {
            System.err.println("Cannot build media locator from: " +
outputURL);
            System.exit(0);
        }

        JpegImagesToMovie imageToMovie = new JpegImagesToMovie();
        imageToMovie.doIt(width, height, frameRate, inputFiles, oml);

        System.exit(0);
    }

    static void prUsage() {
        System.err.println("Usage: java JpegImagesToMovie -w <width> -h
<height> -f <frame rate> -o <output URL> <input JPEG file 1> <input JPEG file
2> ...");
        System.exit(-1);
    }

    /**
     * Create a media locator from the given string.
     */
    static MediaLocator createMediaLocator(String url) {

        MediaLocator ml;

        if (url.indexOf(":") > 0 && (ml = new MediaLocator(url)) != null)
            return ml;

        if (url.startsWith(File.separator)) {
            if ((ml = new MediaLocator("file:" + url)) != null)
                return ml;
        } else {
            String file = "file:" + System.getProperty("user.dir") +
File.separator + url;
            if ((ml = new MediaLocator(file)) != null)
                return ml;
        }

        return null;
    }


    ////////////////////////////////////////////
    //
    // Inner classes.
    ////////////////////////////////////////////
```

```
    /**
     * A DataSource to read from a list of JPEG image files and
     * turn that into a stream of JMF buffers.
     * The DataSource is not seekable or positionable.
     */
    class ImageDataSource extends PullBufferDataSource {

        ImageSourceStream streams[];

        ImageDataSource(int width, int height, int frameRate, Vector images) {
            streams = new ImageSourceStream[1];
            streams[0] = new ImageSourceStream(width, height, frameRate,
images);
        }

        public void setLocator(MediaLocator source) {
        }

        public MediaLocator getLocator() {
            return null;
        }

        /**
         * Content type is of RAW since we are sending buffers of video
         * frames without a container format.
         */
        public String getContentType() {
            return ContentDescriptor.RAW;
        }

        public void connect() {
        }

        public void disconnect() {
        }

        public void start() {
        }

        public void stop() {
        }

        /**
         * Return the ImageSourceStreams.
         */
        public PullBufferStream[] getStreams() {
            return streams;
        }

        /**
         * We could have derived the duration from the number of
         * frames and frame rate.  But for the purpose of this program,
         * it's not necessary.
         */
        public Time getDuration() {
            return DURATION_UNKNOWN;
```

```
        }

        public Object[] getControls() {
            return new Object[0];
        }

        public Object getControl(String type) {
            return null;
        }
    }


    /**
     * The source stream to go along with ImageDataSource.
     */
    class ImageSourceStream implements PullBufferStream {

        Vector images;
        int width, height;
        VideoFormat format;

        int nextImage = 0;     // index of the next image to be read.
        boolean ended = false;

        public ImageSourceStream(int width, int height, int frameRate, Vector
images) {
            this.width = width;
            this.height = height;
            this.images = images;

            format = new VideoFormat(VideoFormat.JPEG,
                                     new Dimension(width, height),
                                     Format.NOT_SPECIFIED,
                                     Format.byteArray,
                                     (float)frameRate);
        }

        /**
         * We should never need to block assuming data are read from files.
         */
        public boolean willReadBlock() {
            return false;
        }

        /**
         * This is called from the Processor to read a frame worth
         * of video data.
         */
        public void read(Buffer buf) throws IOException {

            // Check if we've finished all the frames.
            if (nextImage >= images.size()) {
                // We are done.  Set EndOfMedia.
                System.err.println("Done reading all images.");
                buf.setEOM(true);
                buf.setOffset(0);
```

```
            buf.setLength(0);
            ended = true;
            return;
        }

        String imageFile = (String)images.elementAt(nextImage);
        nextImage++;

        System.err.println("  - reading image file: " + imageFile);

        // Open a random access file for the next image.
        RandomAccessFile raFile;
        raFile = new RandomAccessFile(imageFile, "r");

        byte data[] = null;

        // Check the input buffer type & size.

        if (buf.getData() instanceof byte[])
            data = (byte[])buf.getData();

        // Check to see the given buffer is big enough for the frame.
        if (data == null || data.length < raFile.length()) {
            data = new byte[(int)raFile.length()];
            buf.setData(data);
        }

        // Read the entire JPEG image from the file.
        raFile.readFully(data, 0, (int)raFile.length());

        System.err.println("    read " + raFile.length() + " bytes.");

        buf.setOffset(0);
        buf.setLength((int)raFile.length());
        buf.setFormat(format);
        buf.setFlags(buf.getFlags() | buf.FLAG_KEY_FRAME);

        // Close the random access file.
        raFile.close();
    }

    /**
     * Return the format of each video frame.  That will be JPEG.
     */
    public Format getFormat() {
        return format;
    }

    public ContentDescriptor getContentDescriptor() {
        return new ContentDescriptor(ContentDescriptor.RAW);
    }

    public long getContentLength() {
        return 0;
    }
```

```
        public boolean endOfStream() {
            return ended;
        }

        public Object[] getControls() {
            return new Object[0];
        }

        public Object getControl(String type) {
            return null;
        }
    }
}
```

Program2:

```
import java.io.*;
import java.util.*;
import java.awt.Dimension;

import javax.media.*;
import javax.media.control.*;
import javax.media.protocol.*;
import javax.media.protocol.DataSource;
import javax.media.datasink.*;
import javax.media.format.VideoFormat;


/**
 * This program takes a list of JPEG image files and converts them into
 * a QuickTime or AVI movie.
 * @author Sun - orig
 * @author Barb Ericson ericson@cc.gatech.edu
 * modified to use List<String> instead of Vector
 * and added methods for Quicktime and AVI
 */
public class JpegImagesToMovie implements ControllerListener,
  DataSinkListener {

  /** number of frames per second */
  private int fRate = 16;

  /**
   * Method to write out a quicktime movie
   * @param width the width of the resulting movie
   * @param height the height of the resulting movie
   * @param frameRate the number of frames per second
   * @param inFiles string full path names of the frames
   * @param outML the Media Locator
   */
  public boolean doItQuicktime(int width, int height,
                               int frameRate,
                               List<String> inFiles,
                               MediaLocator outML) {
```

```
  this.fRate = frameRate;
  return doIt(width,height,frameRate,inFiles,outML,
       FileTypeDescriptor.QUICKTIME);
}

/**
 * Method to write out a quicktime movie
 * @param width the width of the resulting movie
 * @param height the height of the resulting movie
 * @param frameRate the number of frames per second
 * @param inFiles string full path names of the frames
 * @param outputURL the output URL for the movie
 */
public boolean doItQuicktime(int width, int height,
                             int frameRate,
                             List<String> inFiles,
                             String outputURL)
{
  this.fRate = frameRate;
   MediaLocator oml = createMediaLocator(outputURL);
   return doItQuicktime(width,height,frameRate,inFiles,oml);
}

/**
 * Method to write out an AVI movie
 * @param width the width of the resulting movie
 * @param height the height of the resulting movie
 * @param frameRate the number of frames per second
 * @param inFiles string full path names of the frames
 * @param outputURL the output URL for the movie
 */
public boolean doItAVI(int width, int height,
                       int frameRate, List<String> inFiles,
                       String outputURL)
{
  this.fRate = frameRate;
   MediaLocator oml = createMediaLocator(outputURL);
   return doItAVI(width,height,frameRate,inFiles,oml);
}

/**
 * Method to write out an AVI movie
 * @param width the width of the resulting movie
 * @param height the height of the resulting movie
 * @param frameRate the number of frames per second
 * @param inFiles string full path names of the frames
 * @param outML the Media Locator
 */
public boolean doItAVI(int width, int height,
                       int frameRate,
                       List<String> inFiles,
                       MediaLocator outML) {
  this.fRate = frameRate;
  return doItAVI(width,height,frameRate,inFiles,outML,
       FileTypeDescriptor.MSVIDEO);
}
```

```
  /**
   * Method to write out the movie for a given type
   * @param width the width of the resulting movie
   * @param height the height of the resulting movie
   * @param frameRate the number of frames per second
   * @param inFiles string full path names of the frames
   * @param outML the Media Locator
   * @param type the VideoFormat type
   */
  public boolean doIt(int width, int height,
                      int frameRate, List<String> inFiles,
                      MediaLocator outML,
                      String type) {
    this.fRate = frameRate;
    ImageDataSource ids = new ImageDataSource(width, height,
                                              frameRate, inFiles);

    Processor p;

    try {
      //System.err.println("- create processor for the image datasource
...");
      p = Manager.createProcessor(ids);
    } catch (Exception e) {
      System.err.println("Yikes!  Cannot create a processor from the data
source.");
      return false;
    }

    p.addControllerListener(this);

    // Put the Processor into configured state so we can set
    // some processing options on the processor.
    p.configure();
    if (!waitForState(p, p.Configured)) {
      System.err.println("Failed to configure the processor.");
      return false;
    }

    // Set the output content descriptor to QuickTime.
    p.setContentDescriptor(new ContentDescriptor(type));

    // Query for the processor for supported formats.
    // Then set it on the processor.
    TrackControl tcs[] = p.getTrackControls();
    Format f[] = tcs[0].getSupportedFormats();
    if (f == null || f.length <= 0) {
      System.err.println("The mux does not support the input format: " +
tcs[0].getFormat());
      return false;
    }

    tcs[0].setFormat(f[0]);

    //System.err.println("Setting the track format to: " + f[0]);
```

```
    // We are done with programming the processor.  Let's just
    // realize it.
    p.realize();
    if (!waitForState(p, p.Realized)) {
      System.err.println("Failed to realize the processor.");
      return false;
    }

    // Now, we'll need to create a DataSink.
    DataSink dsink;
    if ((dsink = createDataSink(p, outML)) == null) {
      System.err.println("Failed to create a DataSink for the given output
MediaLocator: " + outML);
      return false;
    }

    dsink.addDataSinkListener(this);
    fileDone = false;

    System.err.println("start processing...");

    // OK, we can now start the actual transcoding.
    try {
      p.start();
      dsink.start();
    } catch (IOException e) {
      System.err.println("IO error during processing");
      return false;
    }

    // Wait for EndOfStream event.
    waitForFileDone();

    // Cleanup.
    try {
      dsink.close();
    } catch (Exception e) {}
    p.removeControllerListener(this);

    System.err.println("...done processing.");

    return true;
  }

  /**
   * Method to write out the movie for an AVI movie
   * @param width the width of the resulting movie
   * @param height the height of the resulting movie
   * @param frameRate the number of frames per second
   * @param inFiles string full path names of the frames
   * @param outML the Media Locator
   * @param type the VideoFormat type
   */
  public boolean doItAVI(int width, int height,
                         int frameRate, List<String> inFiles,
```

```
                    MediaLocator outML,
                    String type) {
    ImageDataSource ids = new ImageDataSource(width, height,
                                              frameRate, inFiles);

    Processor p;

    try {
      System.err.println("- create processor for the image datasource ...");
      p = Manager.createProcessor(ids);
    } catch (Exception e) {
      System.err.println("Yikes!  Cannot create a processor from the data
source.");
      return false;
    }

    p.addControllerListener(this);

    // Put the Processor into configured state so we can set
    // some processing options on the processor.
    p.configure();
    if (!waitForState(p, p.Configured)) {
      System.err.println("Failed to configure the processor.");
      return false;
    }

    // Set the output content descriptor to QuickTime.
    p.setContentDescriptor(new ContentDescriptor(type));

     // Query for the processor for supported formats.
    // Then set it on the processor.
    TrackControl tcs[] = p.getTrackControls();
    Format format;
   TrackControl [] arrTrackControls = p.getTrackControls();
   for ( int i = 0;  i < arrTrackControls.length;  i++ ) {
   format = arrTrackControls[i].getFormat ();
      if ( format  instanceof VideoFormat ){
         arrTrackControls[i].setFormat(new VideoFormat(VideoFormat.CINEPAK));
      }
      //else if ( format instanceof AudioFormat ){
      //arrTrackControls[i].setFormat(new AudioFormat(AudioFormat.GSM_MS));
      //}
   }

    // We are done with programming the processor.  Let's just
    // realize it.
    p.realize();
    if (!waitForState(p, p.Realized)) {
      System.err.println("Failed to realize the processor.");
      return false;
    }

    // Now, we'll need to create a DataSink.
    DataSink dsink;
    if ((dsink = createDataSink(p, outML)) == null) {
```

```
      System.err.println("Failed to create a DataSink for the given output
MediaLocator: " + outML);
      return false;
    }

    dsink.addDataSinkListener(this);
    fileDone = false;

    System.err.println("start processing...");

    // OK, we can now start the actual transcoding.
    try {
      p.start();
      dsink.start();
    } catch (IOException e) {
      System.err.println("IO error during processing");
      return false;
    }

    // Wait for EndOfStream event.
    waitForFileDone();

    // Cleanup.
    try {
      dsink.close();
    } catch (Exception e) {}
    p.removeControllerListener(this);

    System.err.println("...done processing.");

    return true;
  }


  /**
   * Create the DataSink.
   */
  DataSink createDataSink(Processor p, MediaLocator outML) {

    DataSource ds;

    if ((ds = p.getDataOutput()) == null) {
      System.err.println("Something is really wrong: the processor does not
have an output DataSource");
      return null;
    }

    DataSink dsink;

    try {
      System.err.println("- create DataSink for: " + outML);
      dsink = Manager.createDataSink(ds, outML);
      dsink.open();
    } catch (Exception e) {
      System.err.println("Cannot create the DataSink: " + e);
      return null;
```

```
  }

  return dsink;
}


Object waitSync = new Object();
boolean stateTransitionOK = true;

/**
 * Block until the processor has transitioned to the given state.
 * Return false if the transition failed.
 */
boolean waitForState(Processor p, int state) {
  synchronized (waitSync) {
    try {
      while (p.getState() < state && stateTransitionOK)
        waitSync.wait();
    } catch (Exception e) {}
  }
  return stateTransitionOK;
}


/**
 * This method is called when an event is generated by a Controller
 * that this listener is registered with.
 * @param evt the event generated
 */
public void controllerUpdate(ControllerEvent evt) {

  if (evt instanceof ConfigureCompleteEvent ||
      evt instanceof RealizeCompleteEvent ||
      evt instanceof PrefetchCompleteEvent) {
    synchronized (waitSync) {
      stateTransitionOK = true;
      waitSync.notifyAll();
    }
  } else if (evt instanceof ResourceUnavailableEvent) {
    synchronized (waitSync) {
      stateTransitionOK = false;
      waitSync.notifyAll();
    }
  } else if (evt instanceof EndOfMediaEvent) {
    evt.getSourceController().stop();
    evt.getSourceController().close();
  }
}


Object waitFileSync = new Object();
boolean fileDone = false;
boolean fileSuccess = true;

/**
 * Block until file writing is done.
```

```
 */
boolean waitForFileDone() {
  synchronized (waitFileSync) {
    try {
      while (!fileDone)
        waitFileSync.wait();
    } catch (Exception e) {}
  }
  return fileSuccess;
}


/**
 * Event handler for the file writer. This method is called when an
 * event is generated by a DataSink that this listener is registered with.
 * @param evt the event generated
 */
public void dataSinkUpdate(DataSinkEvent evt) {

  if (evt instanceof EndOfStreamEvent) {
    synchronized (waitFileSync) {
      fileDone = true;
      waitFileSync.notifyAll();
    }
  } else if (evt instanceof DataSinkErrorEvent) {
    synchronized (waitFileSync) {
      fileDone = true;
      fileSuccess = false;
      waitFileSync.notifyAll();
    }
  }
}


public static void main(String args[]) {

  if (args.length == 0)
    prUsage();

  // Parse the arguments.
  int i = 0;
  int width = -1, height = -1, frameRate = 1;
  List<String> inputFiles = new ArrayList<String>();
  String outputURL = null;

  while (i < args.length) {

    if (args[i].equals("-w")) {
      i++;
      if (i >= args.length)
        prUsage();
      width = new Integer(args[i]).intValue();
    } else if (args[i].equals("-h")) {
      i++;
      if (i >= args.length)
        prUsage();
```

```
        height = new Integer(args[i]).intValue();
      } else if (args[i].equals("-f")) {
        i++;
        if (i >= args.length)
          prUsage();
        frameRate = new Integer(args[i]).intValue();
      } else if (args[i].equals("-o")) {
        i++;
        if (i >= args.length)
          prUsage();
        outputURL = args[i];
      } else {
        inputFiles.add(args[i]);
      }
      i++;
    }

    if (outputURL == null || inputFiles.size() == 0)
      prUsage();

    // Check for output file extension.
    if (!outputURL.endsWith(".mov") && !outputURL.endsWith(".MOV")) {
      System.err.println("The output file extension should end with a .mov
extension");
      prUsage();
    }

    if (width < 0 || height < 0) {
      System.err.println("Please specify the correct image size.");
      prUsage();
    }

    // Check the frame rate.
    if (frameRate < 1)
      frameRate = 1;

    // Generate the output media locators.
    MediaLocator oml;

    if ((oml = createMediaLocator(outputURL)) == null) {
      System.err.println("Cannot build media locator from: " + outputURL);
      System.exit(0);
    }

    JpegImagesToMovie imageToMovie = new JpegImagesToMovie();
    imageToMovie.doItQuicktime(width, height, frameRate, inputFiles, oml);

    System.exit(0);
  }

  static void prUsage() {
    System.err.println("Usage: java JpegImagesToMovie -w <width> -h <height>
-f <frame rate> -o <output URL> <input JPEG file 1> <input JPEG file 2>
...");
    System.exit(-1);
  }
```

```
  /**
   * Create a media locator from the given string.
   */
  static MediaLocator createMediaLocator(String url) {

    MediaLocator ml;

    if (url.indexOf(":") > 0 && (ml = new MediaLocator(url)) != null)
      return ml;

    if (url.startsWith(File.separator)) {
      if ((ml = new MediaLocator("file:" + url)) != null)
        return ml;
    } else {
      String file = "file:" + System.getProperty("user.dir") + File.separator
+ url;
      if ((ml = new MediaLocator(file)) != null)
        return ml;
    }

    return null;
  }


  ////////////////////////////////////////////
  //
  // Inner classes.
  ////////////////////////////////////////////


  /**
   * A DataSource to read from a list of JPEG image files and
   * turn that into a stream of JMF buffers.
   * The DataSource is not seekable or positionable.
   */
  class ImageDataSource extends PullBufferDataSource {

    ImageSourceStream streams[];

    ImageDataSource(int width, int height, int frameRate,
                    List images) {
      streams = new ImageSourceStream[1];
      streams[0] = new ImageSourceStream(width, height,
                                         frameRate, images);
    }

    public void setLocator(MediaLocator source) {
    }

    public MediaLocator getLocator() {
      return null;
    }

    /**
     * Content type is of RAW since we are sending buffers of video
```

```
   * frames without a container format.
   */
  public String getContentType() {
    return ContentDescriptor.RAW;
  }

  public void connect() {
  }

  public void disconnect() {
  }

  public void start() {
  }

  public void stop() {
  }

  /**
   * Return the ImageSourceStreams.
   */
  public PullBufferStream[] getStreams() {
    return streams;
  }

  /**
   * We could have derived the duration from the number of
   * frames and frame rate.  But for the purpose of this program,
   * it's not necessary.
   */
  public Time getDuration() {
    return DURATION_UNKNOWN;
  }

  public Object[] getControls() {
    return new Object[0];
  }

  public Object getControl(String type) {
    return null;
  }
}


/**
 * The source stream to go along with ImageDataSource.
 */
class ImageSourceStream implements PullBufferStream {

  List images;
  int width, height;
  VideoFormat format;
  long frame = 0;

  int nextImage = 0; // index of the next image to be read.
  boolean ended = false;
```

```
public ImageSourceStream(int width, int height,
                         int frameRate, List images) {
  this.width = width;
  this.height = height;
  this.images = images;

  format = new VideoFormat(
                           VideoFormat.JPEG,
                           new Dimension(width, height),
                           Format.NOT_SPECIFIED,
                           Format.byteArray,
                           (float)frameRate);
}

/**
 * We should never need to block assuming data are read from files.
 */
public boolean willReadBlock() {
  return false;
}

/**
 * This is called from the Processor to read a frame worth
 * of video data.
 */
public void read(Buffer buf) throws IOException {

  // Check if we've finished all the frames.
  if (nextImage >= images.size()) {
    // We are done.  Set EndOfMedia.
    System.err.println("Done reading all images.");
    buf.setEOM(true);
    buf.setOffset(0);
    buf.setLength(0);
    ended = true;
    return;
  }

  String imageFile = (String)images.get(nextImage);
  nextImage++;

  //System.err.println("  - reading image file: " + imageFile);

  // Open a random access file for the next image.
  RandomAccessFile raFile;
  raFile = new RandomAccessFile(imageFile, "r");

  byte data[] = null;

  // Check the input buffer type & size.

  if (buf.getData() instanceof byte[])
    data = (byte[])buf.getData();

  // Check to see the given buffer is big enough for the frame.
```

```
      if (data == null || data.length < raFile.length()) {
        data = new byte[(int)raFile.length()];
        buf.setData(data);
      }

      // Read the entire JPEG image from the file.
      raFile.readFully(data, 0, (int)raFile.length());

      //System.err.println("    read " + raFile.length() + " bytes.");

      buf.setOffset(0);
      buf.setLength((int)raFile.length());
      buf.setFormat(format);
      buf.setFlags(buf.getFlags() | buf.FLAG_KEY_FRAME);
      buf.setTimeStamp(frame++ * Time.ONE_SECOND/fRate); /*AVI*/


      // Close the random access file.
      raFile.close();
    }

    /**
     * Return the format of each video frame.  That will be JPEG.
     */
    public Format getFormat() {
      return format;
    }

    public ContentDescriptor getContentDescriptor() {
      return new ContentDescriptor(ContentDescriptor.RAW);
    }

    public long getContentLength() {
      return 0;
    }

    public boolean endOfStream() {
      return ended;
    }

    public Object[] getControls() {
      return new Object[0];
    }

    public Object getControl(String type) {
      return null;
    }
  }
}
```