Analyse verschiedener Werkzeuge zur automatisierten Plagiatsprüfung von Programmtexten

Bachelorarbeit

Conni Müller

WT 2010

FAKULTÄT FÜR ELEKTROTECHNIK UND TECHNISCHE INFORMATIK
Wissenschaftliche Einrichtung 6 – Informationstechnik
Prof. Dr. Dieter Pawelczak

Betreuer: Prof. Dr. Dieter Pawelczak

20. Dezember 2013

ERKLÄRUNG

Ich versichere, dass ich diese Bachelorarbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht. Weiterhin versichere ich, dass diese Arbeit weder bei einer anderen Prüfungsbehörde vorgelegt noch veröffentlicht wurde.

20. Dezember 2013
. . . . . .
Datum Unterschrift

Inhalt

Abbildungs- und Tabellenverzeichnis

1 Einführung

Ein Plagiat (frz. plagiaire „Diebstahl geistigen Eigentums" aus lat. plagiārius „Seelenverkäufer, Menschenräuber") ist eine fremde Leistung, die als die eigene Leistung ausgegeben wird. [10] Dies ist verwerflich, denn ein Diebstahl ist ein Verbrechen und kein einfaches Kavaliersdelikt. Außerdem kann mit einem Plagiat nicht nur ein wirtschaftlicher Schaden eintreten, den ein Unternehmen erleiden kann, wenn sein Produkt plagiiert wird, sondern auch ein Imageschaden, wenn das Vertrauen in eine Person oder ein Unternehmen verloren geht.

Es gibt Plagiate auf jedem Gebiet: in der Malerei, in der Musik, in der Literatur und dem Journalismus, bei Produkten, in Wissenschaften, etc.

In wissenschaftlichen Publikationen, sind seit 2011 vermehrt Plagiate durch Gremien an Universitäten, aber auch durch einzelne Personen aufgedeckt worden. Dies ist auf die moderne Technik zurückzuführen. Während es in der Vergangenheit zeitaufwändig und beschwerlich war, nachzuprüfen, ob alle Zitate kenntlich gemacht oder Passagen aus anderen Veröffentlichungen kopiert worden, bieten heute spezielle Programme und das Internet Möglichkeiten der vereinfachten Kontrolle von schriftlichen Arbeiten.

Das wohl bekannteste Beispiel hierfür ist die Guttenberg-Affäre. Anfang 2011 wurde dem damaligen deutschen Bundesverteidigungsminister Karl-Theodor zu Guttenberg nachgewiesen, dass seine Doktorarbeit plagiiert war und ihm in diesem Zuge die Doktorwürde aberkannt. Zusätzlich dazu musste Karl-Theodor zu Guttenberg auf Grund des verlorenen Vertrauens seine politischen Ämter niederlegen.[9] Einige weitere Politiker mussten ihm folgen und ebenfalls ihre Ämter und den Doktortitel ablegen, da auch in ihren Dissertationen Plagiate nachgewiesen wurden.

Im vorgestellten Beispiel wurde ein Plagiat in einer umfangreichen, schriftlichen Arbeit festgestellt, aber das Plagiieren ist bereits in früheren Phasen der akademischen Ausbildung ein Thema. In einer Hausarbeit oder beim Programmieren von Software im Rahmen des Studiums wird gerne kopiert, ohne den Urheber kenntlich zu machen. Um dies in Zukunft zu verhindern, müssen aktuelle Fälle aufgedeckt werden. Dafür benötigen Professoren, Dozenten und Lehrer Hilfe, um vor allem den Zeitaufwand zu minimieren. Im Bereich der Softwareprogrammierung gibt es zum Teil bereits seit 1994 Werkzeuge, die zwei Programmtexte miteinander vergleichen und Gemeinsamkeiten aufzeigen. Im Rahmen dieser Bachelorarbeit werden einige dieser Prüfprogramme vorgestellt und analysiert, um dem Anwender eine Überblick über die zur Verfügung stehenden Hilfsmittel zu verschaffen und ihm die Möglichkeit zu bieten, das für ihn geeignete Werkzeug zu wählen.

2 Plagiieren im Rahmen des MOP-Praktikums

In der Fakultät Elektrotechnik und Technische Informatik (ETTI) der Universität der Bundeswehr München (UniBwM) starten die Studenten im ersten Studienjahr mit sehr unterschiedlichen Voraussetzungen. Während einige der neuen Studenten bereits eine Lehre im Bereich der Elektrotechnik oder Informatik absolviert haben, kommen andere direkt von der Schule. Für die Professoren der Wissenschaftliche Einrichtung 6 – Informationstechnik (WE6 / ETTI6) sind diese unterschiedlichen Voraussetzungen immer auch eine Herausforderung. Auf der einen Seite gibt es Studenten, für die sehr viele neue Bereiche dabei sind und die das Programmieren von Grund auf erlernen müssen, auf der anderen Seite gibt es Studenten, die sich in den Grundlagenfächern schnell langweilen, weil sie das alles schon kennen und können. Der Spagat, die Anfänger mitzunehmen, aber gleichzeitig die Fortgeschrittenen nicht zu verlieren, ist dabei die hohe Kunst.

Um das in den Vorlesungen Vermittelte zu vertiefen und anzuwenden finden parallel zu den Vorlesungen Praktika statt. Laut Modulhandbuch ist beispielsweise der Arbeitsaufwand für eine Praktikumsaufgabe im Fach "Maschinenorientiertes Programmieren" (MOP) relativ gering (11 Stunden/Woche)[8], jedoch sieht es in der Praxis anders aus. Während Studenten mit Vorkenntnissen weniger Zeit benötigen, ist das bei den Anfängern oft das Gegenteil. Um sich diesen Zeitaufwand zu sparen, greifen Studenten oft auf die Lösungen von Kommilitonen des gleichen oder vorangegangenen Jahrgangs zurück. Da von den Studenten erwartet wird, dass sie eine selbstständig programmierte Lösung im Praktikum einreichen, tritt hier ein Konflikt auf.

Für die Zeit des Praktikums stehen Betreuer bereit, um eventuell auftretende Fragen zu klären beziehungsweise die Lösung der Studenten abzunehmen. Dabei muss aber nicht nur ein Programmcode (Programmtext) abgegeben, sondern immer auch ein Fragebogen ausgefüllt werden. Die Studenten werden außerdem im Vorfeld darüber aufgeklärt, dass ihr Programmcode einer Plagiatsprüfung unterzogen wird. Oftmals genügt ein kurzes Gespräch zur aktuellen Aufgabe, um festzustellen, ob ein Student die vorliegende Arbeit selbst erbracht hat oder nicht. Da dies nicht immer möglich ist, kommt die Überprüfung als zweites Mittel hinzu. Vor einigen Jahren noch mussten sämtliche Programmtexte dabei von Hand überprüft werden. Das war zeitaufwendig. Nachdem Prof. Dr. Pawelczak eine Plagiatserkennungssoftware für das Praktikum zur Vorlesung "Maschinenorientiertes Programmieren" geschrieben hat, können die Programme der Studenten noch vor Ort überprüft werden.

Wie bereits erwähnt, gibt es für die Plagiatserkennung bei Programmtexten bereits Programme, die im Internet zur Verfügung stehen. Im Rahmen dieser Bachelorarbeit sollen einige dieser Programme, sowie das an der Fakultät ETTI geschriebene Programm getestet und die Ergebnisse ausgewertet werden.

3 Plagiatserkennungssoftware

In diesem Kapitel werden die drei Werkzeuge zur automatisierten Plagiatsprüfung von Programmtexten und ihre Suchalgorithmen vorgestellt.

Die drei Werkzeuge sind Measure of Software Similarity (MoSS), JPlag und PlagC2. Die folgende Tabelle gibt einen kurzen Überblick über die zu analysierenden Programme.

Name	Entstehungsjahr/ Entstehungsort	Entwickler	Suchalgorithmus
Measure of Software Similarity (MoSS)	1994 / Stanford University	Alex Aiken	Winnowing-Algorithmus
JPlag	2000 / Universität Karlsruhe	Lutz Prechelt Guido Malpohl Michael Phlippsen	Greedy-String-Tiling-Algorithmus Und Rabin-Karp-Algorithmus
PlagC2	2012 / Universität der Bundeswehr München	Dieter Pawelczak	Greedy-String-Tiling-Algorithmus

Tabelle 1: Gegenüberstellung der Plagiatserkennungsprogramme

Zunächst sollen die Programme und die verwendeten Algorithmen genauer betrachtet werden, um einen ausführlichen Vergleich zu ermöglichen. Dabei soll das Hauptaugenmerk auf dem neuesten Programm, PlagC2, liegen.

3.1 Measure of Software Similarity – MoSS

Die Plagiatserkennungssoftware Measure of Software Similarity (MoSS) ist "ein automatisiertes System zum Feststellen von Ähnlichkeiten zwischen Programmen"[1]. Sie wurde bereits 1994 von Alex Aiken an der Stanford University, USA, entwickelt.

MoSS ist eine Client-Server-Anwendung, bei der sich der Server an der Stanford University befindet. Bei dem Client handelt es sich um ein Perl-Skript, welches sowohl unter Unix als auch Cygwin/Windows angewandt werden kann. Das Programm kann unter anderem die Programmiersprachen C, C++, Java, C#, Python, Visual Basic und Javascript auswerten. Um das Perl-Skript nutzen zu können, muss man sich unter http://theory.stanford.edu/~aiken/moss/ registrieren und erhält das Skript per E-Mail zugesandt. MoSS ist derart konzipiert, dass es alle eingereichten Daten gleichzeitig miteinander vergleicht und das Ergebnis als Hypertext Markup Language-Datei (HTML) aufbereitet und eine Uniform Resource Locator (url) generiert, unter der die Ergebnisse für

[1] http://theory.stanford.edu/~aiken/moss/ (Zugriff: 28.10.2013)

den Nutzer einsehbar sind. Um den Server nicht zu überlasten, bleiben die Ergebnisse 14 Tage erhalten und werden dann vom Server gelöscht.

Der von MoSS genutzte Algorithmus, ist der Winnowing-Algorithmus. Er wird im Kapitel 3.4.3 näher erläutert.

3.2 JPlag

JPlag "ist ein System, welches Paare von ähnlichen Programmen in einem gegebenen Satz von Programmen findet"[2]. Das Programm wurde 2000 an der Universität Karlsruhe von Lutz Prechelt, Guido Malpohl und Michael Phlippsen entwickelt.

JPlag ist, genau wie MoSS, eine Client-Server-Anwendung. Der Server befindet sich an der Universität Karlsruhe. Allerdings wird dem Nutzer kein Skript, das er ausführen muss, zur Verfügung gestellt, sondern er muss sich unter https://jplag.ipd.kit.edu/ für einen Webservice registrieren. Einmal registriert, bietet der Webservice die Möglichkeit einen Satz von Daten auf den Server hochzuladen und analysieren zu lassen. Das Ergebnis dieser Analyse wird als HTML-Datei aufbereitet und dem Nutzer innerhalb des Webservice zur Verfügung gestellt. Der Anwender erhält dabei unter anderem eine Statistik, wie viele Übereinstimmungen in welchem Prozentbereich liegen, beispielsweise 207 Übereinstimmungen im Bereich 20-30%. Die prozentual ähnlichsten Daten können über den Webservice eingesehen und direkt verglichen werden. Die Daten werden 30 Tage auf dem Server gespeichert und dann gelöscht.

JPlag benutzt zwei Algorithmen: den Greedy-String-Tiling-Algorithmus (Kapitel 3.4.1) und, zur Beschleunigung der Suche, den Rabin-Karp-Algorithmus (Kapitel 3.4.2).

3.3 PlagC2

Das Programm PlagC2 wurde von Dieter Pawelczak 2012 an der UniBwM entwickelt. PlagC2 ist Teil eines größeren Systems, zu dem unter anderem eine Benutzerschnittstelle und eine Datenbank gehören. Dieses System ist eine Client-Server-Anwendung. Die Benutzerschnittstelle des Systems wird dazu verwendet, Daten an den Server zu schicken und PlagC2 zu starten. Die gefundenen Übereinstimmungen werden in der Datenbank abgespeichert und können über die Benutzerschnittstelle abgefragt werden. In dieser Bachelorarbeit wird allerdings nur die eigentliche Plagiatserkennungssoftware PlagC2 betrachtet. Im Gegensatz zu MoSS und JPlag werden die Daten nicht alle gleichzeitig untersucht, sondern sukzessive. Das bedeutet, dass die erste abgegebene Lösung als Referenz für alle weiteren Abgaben dient und jede weitere Lösung mit alle vorher eingereichten Lösungen verglichen wird.

PlagC2 verwendet den Greedy-String-Tiling-Algorithmus zur Suche von Übereinstimmungen. Genau wie bei MoSS und JPlag werden bei PlagC2 Übereinstimmungen aufgezeigt. Eine Bewertung hinsichtlich eines Plagiates nimmt keines der drei Werkzeuge vor. Die Bewertung muss durch den Menschen erfolgen.

[2] digbib.ubka.uni-karslruhe.de/volltexte/542000 (Zugriff: 28.10.2013)

3.4 Such-Algorithmus

In diesem Abschnitt werden die drei bereits erwähnten Algorithmen kurz erläutert.

Sowohl PlagC2 als auch JPlag verwenden den Greedy-String-Tiling-Algorithmus. JPlag nutzt zusätzlich den Rabin-Karp-Algorithmus. Als dritter Algorithmus wird der Winnowing-Algorithmus, den MoSS verwendet, kurz erläutert.

3.4.1 Greedy-String-Tiling-Algorithmus

Der Greedy-String-Tiling-Algorithmus (GSTA) wurde 1993 von Michael J. Wise eingeführt. [7]

Der GSTA wird dazu verwendet, Ähnlichkeiten zwischen zwei Sequenzen, zum Beispiel Zeichenketten oder ganzen Texten, zu finden. Der Algorithmus besteht dabei aus mehreren ineinander verschachtelten Schleifen, um alle Übereinstimmungen zu finden. Wurde in den Sequenzen die längste zusammenhängende Übereinstimmung gefunden, so wird diese markiert und fällt beim nächsten Suchdurchlauf raus. Damit ist sichergestellt, dass die Subsequenzen, die verglichen werden, immer kleiner werden und der Algorithmus terminiert. Allerdings kann der Algorithmus auf diese Weise auch Sequenzen der Länge 1 finden. Da dies nicht das Ziel ist, wird eine minimale Zeichenlänge, die sogenannte MinimumMatchLength (MML), festgelegt. Subsequenzen innerhalb des Textes, die kürzer als diese MML sind, das heißt weniger Zeichen besitzen, werden nicht markiert. Auf diese Art und Weise terminiert der GSTA, wenn er keine Sequenzen mehr findet, die größer oder gleich der MML sind.

Abb. 1 zeigt den Pseudocode für den Greedy-String-Tiling-Algorithmus.
Das Symbol in Zeile 12 - $\oplus$ - besagt, dass eine gefundene Übereinstimmung nur dann markiert wird, wenn es zum einen noch nicht in der Liste vorhanden ist und sich zum anderen mit keiner bisher gefundenen Übereinstimmung in der Liste überschneidet. Die Liste (*tiles*) enthält am Ende alle gefundenen Übereinstimmungen (*matches*).

```
0     Greedy-String-Tiling(String A, String B) {
1           tiles = { };
2           do{
3                 maxmatch = MinimumMaxLength;
4                 matches = { };
5                 Forall unmarked tokens A_a in A {
6                       Forall unmarked tokens B_b in B {
7                             j = 0;
8                             while(A_a+j == B_b+j && unmarked(A_a+j) &&
9                                   umarked(B_b+j))
10                                  j++;
11                            if(j == maxmatch)
12                                  matches = matches ⊕ match(a, b, j);
13                            else if(j > maxmatch) {
14                                  matches = { match(a, b, j) };
15                                  maxmatch = j;
16                            }
17                      }
18                }
19                Forall match(a, b, maxmatch) ∈ matches {
20                      For j= 0 ... (maxmatch - 1) {
21                            mark(A_a+j);
22                            mark(B_b+j);
23                      }
24                      tiles = tiles ∪ match (a, b, maxmatch);
25                }
26          } while (maxmatch > MinimumMaxLength);
27          return tiles;
28    }
```

Abb. 1: Pseudocode für Greedy-String-Tiling-Algorithmus

Um den dargestellt Code besser zu verstehen, wird er schrittweise erklärt.
Der Algorithmus besteht aus insgesamt sechs Schleifen, wobei die erste Schleife S1 (Zeile 2-26) die Bedingung zur Wiederholung der Suche enthält und die einzelnen Durchläufe steuert. Die in S1 liegenden Schleifen S2-S6 können in zwei Phasen P1 und P2 unterteilt werden. Wobei die Schleifen S2-S4 (Zeile 5-18) zur ersten und die Schleifen S5-S6 (Zeile 19-25) zur zweiten Phase gehören.

In P1 werden über sämtliche noch nicht markierten Token (deutsch: Zeichen) in der Sequenz A (S1, Zeile 5) und der Sequenz B (S2, Zeile 6) iteriert. Zunächst wird dabei die Zeichenanzahl *j* auf 0 gesetzt (Zeile 10). Dann wird *j* solange inkrementiert, wie die gefundenen Zeichen in A und B identisch sind und beide nicht markiert sind. Dies erfolgt in S4 (Zeile 8-10). Sobald die vierte Schleife terminiert, enthält *j* die Anzahl der zusammenhängenden, übereinstimmenden Zeichen aus den Sequenzen A und B. Im Anschluss wird in einer if-else-Abfrage überprüft, ob *j* dem *maxmatch*, also der längsten Zeichenanzahl, entspricht oder größer ist. Entspricht *j* dem *maxmatch* (Zeile 11-12), so wird der gefundene *match* in die Liste der matches aufgenommen. Allerdings nur, wenn der *match* noch nicht enthalten ist und sich mit keinem in *matches* beinhalteten *match* überschneidet. Wenn *j* größer als *maxmatch* ist, so werden alle bis dahin gefundenen Übereinstimmungen aus der Liste *matches* gelöscht und nur die aktuell gefundene Sequenz abgespeichert.

Sobald die Phase P1 beendet ist, werden alle gefundenen Übereinstimmungen in *tiles* übernommen (S5, Zeile 19) und die gefundenen *matches* markiert (S6, Zeile 20-24), damit sie in nächsten Suchlauf nicht mehr berücksichtigt werden.

Zum Schluss wird in der Schleife S1 überprüft, ob die längste gefundene Zeichenanzahl *maxmatch* länger als die MML ist. Wenn dies der Fall ist, wird ein neuer Suchdurchlauf gestartet. Ansonsten terminiert der GSTA.

Die Laufzeit des GSTA wurde bereits bei Prechelt [1] ausführlich erklärt. Als kurze Zusammenfassung kann gesagt werden, dass im schlechtesten Fall (worst case) der GSTA eine Laufzeit von n^3 hat. Die Variable n repräsentiert die Anzahl der Zeichen in der kürzeren der zwei zu vergleichenden Sequenzen. Der worst case tritt dann ein, wenn bei jedem Suchdurchlauf die inneren Schleifen jedes Mal vollständig durchlaufen werden und immer nur eine Übereinstimmung gefunden wird. Im besten Fall (best case) benötigt der GSTA eine Zeit von n^2. Der best case tritt ein, wenn die zwei zu vergleichenden Sequenzen vollständig unterschiedlich sind.

3.4.2 Rabin-Karp-Algorithmus

Wie bereits weiter oben erwähnt, verwendet JPlag zusätzlich zum GSTA, den Rabin-Karp-Algorithmus (RKA), um die Suche zu beschleunigen und die Laufzeit im best case auf $<n^2$ zu verringern. Dieser Algorithmus ist von Michael O. Rabin und Richard M. Karp entwickelt worden um Muster in Texten zu entdecken. Der RKA arbeitet dabei mit Hash-Werten, die aus den Subsequenzen errechnet werden. Ein Hash-Wert ist ein Wert fester Länge. Dieser Wert kann zum Beispiel aus dem ASCII-Wert eines Zeichens multipliziert mit einem festgelegten Faktor berechnet werden. Wenn der Hash-Wert einer Zeichenfolge berechnet werden soll, dann werden die berechneten Werte der einzelnen Zeichen dieser Folge aufsummiert. Dem RKA wird dabei die Zeichenlänge (Muster m) vorgegeben. Danach geht der Algorithmus den Text der Länge n durch und errechnet aus jeder m-langen Zeichenfolge einen Hash-Wert und speichert diesen in einer Tabelle ab. Damit hat der RKA eine annähernd lineare Laufzeit, da er circa n-Mal die Sequenzen durchlaufen muss. Diesem Prinzip haben sich die Entwickler von JPlag angenommen. Da allerdings auch unterschiedliche Zeichenfolgen den gleichen Hash-Wert haben können, reicht es nicht aus, nach gleichen Hash-Werten in den einzelnen, zu den zu vergleichenden Texten gehörenden Tabellen, zu suchen, sondern die Zeichenfolgen werden dann mit den GSTA auf Gleichheit untersucht. Allerdings ist der Vergleich dann bereits auf anzunehmende gleiche Zeichenketten reduziert.

3.4.3 Winnowing-Algorithmus

Der Winnowing-Algorithmus (WA) wurde 2003 von Saul Schleimer, Daniel S. Wilkerson und Alex Aiken vorgestellt.[6] Dieser Algorithmus arbeitet ebenfalls mit Hash-Werten, die im Englischen auch als "Fingerprint" bezeichnet werden. Diese "Fingerprints" (deutsch: Fingerabdruck) sollen eine genaue Identifizierung eines Textes ermöglichen, genau wie ein Fingerabdruck eines Menschen, dieses eindeutig identifiziert. Zunächst werden beim WA, genau wie beim RKA, Hash-Werte eines Textes oder einer Sequenz ermittelt und in einer

Tabelle gespeichert. Im Anschluss werden die Hash-Werte gruppiert, zum Beispiel in Gruppen von vier: (17, 74, 42, 17). Danach wird jeder Gruppe durchgegangen und der niedrigste Hash-Wert, im Beispiel "17", ausgewählt. Sollte der niedrigste Hash-Wert mehrfach vorkommen, so wird der Wert, der am weitesten rechts in der Gruppe steht, ausgewählt. Diese ausgewählten Hash-Werte werden als "Fingerprints" des Textes gespeichert. Um eine Subsequenz identifizieren zu können, wird zumeist zusätzlich zu dem Hash-Wert noch seine Position (hier: seine Gruppe) abgespeichert, zum Beispiel (17, 3) mit 17 als Hash-Wert und 3 als Position. Im Anschluss werden die "Fingerprints" indiziert und die "Fingerprints" der einzelnen Texte, Sequenzen, Dokumente erneut durchlaufen. Wenn gleiche "Fingerprints" gefunden wurden, werden Paare gebildet, zum Beispiel A(17, 3)-B(17,5). Zum Schluss werden die Informationen nach Anzahl der Übereinstimmungen in zwei Texten sortiert und von MoSS an den Nutzer ausgegeben.

Abb. 2 zeigt Code für den Kern der winnow-Funktion.

```
void winnow(int w /*window size*/) {
  // circular buffer implementing window of size w
  hash_t h[w];
  for (int i=0; i<w; ++i) h[i] = INT_MAX;
  int r = 0;                // window right end
  int min = 0;              // index of minimum hash
  // At the end of each iteration, min holds the
  // position of the rightmost minimal hash in the
  // current window.  record(x) is called only the
  // first time an instance of x is selected as the
  // rightmost minimal hash of a window.
  while (true) {
    r = (r + 1) % w;        // shift the window by one
    h[r] = next_hash();     // and add one new hash
    if (min == r) {
      // The previous minimum is no longer in this
      // window.  Scan h leftward starting from r
      // for the rightmost minimal hash.  Note min
      // starts with the index of the rightmost
      // hash.
      for(int i=(r-1)%w; i!=r; i=(i-1+w)%w)
        if (h[i] < h[min]) min = i;
      record(h[min], global_pos(min, r, w));
    } else {
      // Otherwise, the previous minimum is still in
      // this window. Compare against the new value
      // and update min if necessary.
      if (h[r] <= h[min]) {  // (*)
        min = r;
        record(h[min], global_pos(min, r, w));
      }
    }
  }
}
```

Abb. 2: Code für Winnowing-Algorithmus

4 Datensätze

Für die Analyse der Werkzeuge wird ein aktueller Datensatz benötigt. Hierfür stehen die Daten des Jahrgangs 2012 der Fakultät ETTI zur Verfügung. Die Daten wurden vorher anonymisiert, d.h. jegliche Namen oder sonstige Bezeichnungen, die auf den Studenten rückschließen lassen, entfernt und die Programmtexte durchnummeriert. Insgesamt wurden während des MOP-Praktikums sieben Versuche durchgeführt, von denen drei Versuche exemplarisch als Untersuchungsdaten herangezogen werden.

Aus langjähriger Erfahrung der Professoren ist es bekannt, dass vor allem Programmierkonzepte wie Listen und Binärbäume den Studenten am meisten Probleme bereiten und am zeitaufwändigsten zu lösen sind. Aus diesem Grund wurden die Versuche V6 und V7 als Untersuchungsdaten ausgewählt. Zusätzlich dazu wurde der Versuch V1 als dritter Versuch gewählt.

V1 ist die erste MOP-Praktikumsaufgabe für die Studenten des Jahrgangs 2012 gewesen. In diesem Versuch geht es um Integerarithmetik (Ganzzahlarithmetik). Die Studenten sollen Fibonaccizahlen und den Goldenen Schnitt ermitteln. Da in der Praktikumsaufgabe die notwendigen Formeln vorgegeben sind, ist dieser Versuch ein kurzer, schnell zu lösender Versuch gewesen. Es kann davon ausgegangen werden, dass die Studenten die Aufgabe ohne fremde Hilfe lösen können und der Grad des Plagiierens gering sein sollte.

V6 ist die vorletzte MOP-Praktikumsaufgabe, in der es um die Erstellung und Verwaltung von Listen geht. Im Versuch V7, der letzten MOP-Praktikumsaufgabe geht es um die Erstellung und Verwaltung von Binärbäumen und die Verwaltung von Dateien. Beide Konzepte verlangen ein grundlegendes Verständnis vom Programmieren und sind zeitaufwändige Aufgaben. Der Grad des Plagiierens ist als erhöht anzunehmen.

5 Inhalt der Praktikumsversuche

Auf Basis der Komplexität und des Inhalts der Praktikumsversuche wurden zur Detektion eines Plagiats Schwellenwerte (Threshold) definiert, ab denen eine Lösung als Plagiat eingestuft wird. Bei einer Übereinstimmung von 83% gilt Versuch V6 als plagiiert, V1 ab 88%, V7 ab 90%. Der niedrigste Schwellenwert begründet sich darin, dass bei der programmatischen Erstellung von Listen vielfache Vorgehensweisen koexistieren und den Studenten zu diesem Zeitpunkt auch bekannt sind. Die Wahrscheinlichkeit gleichen Programmcodes ist daher niedriger als bei dem Versuch mit Binärbäumen.

Bei Versuch V1 werden auf 69 Lösungen und bei den beiden anderen Versuchen auf 56 Lösungen zurückgegriffen. Das ist darauf zurückzuführen, dass nicht alle Studenten die Versuche V6 und V7 erfolgreich abgeschlossen haben. Dies bedeutet bei einem Vergleich aller Lösungen miteinander:

- V1 $\sum_{i=1}^{68} i = 2346$ Vergleiche
- V6 & V7 $\sum_{i=1}^{55} i = 1540$ Vergleiche

5.1 Versuch V1

Wie bereits erwähnt, geht es in V1 um Integerarithmetik. Der Schwellenwert, ab dem eine Lösung eines Studenten als Plagiat deklariert wird liegt bei 88%.

Die Abbildungen 3 bis 5 zeigen die Ergebnisse der einzelnen Plagiatserkennungsprogramme.

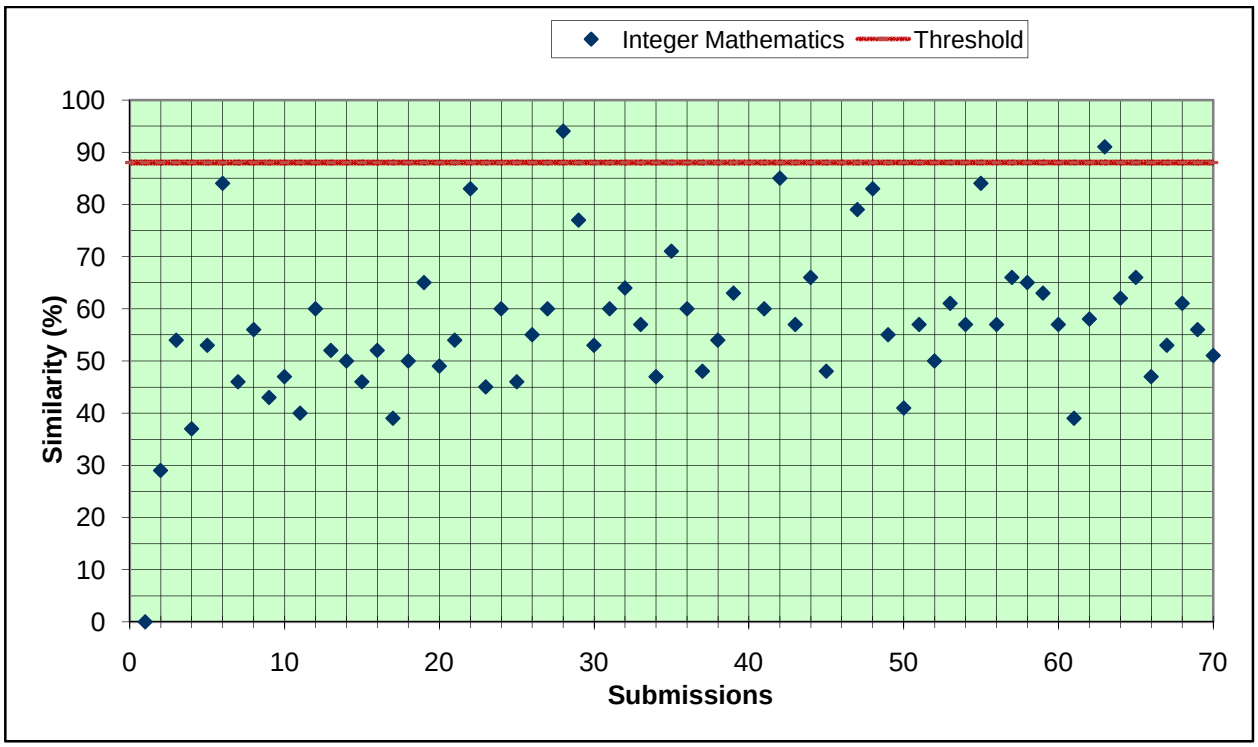

Abb. 3: Ergebnis V1 PlagC2

Die x-Achse stellt die eingereichten Lösungen (Submissions) dar und die y-Achse die prozentuale Ähnlichkeit (Similarity). Wie bereits vorher genannt, dient die erste eingereichte Lösung bei PlagC2 als Referenz, so dass für diese Lösung kein Wert einer Übereinstimmung ermittelt werden kann.

Nach der Prüfung mit PlagC2 und dem definierten Schwellenwert ist davon auszugehen, dass zwei Lösungen Plagiate sind. Bei fünf weiteren Lösungen ist die Wahrscheinlichkeit eines Plagiats aufgrund der sehr hohen Übereinstimmung hoch.

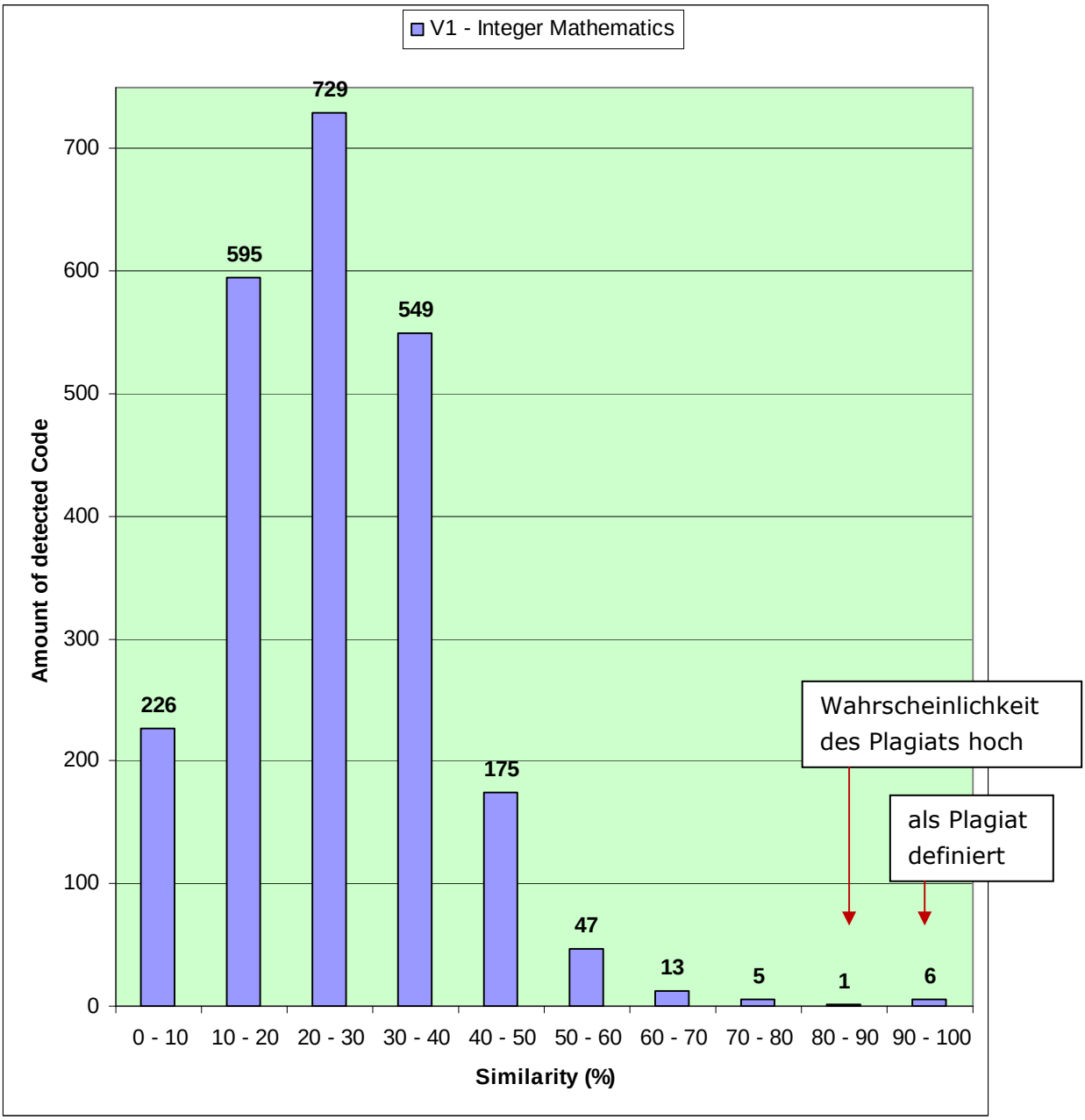

Abb. 4: Ergebnis von V1 JPlag

Die x-Achse stehlt die prozentuale Ähnlichkeit in 10er Gruppen dar (0-10%, 10-20%, et cetera). Die y-Achse stellt die Anzahl der Vergleiche dar, die in den jeweiligen Bereich von Übereinstimmungen, zum Beispiel 226 Vergleiche im Bereich 0-10% Ähnlichkeit, fallen. Bei JPlag werden alle Dateien gleichzeitig eingereicht und alle miteinander verglichen. Daraus

ergibt sich eine andere Darstellung der Ergebnisse. JPlag gibt nicht an, wie hoch die Wahrscheinlichkeit ist, dass eine Lösung ein Plagiat ist, sondern wie viele Vergleiche (V1 = 2346) zu wie viel Prozent Ähnlichkeiten aufweisen.

Insgesamt ist festzustellen, dass JPlag sechs Lösungen als Plagiate detektiert hat und eine weitere Lösung, bei der die Wahrscheinlichkeit eines Plagiates hoch ist.
Damit haben sowohl PlagC2 als auch JPlag im Schnitt die gleiche Anzahl von Plagiaten erkannt. Ob es sich tatsächlich um Plagiate handelt muss bei beiden Programmen durch die manuelle Überprüfung durch den Menschen erweisen. Des Weiteren ist aus den Ergebnissen von JPlag nicht zu erkennen, um welche Lösungen es sich im Detail handelt.

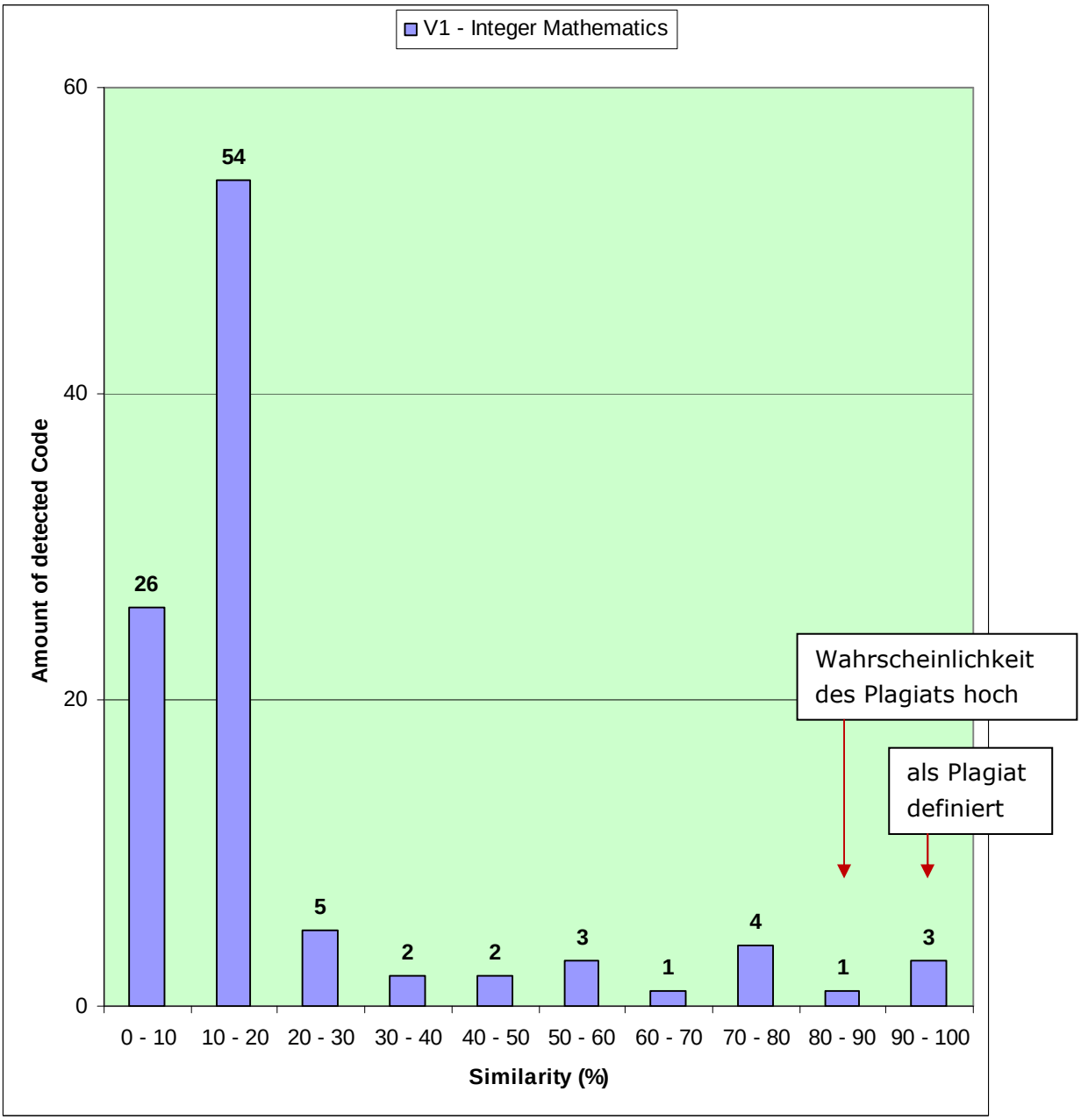

Abb. 5: Ergebnis von V1 Measure of Software Similarity

Genau wie bei JPlag werden auch bei MoSS alle Dateien gleichzeitig eingereicht und eine Totalsuche durchgeführt. Die Auftragung im Diagramm entspricht jener für JPlag. MoSS gibt

im Gegensatz zu JPlag keine Gesamtübersicht aus, wie viele Vergleiche in welchem Prozentbereich liegen, so dass für V1 nur 101 statt der erwarteten 2346 Vergleiche vorliegen.

Aus diesem Ergebnis ist zu erkennen, dass MoSS insgesamt drei Lösungen als Plagiate und eine weitere Lösung mit hoher Wahrscheinlichkeit für ein Plagiat detektiert hat. Auch hier ist nicht zu erkennen, um welche Lösungen es sich im Detail handelt. Eine manuelle Überprüfung durch den Anwender ist erforderlich.

5.2 Versuch V6

Im Versuch V6 geht es um die Erstellung und Verwaltung von Listen. Der Schwellenwert ist auf 83% gesetzt und damit der strengste Wert aller drei Versuche.

Die Abbildungen 6 bis 8 stellen die Ergebnisse der einzelnen Plagiatserkennungsprogramme dar.

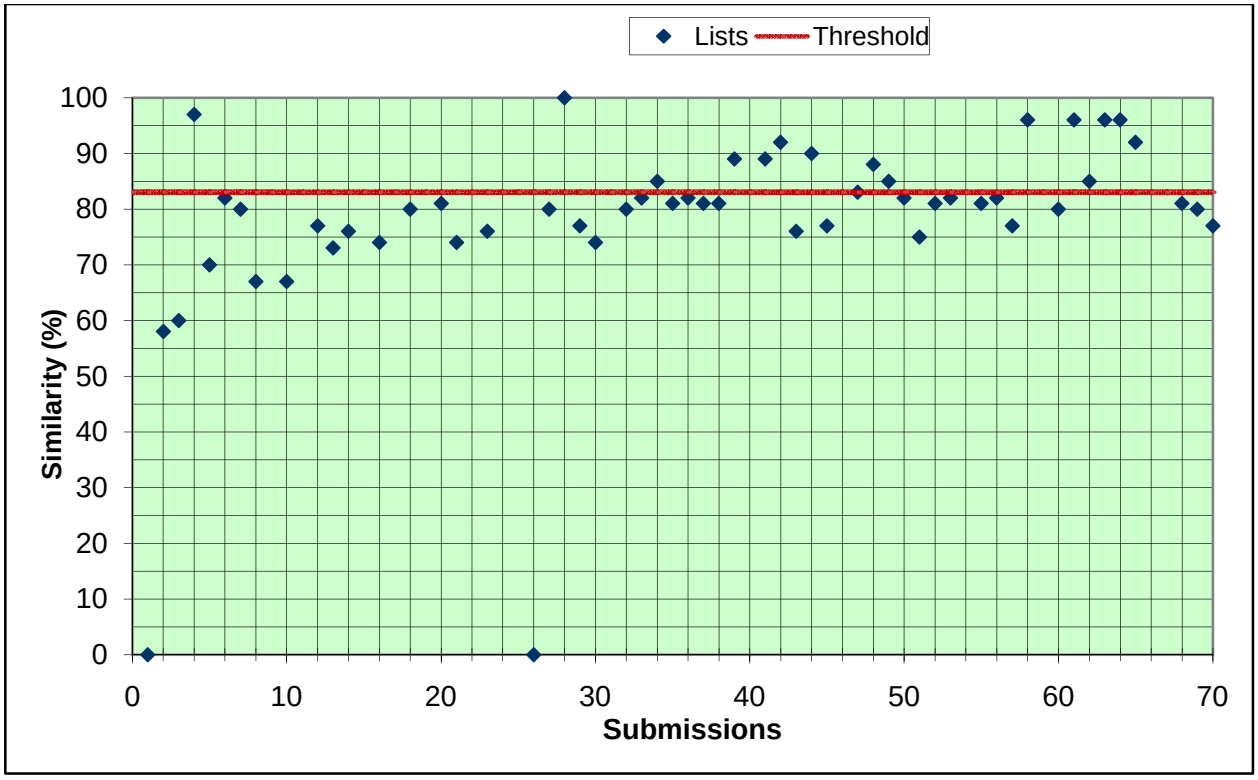

Abb. 6: Ergebnis V6 PlagC2

Durch den niedrigeren Schwellenwert und die höhere Komplexität der Aufgabe sieht das Ergebnis von V6 erheblich anders aus als bei V1. Interessant ist dabei, dass sowohl Lösung 1 als auch Lösung 26 laut PlagC2 keinerlei Ähnlichkeit aufweisen. Dagegen handelt es sich bei Lösung 28 um ein eindeutiges Plagiat (100%). Insgesamt liegen 30,4% aller Lösungen oberhalb des Schwellenwertes. Dieser Wert bestätigt die Annahme, dass der Grad des Plagiierens bei V6 im Verhältnis zu V1 erhöht sein wird.

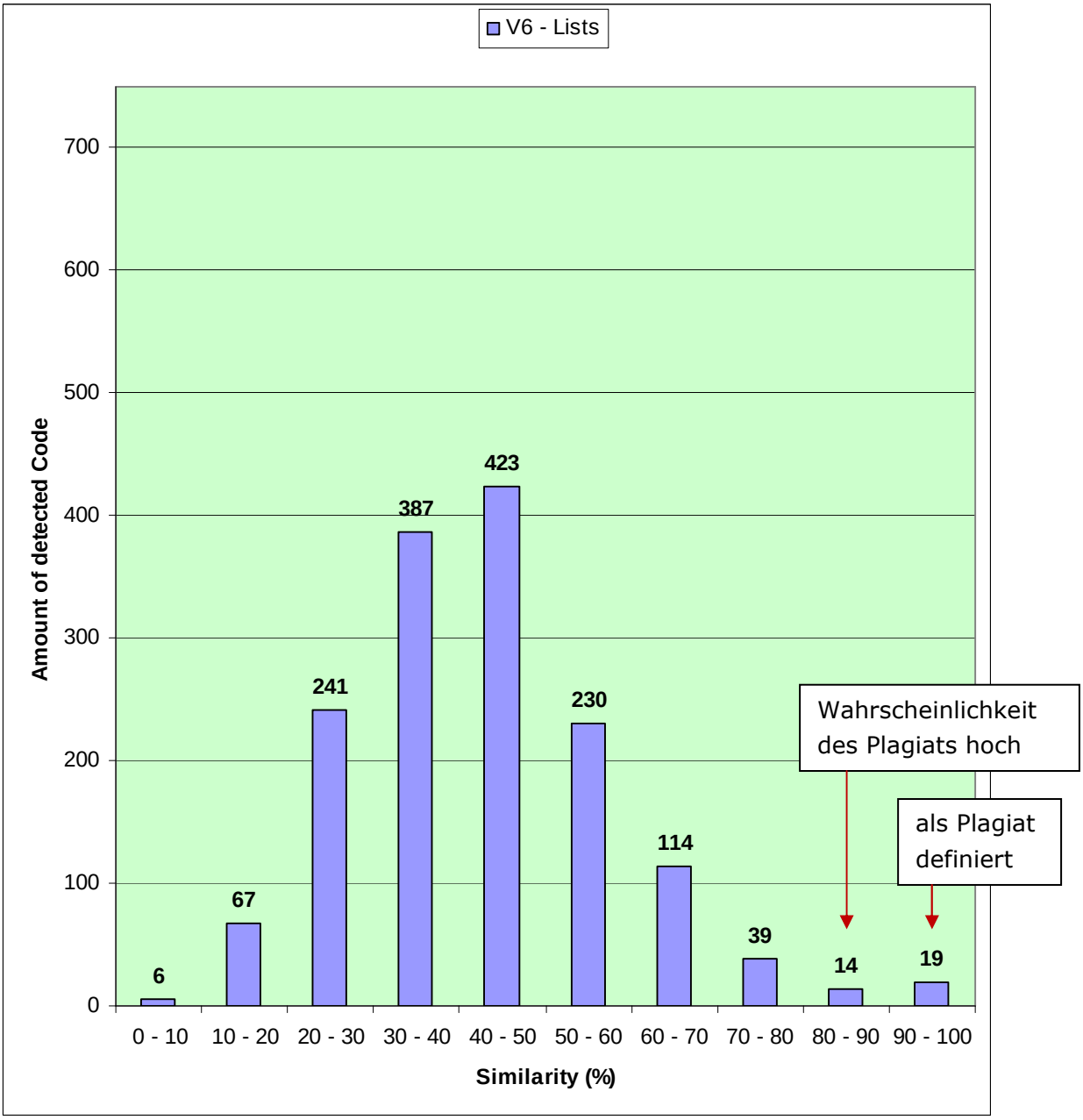

Abb. 7: Ergebnis V6 JPlag

Laut JPlag liegen insgesamt 33 von 1540 Vergleichen an oder über dem angegebenen Schwellenwert von 83%. Bei nähere Betrachtung der Ergebnisse (siehe Daten-CD: JPlag-Ergebnisse), wurde nicht nur Lösung 28 als 100%iges Plagiat detektiert, sondern noch vier weitere Lösungen. Damit erweist sich JPlag im Vergleich zu PlagC2 bei V6 als das strengere Werkzeug.

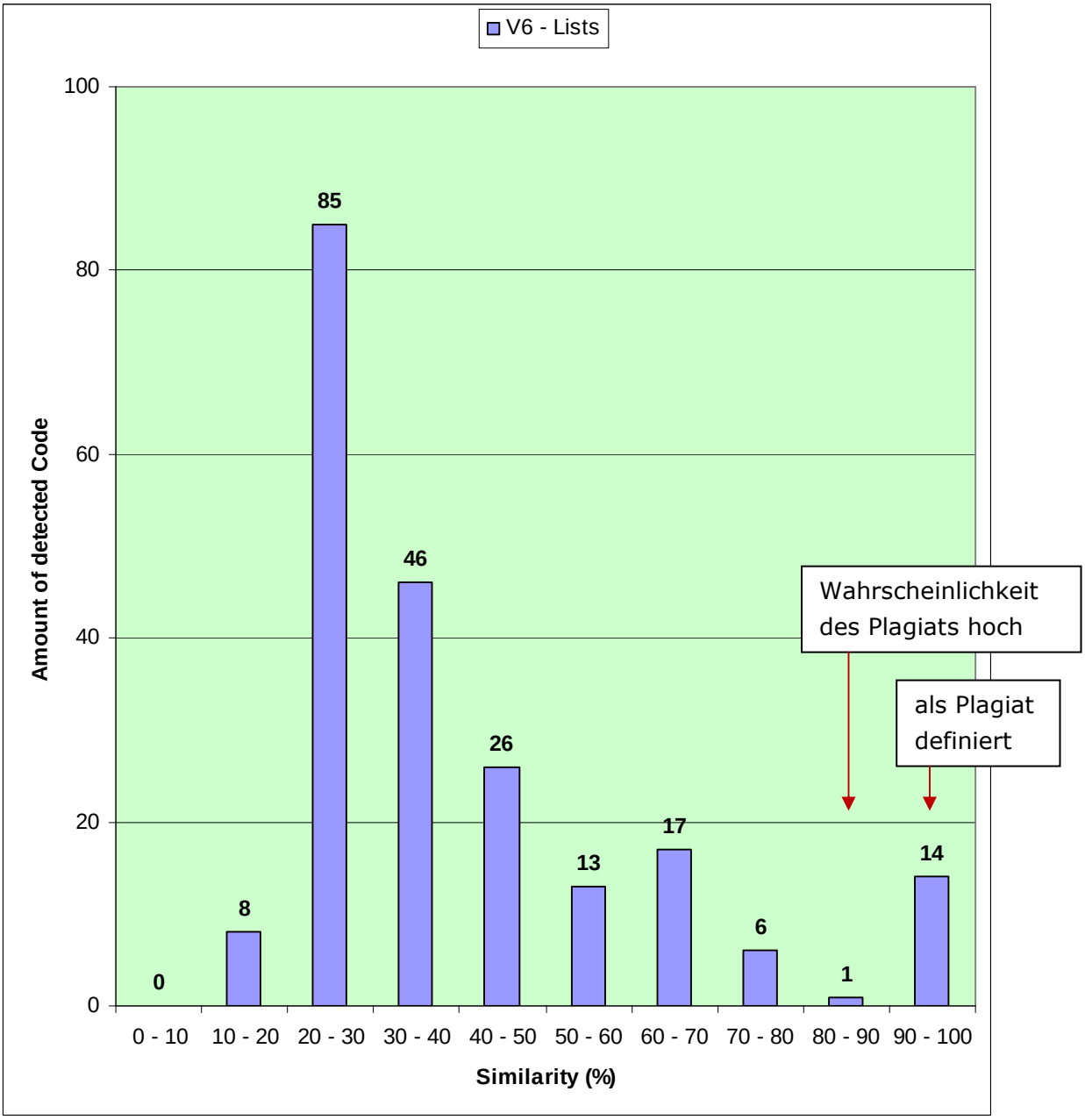

Abb. 8: Ergebnis V6 Measure of Software Similarity

Bei V6 hat MoSS 216 Ergebnisse ausgegeben. So liegen zum Beispiel 85 der 1540 Vergleiche bei 20-30%. Laut MoSS liegen insgesamt 15 Lösungen an oder über dem Schwellenwert von 83%. Damit hat MoSS im Vergleich zu JPlag weniger als die Hälfte an Plagiaten detektiert.

5.3 Versuch V7

Bei dem Versuch V7 geht es um die Erstellung und Verwaltung von Binärbäumen und die Verwaltung von Dateien. Der Schwellenwert wurde auf 90% gesetzt, da die programmatischen Unterschiede als gering anzusehen sind.

Die Abbildungen 9 bis 11 zeigen die Ergebnisse der drei Plagiatserkennungsprogramme auf.

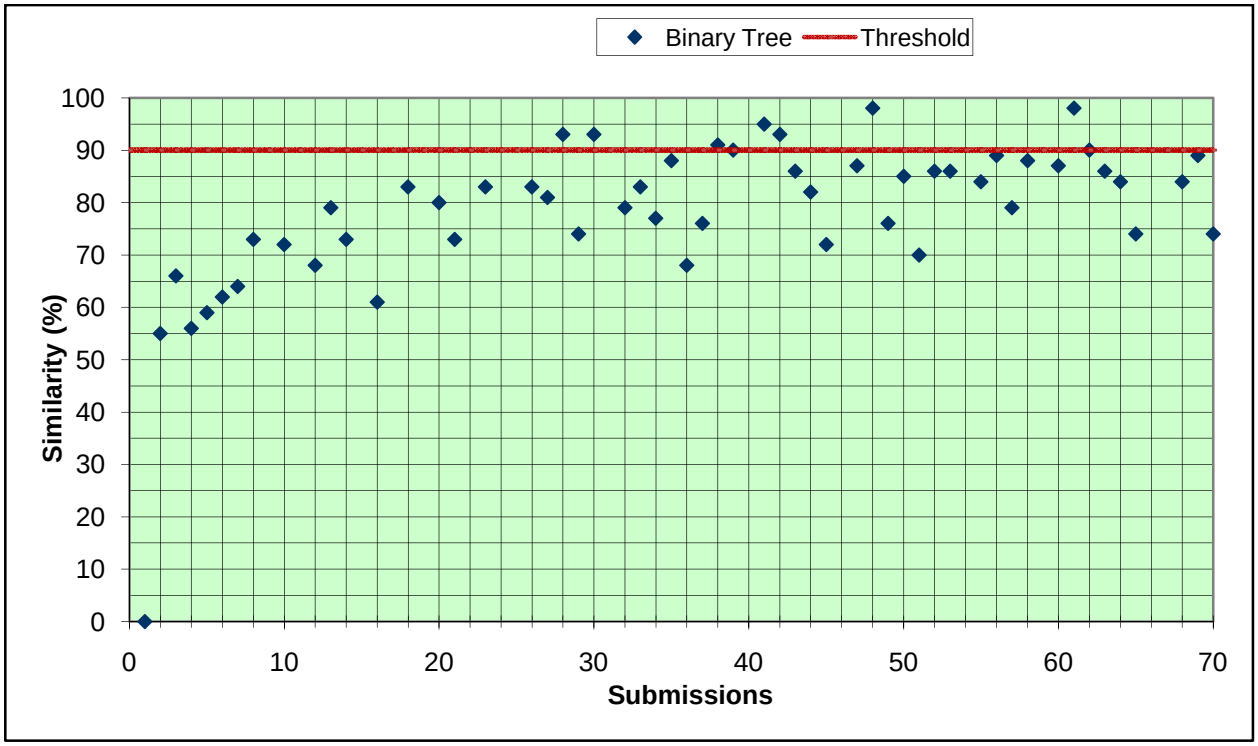

Abb. 9: Ergebnis V7 PlagC2

Laut PlagC2 liegen neun von 56 Lösungen und damit 16,1% an oder oberhalb des Schwellenwertes. Dies ist ein überraschendes Ergebnis, da aufgrund des Ergebnisses bei V6 (30,4%) und der Komplexität der Praktikumsaufgabe ein höheres Ergebnis zu erwarten war.

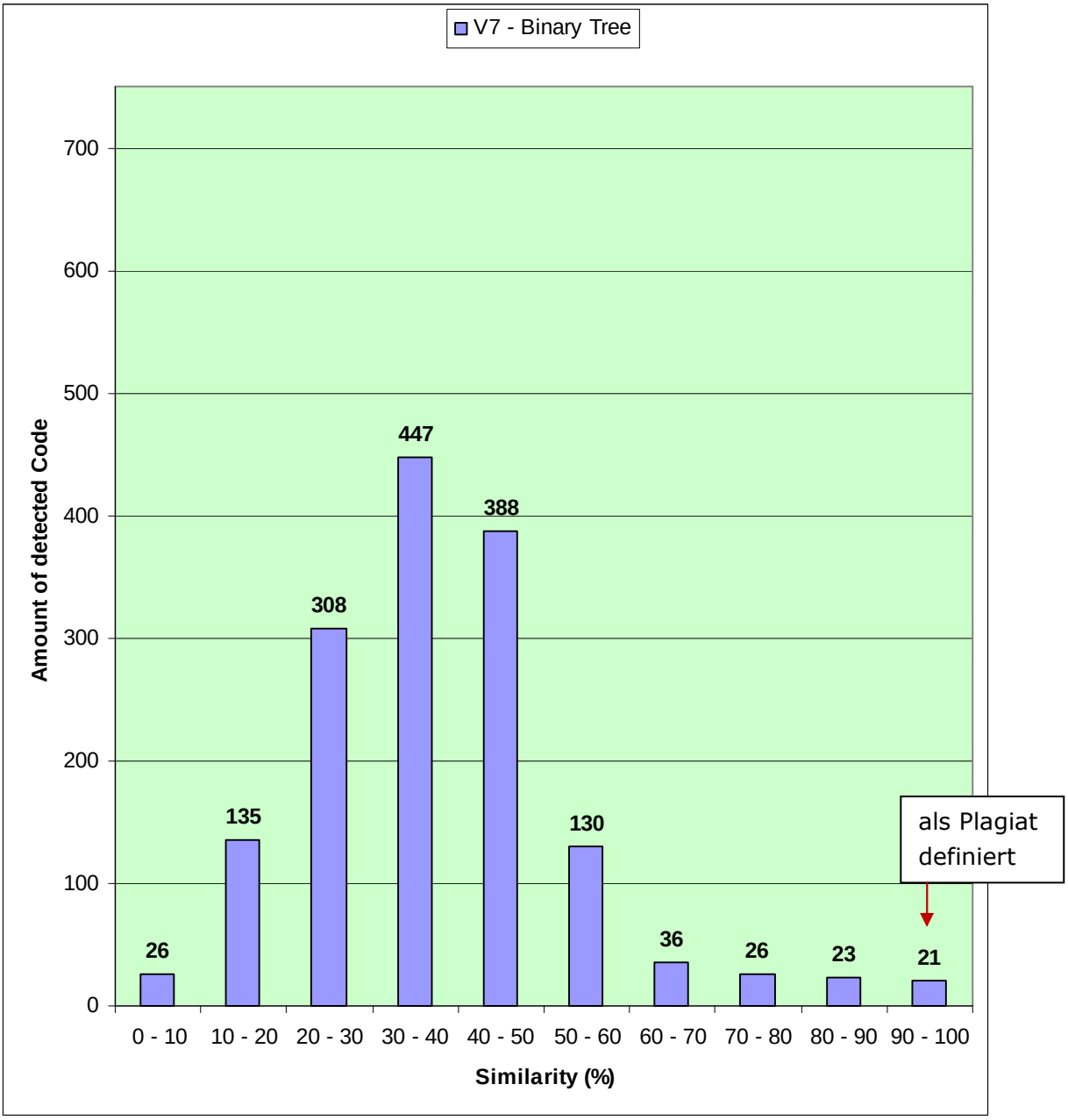

Abb. 10: Ergebnis V7 JPlag

Bei JPlag liegen 21 der 1540 Vergleiche an oder oberhalb des Schwellenwertes von 90%. Auch JPlag hat damit weniger Plagiate bei V7 detektiert als bei V6. Wie bereits bei PlagC2 erwähnt, ist dies unerwartet. Bei genauerer Betrachtung der Ergebnisse (siehe CD: JPlag-Ergebnisse) fällt auf, dass einige Lösungen, die PlagC2 nicht als Plagiat detektiert hat, bei JPlag eine höhere Übereinstimmung erhalten, zum Beispiel Lösung 63 (PlagC2 86%, JPlag 95%). Dieses Ergebnis erfordert eine Überprüfung durch den Anwender.

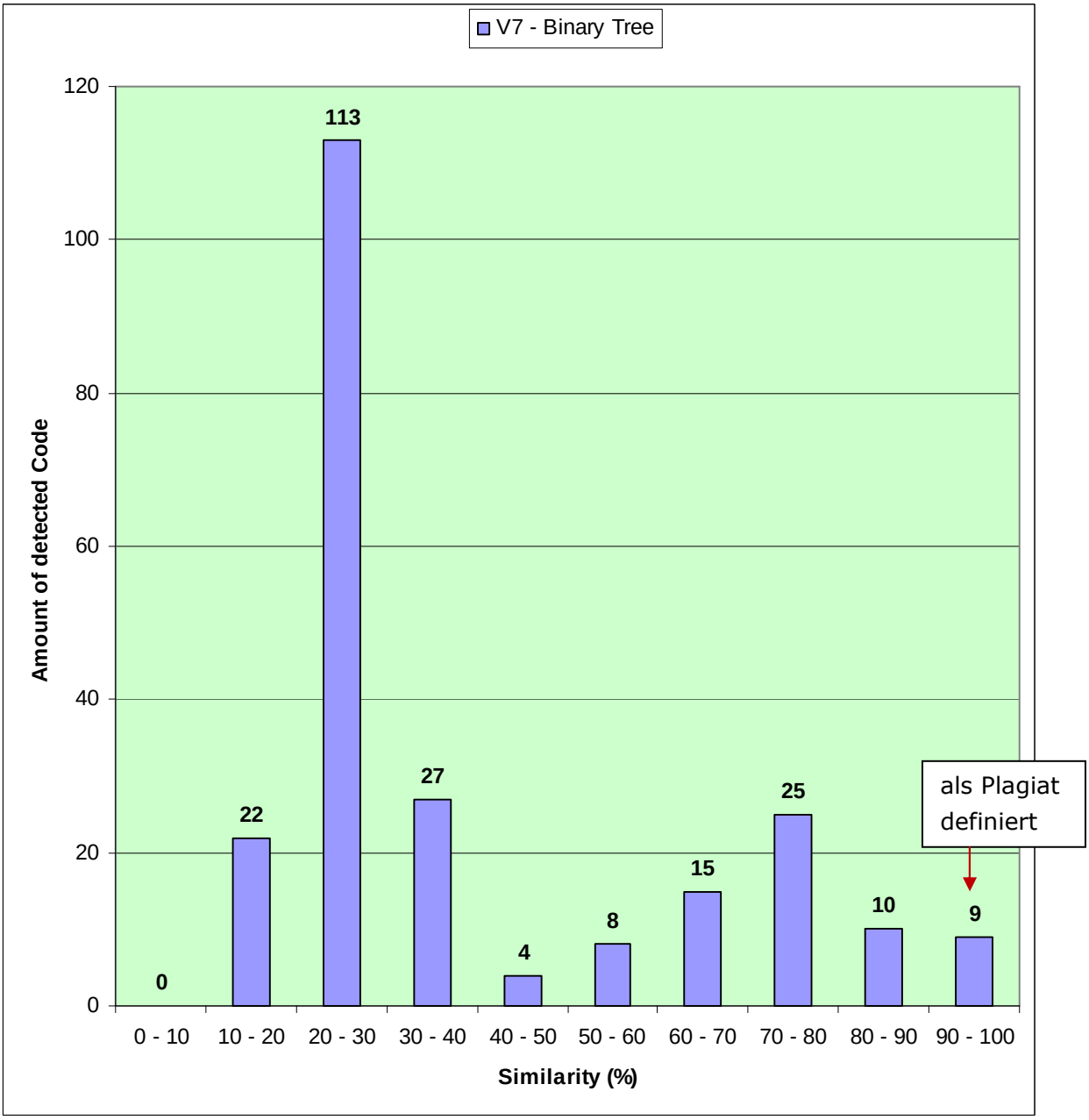

Abb. 11: Ergebnis V7 Measure of Software Similarity

MoSS hat 233 der 1540 Vergleiche dem Nutzer als Ergebnis zur Verfügung gestellt. Aus diesem Ergebnis ist ersichtlich, dass 9 von 1540 Vergleichen an oder oberhalb des Schwellenwertes liegen. Vergleicht man diese Zahl mit dem Ergebnis von JPlag, so fällt auf, dass auch dieses Mal weniger als die Hälfte an Plagiaten detektiert wurden. Diese erneute Diskrepanz lässt sich nicht so ohne weiteres erklären. Eine genauere Betrachtung erfolgt dazu in Kapitel 6.

6 Auswertung

Nachdem im Kapitel 5 die Ergebnisse der einzelnen Suchdurchläufe in Diagrammen dargestellt wurden, werden diese Ergebnisse in diesem Kapitel näher beleuchtet und weitere Aussagen zu Bedienerfreundlichkeit, Güte der Ergebnisse, graphische Aufbereitung und Gesamteindruck gemacht.

Ein direkter Vergleich der Ergebnisse wäre wünschenswert gewesen ist aber nicht möglich, da bei MoSS und JPlag auf die vollständigen Datenbasis kein Zugriff besteht und somit nur die angebotenen Datenpaare einsehbar sind. Bei PlagC2 kann aus den Ergebnissen zwar ausgelesen werden, ob eine Lösung als Plagiat eingestuft werden kann oder nicht, aber es gibt keine Angabe, mit welcher anderen Lösung die Übereinstimmung gefunden wurde. Daher wurden die Ergebnisse in einzelnen Diagrammen und nicht in einem Diagramm dargestellt.

6.1 Measure of Software Similarity

Measure of Software Similarity berücksichtigt in seinem ersten Suchdurchlauf, bei dem die "Fingerprints" ermittelt werden, werden alle Zeichen des Textes berücksichtigt. Das ist sehr gut, da sich eine Kopie nicht nur in dem ausführbaren Programmcode, sondern oft auch in den Kommentaren. Kommentare sind Bestandteile eines Programmcodes, die für die Ausführung des Programmes nicht relevant sind. Allerdings werden Kopien von den Studenten oft nicht gelöscht, sondern auskommentiert. Das heißt, auch wenn der Programmcode sich von dem eines Kommilitonen unterscheidet sind Kommentare ein guter Indikator ob kopiert wurde oder nicht. Allerdings berücksichtigt MoSS im finalen Suchdurchlauf nicht mehr alle Zeichen, sondern nur noch die als "Fingerprints" ausgewählten Zeichenfolgen. Damit können übereinstimmende Zeichenfolgen durch das Raster fallen und nicht erkannt werden. Eine manuelle Überprüfung der angezeigten Ergebnisse hat dies bestätigt. Was der Mensch als Kopie erkennt ist zum Teil von MoSS nicht oder nur teilweise erkannt worden. Trotz allem sind die Ergebnisse von MoSS gut.

Die Nutzung von MoSS ist bereits in Kapitel 3.1 kurz genannt wurden. Nach dem Absenden einer E-Mail erhält man das Perl-Skript per E-Mail zugesandt. Dieses Skript ist unter Linux sofort ausführbar. Allerdings benötigt man, um das Skript auch unter Windows nutzen zu können, Cygwin. Cygwin ermöglicht die Nutzung beziehungsweise Ausführung von Linux-Befehlen in einer Windowsumgebung. Dies ist etwas umständlich, da Cygwin extra installiert werden muss. Das Ausführen des Skriptes erfolgt in einem Kommandofenster (Shell), in der die im Skript erläuterten Befehle eingegeben werden müssen. Um das Skript erfolgreich ausführen zu können, wird zusätzlich ein Internetzugang benötigt, da die Daten über das Internet an den MoSS-Server geschickt und die Antwort via Internet empfangen wird. Außerdem erfolgt die Ausgabe der Ergebnisse ebenfalls via Internet. Damit ist MoSS nicht lokal, offline nutzbar. Dies ist ein Nachteil für dieses Werkzeug. Ebenfalls nicht sehr benutzerfreundlich ist die Ausgabe der Ergebnisse. Auf der Webseite, die vom MoSS-Sever generiert wird, werden zwei Dokumente, in denen Übereinstimmungen gefunden worden, als Paar aufgelistet. Alle Paare werden untereinander aufgelistet. Dabei werden der

Dateiname und die prozentuale Übereinstimmung genannt. Jedes Paar ist mit einer weiteren Webseite verlinkt, auf der die Dateien graphisch aufbereitet sind. Die Dateien werden dabei gegenüber gestellt und die übereinstimmenden Passagen farbig hervorgehoben. Diese graphische Aufbereitung stammt von Guido Malpohl, einem der Entwickler von JPlag. Derweil die Auflistung der Ergebnisse unübersichtlich, nicht sortiert und nicht vollständig ist, ist die graphische Aufbereitung der Dateien eine große Hilfe bei der manuellen Untersuchung potentieller Plagiate.

6.2 JPlag

JPlag berücksichtigt in seinem Algorithmus alle Zeichen. Einzig die sogenannten Whitespaces, also Leerzeichen, Semikola, et cetera werden ignoriert. Damit werden sämtliche in einem Programmtext enthaltene Zeichen beziehungsweise Zeichenfolgen mit denen des gegenübergestellten Programmtextes verglichen. Es ist anzunehmen, dass damit sämtliche Übereinstimmungen gefunden werden. Dies erklärt, warum JPlag zum Teil erheblich mehr Übereinstimmungen zwischen zwei Dokumenten gefunden hat als MoSS. Um dies zu verdeutlichen, sollen drei Paare aus Versuch V1 exemplarisch genannt werden:

- Die Lösungen 22 und 29: JPlag ca. 90% Übereinstimmung
  MoSS ca. 12% Übereinstimmung
- Die Lösungen 11 und 28: JPlag 100% Übereinstimmung
  MoSS 99% Übereinstimmung
- Die Lösungen 16 und 48: JPlag ca. 92% Übereinstimmung
  MoSS ca. 86% Übereinstimmung

Gerade das erste Beispiel zeigt deutlich, welchen Unterschied die Verwendung verschiedener Algorithmen machen kann.

Die Nutzung von JPlag ist in Kapitel 3.2 kurz erläutert wurden. Nachdem man sich erfolgreich bei JPlag mit einem Benutzernamen und Passwort registriert hat, kann man über die Webseite der Universität Karlsruhe auf den Webservice von JPlag zugreifen. Um JPlag nutzen zu können, ist, genau wie bei MoSS, ein Internetzugang erforderlich und JPlag damit nicht lokal, offline nutzbar. Der Webservice ist sehr benutzerfreundlich und selbst erklärend gestaltet. Dateien können über ein Menüfenster, wie man es zum Beispiel von Microsoft Word kennt, ausgewählt werden. Zusätzlich kann der Nutzer bestimmen, wie lange die Zeichenfolge mindestens sein soll, um Übereinstimmungen zu finden. Wenn die Dateien vom JPlag-Server analysiert wurden, wird eine neue Webseite erstellt, mit den Ergebnissen. Die generierten Webseiten mit der Gesamtübersicht, sowie der graphischen Aufbereitung der zur Verfügung gestellten Paare werden zusätzlich lokal auf dem Computer des Nutzers gespeichert. Damit muss der Nutzer sich die Adresse der Webseite nicht merken, sondern kann einfach über den abgespeicherten Link wieder auf die Ergebnisse zugreifen. JPlag hat seine Gesamtübersicht in mehrere Abschnitte unterteilt. Zum einen werden sämtliche eingereichte Dateien aufgelistet und zum anderen wird eine Gesamtansicht gezeigt mit der Verteilung der Übereinstimmungen (siehe Abbildung 12).

Verteilung:

90% - 100%	6	#
80% - 90%	1	#
70% - 80%	5	#
60% - 70%	13	#
50% - 60%	47	####
40% - 50%	175	##################
30% - 40%	549	##
20% - 30%	729	##
10% - 20%	595	##
0% - 10%	226	#######################

Abb. 12: Verteilung der Übereinstimmung im Versuch V1 bei JPlag

Zusätzlich werden Dateipaare sortiert nach mittlerer und maximaler Häufigkeit angezeigt. Allerdings werden nicht alle Paare, die verglichen worden, angezeigt, sondern nur die mit den höchsten Prozentsätzen. Das heißt, Paarungen, die eine Übereinstimmung von weniger als 50% haben, werden von JPlag nicht graphisch aufbereitet. Da für die einzelnen Versuch Schwellenwerte weit über 50% gesetzt worden, ist diese Einschränkung nicht relevant für die Ermittlung von Plagiaten. Wie bereits erwähnt, nutzen MoSS und JPlag die gleiche graphische Darstellung der Dateien (siehe Abbildung 13).

mop13-V7/12/85844/V7/ (62%)	mop13-V7/30/86703/V7/ (62%)
11-68	11-116
181-213	177-224
82-110	228-259
224-242	117-150
113-126	262-275
171-181	320-330
69-79	160-176

```
}
tbinTree* CreateNode(char *name, double telefonnummer){
        tbinTree *root;
        root = malloc(sizeof (tbinTree));
        strcpy(root->Name, name);
        root->Telefonnummer = telefonnummer;
        root->right = NULL;
        root->left = NULL;
        return root;
}
void PutMap(char* name, double telefon){
        if(name == NULL){
                return;

        }
        AddNode(name, telefon);
}

double* GetMap(char* name){
        if(name == NULL || baumwurzel == NULL){
                return NULL;
        }
        tbinTree* treePointer;
        treePointer = baumwurzel;
        double *nummer = SucheName (treePointer, name);
        return nummer;
}

void ReadElement(FILE *f)
{

        char Zeile[FIFTY] = {0};
        char Tel[FIFTY] = {0};
        char Name[FIFTY] = {0};

        int k = 0;

        fgets(&Zeile, FIFTY, f);
        if (Zeile == NULL){
                printf("Einlesen war nicht erfolgreich.");
                return;
        }else{
                for (int i = 0; i <= FIFTY; 0){
```

```
        strcpy(root->Name, name);
        root->Telefonnummer = telefonnummer;
        root->right = NULL; // Nachfolger
        root->left = NULL; // initialisieren
        return root;
}

//Hilfsfunktion NAME einf◆gen
void AddNode(char *name, double telefonnummer)
{

        /* Check if tree is empty */
        if (baumwurzel == NULL)
        {
                baumwurzel = CreateNode(name, telefonnummer);

                return;
        }

        tbinTree* treePtr;
        treePtr = baumwurzel;
        while (1)
        {
                int comp = strcmp(name, treePtr->Name);
                if (comp == 0)
                {
                        printf("Diesen Eintrag gibt es bereits.\n");
                        return; /* string already present */
                }
                if (comp < 0)
                { /* move left */
                        if (treePtr->left != NULL)
                        {
                                treePtr = treePtr->left;
                        }
                        else
                        {
                                treePtr->left = CreateNode(name, telefonnumme
                                return;
                        }
                }
                else
                { /* move right */
```

Abb. 13: Screenshot Darstellung der Übereinstimmung in zwei Programmtexten

Dies ermöglicht es, wenn MoSS und JPlag die gleichen Paare gefunden haben, die Gegenüberstellungen für beide Werkzeuge parallel zu öffnen und einen direkten Vergleich zu ziehen, zwischen den Übereinstimmungen und Unterschieden zwischen MoSS und JPlag. Insgesamt liefert JPlag sehr gute Ergebnisse.

6.3 PlagC2

PlagC2 berücksichtigt nicht alle Zeichen in einem Programmtext. Nicht nur die Whitespaces werden ignoriert, sondern auch sämtliche Kommentare. Da PlagC2 denselben Algorithmus wie JPlag, Greedy-String-Tiling-Algorithmus, verwendet, unterscheiden sich die Ergebnisse an diesem Punkt. Aufgrund dessen, dass PlagC2 die Kommentare nicht mit einbezieht, wird häufig eine etwas geringe prozentuale Übereinstimmung gegenüber JPlag gefunden. Zum Beispiel liegt im Versuch V1 die Übereinstimmung der Lösung 28 mit einer weiteren Lösung laut JPlag bei 100% und laut PlagC2 bei 94%. Bei dem Versuch V6 ist eine Anomalie aufgetreten, die nicht erklärbar ist. Da die Praktikumsaufgaben auf eine begrenzte Anzahl von Art und Weise gelöst werden können und zum Beispiel if-else-Anweisungen oder eine for-Schleife eine typischer Bestandteil für Programmtexte sind, ist es unerwartet, dass die Lösung 26 0% Ähnlichkeit mit anderen eingereichten Lösungen aufweist. Da ein Programmtext vorhanden ist und es sich nicht um eine leere Datei handelt, verlangt dieses Ergebnis eine manuelle Überprüfung durch einen Betreuer. Desweiteren birgt die sukzessive Durchsuchung der Lösungen die Gefahr in sich, Ergebnisse zu verfälschen. Es wird davon ausgegangen, dass bei einer Übereinstimmung zweier Programmtexte die später eingereichte Lösung das Plagiat ist. Das muss nicht sein, da es darauf ankommt, wie schnell ein Student seine Lösung beim Betreuer einreicht. Ist der Student, der abgeschrieben hat schneller, als der Student, von dem abgeschrieben wurde, ergibt das ein falsches Ergebnis in der Beurteilung des Suchdurchlaufs.

Zu der Benutzerfreundlichkeit von PlagC2 kann keine Aussage getroffen werden, da die Ergebnisse zum Vergleich vorlagen und das Programm nicht getestet werden konnte. Allerdings ist bekannt, dass eine graphische Darstellung von Datei-Paaren, wie bei MoSS und JPlag derzeit nicht gegeben ist. Im Gegensatz zu den beiden anderen Werkzeugen kann PlagC2 lokal, offline verwendet werden. Außerdem liefert PlagC2 sehr gute Gesamtergebnisse. Im Gegensatz zu den anderen Plagiatserkennungsprogrammen kann bei PlagC2 der Schwellenwert angegeben werden. Wenn eine Datei nach ihrer Überprüfung diesen Schwellenwert erreicht oder überschritten hat, wird sie als Plagiat gekennzeichnet und in der Datenbank separat abgelegt. Danach wird diese Lösung nicht mehr beim Vergleich mit weiteren eingereichten Lösungen betrachtet. Der Betreuer hat dann die Möglichkeit sich die betroffene Lösung und das dazugehörige Pendant anzusehen und zu entscheiden, ob es sich tatsächlich um ein Plagiat handelt oder nicht.

6.4 Gegenüberstellung

	MoSS	JPlag	PlagC2
Internetzugang erforderlich	Ja	Ja	Nein
Graphische Aufbereitung der Datei-Paare	Ja	Ja	Nein
Güte der Ergebnisse	Gut	Sehr gut	Sehr gut
Benutzerfreund-lichkeit	Ok	Sehr gut	Keine Angabe

Tabelle 2: Übersicht über die Werkzeuge

7 Fazit und Ausblick

Ziel dieser Bachelorarbeit war es, die an der UniBwM geschriebene Plagiatserkennungssoftware PlagC2 mit den via Internet verfügbaren, etablierten Werkzeugen zur automatisieren Plagiatsprüfung Measure of Software Similarity und JPlag zu vergleichen. Dabei wurde festgestellt, dass die Werkzeuge unterschiedliche Suchalgorithmen verwenden, die Einfluss auf die Güte der Suchergebnisse haben. MoSS verwendet den Winnowing-Algorithmus, wogegen sowohl JPlag als auch PlagC2 auf den Greedy-String-Tiling-Algorithmus zurückgreifen. Der GSTA hat sich in dieser Bachelorarbeit als der effektivere Algorithmus herausgestellt, da alle Zeichen bei der Suche berücksichtigt werden. Beim WA werden zunächst alle Zeichen berücksichtigt, aber zum Vergleich nur noch ausgewählte Zeichenfolgen betrachtet. Damit gehen Daten bei der Detektion von Plagiaten verloren, was sich in den Ergebnissen, die MoSS ausgegeben hat, in Form von niedrigerer Übereinstimmung widerspiegelt. Die Suchergebnisse sind bei den drei Werkzeugen dabei gut bis sehr gut einzuschätzen.

PlagC2 verarbeitet die Lösungen der Studenten sukzessiv. Es ist anzuraten, dies zu ändern und genau wie bei MoSS und JPlag eine Totalsuche durchzuführen. Das heißt, erst alle Lösungen zu sammeln und dann alle Programmtexte mit einander gleichzeitig zu vergleichen. Um am Ende des MOP-Praktikums eine schnelle Übersicht zu erhalten, welchem Studenten bei welcher Praktikumsaufgabe ein Plagiat nachgewiesen wurde, sollte dies in einer Datenbank abgespeichert werden. Außerdem hat sich die graphische Aufbereitung der gefundenen Daten-Paare bei JPlag als sehr hilfreich erwiesen. Durch die farbliche Hervorhebung der Übereinstimmungen in zwei Programmtexten, ist es für den Anwender einfacher, die Programmtexte selbst noch einmal zu kontrollieren. In PlagC2 ist das bisher noch nicht implementiert. Da aber gerade bei den Programmtexten, die an der Grenze des jeweils gesetzten Schwellwertes liegen, eine Nachkontrolle empfohlen wird, ist zu dieser Darstellung zu raten. Desweiteren sollten zwei Schwellwerte eingeführt werden. Der obere Schwellwert sollte, so wie bisher, der Wert sein, ab dem ein Programmtext als Plagiat definiert wird. Der zweite, untere Schwellwert sollte zum Beispiel 5% unter dem oberen Schwellwert liegen. Alle Programmtexte die in diesen Bereich fallen, müssen vom Betreuer gesichtet und die Studenten dazu herangezogen werden. Mit wenigen Fragen kann leicht festgestellt werden, ob zufällige Übereinstimmungen vorliegen und der Student den Programmtext selbstständig erstellt hat, oder ob, bei mangelhaften Kenntnissen des Studenten, ein Plagiat vorliegt. Es gibt den Betreuern im MOP-Praktikum einen gewissen Spielraum, um die eingereichten Lösungen zu beurteilen. Ferner ignoriert PlagC2 derzeit die Kommentare in den Programmtexten, wodurch Übereinstimmungen zwischen mehreren Lösungen verloren gehen können. In der Zukunft sollte dies geändert werden, damit die Ergebnisse der Plagiatsprüfung verbessert werden.

Insgesamt sind die drei Werkzeuge gut geeignet, um Plagiate in Programmtexten aufzudecken. Es ist dabei eine Frage des persönlichen Geschmacks, mit welchem Werkzeug ein Anwender am besten umgehen kann und welche Darstellung er bevorzugt. Wenn man für eine Plagiatsprüfung keinen Internetzugang hat, ist allerdings nur PlagC2 verwendbar, da sowohl MoSS als auch JPlag ohne Internetzugang nicht verwendet werden können.

Trotz der guten Werkzeuge zur automatisierten Plagiatsprüfung von Programmtexten ist eine abschließende Überprüfung des Ergebnisses durch den Anwender unabdingbar. Die Maschine ist nicht in der Lage eine Bewertung vorzunehmen und auch wenn zwei Programmtexte zu zum Beispiel 65% übereinstimmen, bedeutet dies nicht, dass es sich um Plagiate handelt. Gerade im Rahmen eines Praktikums wie dem MOP-Praktikum sind die Variationen des Programmierens beschränkt und es ist verständlich, dass Konzepte in mehreren Arbeiten auftauchen. Die Werkzeuge können als große Stütze für die Detektion von Plagiaten angesehen werden, da sie den Zeitaufwand, den eine manuelle Prüfung erfordert, drastisch reduzieren. Außerdem erkennt eine Software alle Übereinstimmungen, während der Anwender, vor allem wenn es sich um einen umfangreicheren Text handelt, den Überblick verlieren kann. Auch hier hat die Software einen Vorteil gegenüber dem Anwender und kann ihm eine große Hilfe sein.

Es ist zu erwarten, dass der Einsatz von Plagiatserkennungsprogrammen langfristig Einfluss auf die Arbeitsweise der Studenten hat und das Plagiieren zurück geht, denn wenn ein Student, um Zeit zu sparen, Programmcode von einem Kommilitonen kopiert und dies bei der Plagiatsprüfung entdeckt wird, muss der Student die Arbeit nochmal machen. Das heißt, die Zeit, die er zunächst gespart hat, muss er am Ende zusetzen. Dies ist kontraproduktiv für den Studenten.

Quellenverzeichnis

[1] Lutz Prechelt, „JPlag: Finding plagiarism among a set of programs“, Universität Karlsruhe, 28. März 2000, http://page.mi.fu-berlin.de/prechelt/Biblio/jplagTR.pdf (Zugriff: 30.10.2013)

[2] Christoph Engelhardt, „Realisierung und Evaluation einer Software zur Plagiatsprüfung von Programmquelltexten“, Universität der Bundeswehr München, 14. Juni 2012

[3] Dieter Pawelczak, „Online detection of source-code plagiarism in undergraduate programming courses“, *The 9th International Conference on Frontiers in Education: Computer Science and Computer Engineering*, July 22-25, 2013, Las Vegas, USA, S.57-63

[4] Rabin-Karp-Algorithmus, http://de.wikipedia.org/wiki/Rabin-Karp-Algorithmus (Zugriff: 20.11.2013)

[5] Michael O. Rabin, Richard M. Karp, „Efficient randomized pattern-matching algorithms”, *IBM Journal of Research and Development 31 (2)*, S. 249-260, 2. März 1987

[6] S. Schleimer, D. S. Wilkerson, A. Aiken, „Winnowing: Local Algorithms for Document Fingerprinting“, *SIGMOD 2003*, June 9-12, 2003, San Diego, USA

[7] Michael J. Wise, "String Similarity via Greedy String Tiling and Running Karp–Rabin Matching", University of Sydney, Australia, Dezember 1993

[8] Modulhandbuch des Studiengangs Wehrtechnik an der Universität der Bundeswehr München, PO-Version 2011

[9] Plagiatsaffäre Guttenberg, http://de.wikipedia.org/wiki/Guttenberg-Affäre (Zugriff: 05.11.2013)

[10] „Duden – das Fremdwörterbuch“, 9. Auflage, 2007

Anhang

Sämtliche im Quellenverzeichnis angegebenen Quellen sind, als pdf-Dateien auf der beigefügten CD enthalten.

Webausgaben JPlag und MoSS

Die Ergebnisse von JPlag und MoSS worden in pdf-Dateien umgewandelt und sind auf der CD enthalten.

Exceltabellen

Die mit Excel aufbereiteten Ergebnisse sind ebenfalls auf der CD enthalten.

JPlag

Versuch 1	
Anzahl Programme	69
Verteilung %	Übereinstimmungen
0 - 10	226
10 - 20	595
20 - 30	729
30 - 40	549
40 - 50	175
50 - 60	47
60 - 70	13
70 - 80	5
80 - 90	1
90 - 100	6

Versuch 6	
Anzahl Programme	56
Verteilung %	Übereinstimmungen
0 - 10	6
10 - 20	67
20 - 30	241
30 - 40	387
40 - 50	423
50 - 60	230
60 - 70	114
70 - 80	39
80 - 90	14
90 - 100	19

Versuch 7	
Anzahl Programme	56
Verteilung %	Übereinstimmungen
0 - 10	26
10 - 20	135
20 - 30	308
30 - 40	447
40 - 50	388
50 - 60	130
60 - 70	36
70 - 80	26
80 - 90	23
90 - 100	21

PlagC2

Versuch V1

Lösung	Übereinstimmung in %
1	0
2	29
3	54
4	37
5	53
6	84
7	46
8	56
9	43
10	47
11	40
12	60
13	52
14	50
15	46
16	52
17	39
18	50
19	65
20	49
21	54
22	83
23	45
24	60
25	46
26	55
27	60
28	94
29	77
30	53
31	60
32	64
33	57
34	47
35	71
36	60
37	48
38	54
39	63
41	60
42	85
43	57
44	66
45	48
47	79
48	83
49	55
50	41
51	57
52	50
53	61
54	57
55	84
56	57
57	66
58	65
59	63
60	57
61	39
62	58
63	91
64	62
65	66
66	47
67	53
68	61
69	56
70	51

Versuch V6

Lösung	Übereinstimmung in %
1	0
2	58
3	60
4	97
5	70
6	82
7	80
8	67
10	67
12	77
13	73
14	76
16	74
18	80
20	81
21	74
23	76
26	0
27	80
28	100
29	77
30	74
32	80
33	82
34	85
35	81
36	82
37	81
38	81
39	89
41	89
42	92
43	76
44	90
45	77
47	83
48	88
49	85
50	82
51	75
52	81
53	82
55	81
56	82
57	77
58	96
60	80
61	96
62	85
63	96
64	96
65	92
68	81
69	80
70	77

Versuch V7

Lösung	Übereinstimmung in %
1	0
2	55
3	66
4	56
5	59
6	62
7	64
8	73
10	72
12	68
13	79
14	73
16	61
18	83
20	80
21	73
23	83
26	83
27	81
28	93
29	74
30	93
32	79
33	83
34	77
35	88
36	68
37	76
38	91
39	90
41	95
42	93
43	86
44	82
45	72
47	87
48	98
49	76
50	85
51	70
52	86
53	86
55	84
56	89
57	79
58	88
60	87
61	98
62	90
63	86
64	84
65	74
68	84
69	89
70	74

Measure of Software Similarity

Versuch 1	
Anzahl Programme	69
Verteilung %	Übereinstimmungen
0 - 10	26
10 - 20	54
20 - 30	5
30 - 40	2
40 - 50	2
50 - 60	3
60 - 70	1
70 - 80	4
80 - 90	1
90 - 100	3

Versuch 6	
Anzahl Programme	56
Verteilung %	Übereinstimmungen
0 - 10	0
10 - 20	8
20 - 30	85
30 - 40	46
40 - 50	26
50 - 60	13
60 - 70	17
70 - 80	6
80 - 90	1
90 - 100	14

Versuch 7	
Anzahl Programme	56
Verteilung %	Übereinstimmungen
0 - 10	0
10 - 20	22
20 - 30	113
30 - 40	27
40 - 50	4
50 - 60	8
60 - 70	15
70 - 80	25
80 - 90	10
90 - 100	9