Chapter 7

The POSAR Protocol for Software Copyright Infringement Forensics

A quest for modification for the AFC protocol in the form of an enhanced variation is the main objective of this chapter. As mentioned in chapter 3, in the context of the technology available in the early 1990s, the IT world would not expect anything more than Abstraction-Filtration-Comparison (AFC) test for software copyright infringement forensics. But today’s software world features users and developers are more intelligent, well-informed, techno-savvy and better-equipped. Above all these, criminals are also expanding their horizons into the software space. This being so, it is the duty of computer experts to update the software forensic tools, processes and procedures so as to make them fit for performing the forensics of modern software products rather than letting legal professionals lie back on the legacy of static forensic tools like AFC. When forensic tools are old and outdated, the results of forensic analysis might be incomplete, inadequate and possibly inaccurate. Cyber forensic experts should remember that incomplete or improper cyber forensic reporting can lead to unreliability of evidence and hence can be challenged in the court of law. In such cases, during subsequent cross examinations in the court of law, the lawyers may well prove that there is unreliability of evidence at least enough to claim ‘reasonable doubt’ in the allegation or that there was still scope for further in-depth analysis. Obviously, events like this might cause inconveniences to the jury which can further lead to unintended delay or miscarriage of justice. Hence, the first report of the cyber forensic expert itself should wherever possible be thorough, convincing and binding. In such a reporting process, for the forensic tools, processes and procedures to play their important role, they need to be updated to the level of the best available development. The new protocol, the POSAR test, is the result of an attempt to do precisely this on AFC test. POSAR (which stands for Planning-Operationalization-Separation-Analysis-Reporting) is an extension of AFC and thus, does not discount AFC.
7.1 Steps to formulate a new protocol (POSAR) out of AFC

This study assumes that as a forensic tool AFC is also in a ready state to be re-designed to make it fit for forensics of state-of-the-art software products. The fact that AFC has been used to establish culpability in several software copyright infringement cases in the US cannot be reason enough to let AFC remain complacent and non-updated.

As said earlier, the contributions to the growth and development of AFC seem to be more in the legal domain, and even jurors, while accepting judgments based on AFC evidence, seemed to be incorporating added legal dimensions to its functionality (Gable, 1998; USDCM, 2010), which somehow relegates the need for its technical development as secondary and less important. It is unfair to get legal functionality and popularity of use get in the way of the technical development and efficiency of any system. That seems to be happening with AFC. It will be both morally and judicially unfair to base modern software piracy decisions mainly on an approach that has failed to incorporate technical developments as it might create a stagnant and moribund system based only on legal functionality and popularity. When technology is applied in any area, the developments in technology as well as in the applied field should constantly and constructively feed into each other. Thus a legal system that bases its reliance more on its own tradition and legacy rather than developmental input from the technology cannot be desirable. Such system will also run the risk of self perpetuation as it may become insecure about its own non receptiveness to technical developments. Therefore not only is it necessary to look into the development of technology but also to ensure that new developments in software engineering are also successfully incorporated into the legal system. To perform this, AFC needs to be modified and the prime requirement towards these modifications appears to be the incorporation of some more features into AFC in order to make AFC, state-of-the-art. The steps followed to formulate a new protocol (POSAR) to enhance AFC can be formulated in the following form.
i. Identifying the positive aspects of AFC which need to be retained in POSAR

ii. Re-considering those aspects of AFC which need to be modified while formulating POSAR

iii. Creating an enhanced list of features along which the final comparison needs to be performed

iv. Defining the different phases in POSAR

v. Creating an algorithmic process for POSAR

### 7.2 AFC features retained in POSAR

The aspects of AFC that can be considered to be retained in POSAR are; (1) removal of all globally common and public domain materials from the abstracted software (2) removal of all “merged” ideas and expressions and (3) removal of codes / expressions which are unavoidable in the software due to external demands. However, these removals could do with some additional explanations or instructions to properly carry them out, and these tasks will be looked into later here. In short, the entire procedure of the filtration stage of AFC (the entire PHASE-II in fig 3.1) can be brought down to POSAR and these considerable aspects of AFC can be directly brought down to POSAR’s filtration stage, with some enhancements (see section 7.4.3).

### 7.3 AFC features to be modified for POSAR

POSAR also needs to seriously deal with those aspects of AFC that in their present form are not fully efficient or productive. These aspects of AFC, which have already been discussed in chapter 4 and chapter 5, can be recalled and summarized as follows. The existing form of AFC has the following deficiencies.

7.3.1. The procedure of AFC is currently linear-sequential and thus, it does not consider the very basic cyclical demands of software piracy forensics (See 7.4 for more details).
7.3.2. The current procedure of AFC does not ensure that the essential post-piracy modifications are retained in order that their contribution towards valuable evidence to establish piracy is accounted for (See 7.5 for more details).

7.3.3. AFC’s “doctrine” exceptions do not ensure that they deal with the design patterns and programming patterns as well (See 7.6 for more details).

7.3.4. AFC does not consider evidence external to the software in hand.

7.3.5. AFC does not have a final stage that stresses the need for presenting the results of the whole investigation in a judiciary-friendly, less-technical, jargon-free, and non-esoteric format (preferably in numerical terms) (See 7.8 for more details).

POSAR thus needs to be an enhanced version that would remedy the shortcomings. These enhancement procedures are explored in greater detail below.

7.3.1 **Incorporating cyclicality**

Cyclicality as a factor is generally absent in existing software piracy forensic methods. However, cyclicality is already present in software development and any software is developed through a process (or using a model) that is very much cyclical or spiral (or at least not linear-sequential) by nature and such software development models are well established and well used, internationally. So it seems most logical that the field of software piracy forensics needs to draw upon this existing model.

Generally, one of the motivations for introducing cyclicality in a multi-stage project management tool is to provide a way to handle errors during project development. To be precise, in case an error that unfortunately had occurred in, say, stage-1 was undetected until stage-3 of the project, the project management tool should facilitate a way to return to stage-1 for necessary correction and resulting modification retrospectively all the way back to stage-3. Software project development management had successfully handled such a situation by introducing cyclicality (Davis et. al. 1988; Pressman, 2001, p.34-47, p.603-627, p. 671-714) way back in 1980s. Further, by introducing cyclicality to software development process, software industry has gained much. Firstly, the cyclic nature of the
process helps the developer to return to any previous stage for correcting errors and then to proceed from there sequentially down by incorporating the corresponding corrections in the subsequent stages; Secondly, cyclicality helps the manager to manage a software project easily; Thirdly, cyclicality has lead to software development becoming a well co-ordinated team event than an individual’s game; Fourthly, cyclicality brings more controls into the development process in order ensure the ultimate integrity and quality of the product; And finally, the evolutionary cyclic model or the spiral model has provided a way to properly manage the risk factor in the software development process. All the above features of cyclicality of the software development process can be of help in software forensics process too, if cyclicality is introduced in this forensic process.

Further, AFC test, with its current linear-sequential process, is more of a single individual effort and the judiciary may question the credibility of a single person handling the forensics of large software systems (which can run into millions of codes spanning hundreds of files and databases). It will therefore be wise for the protocol to be a well co-ordinated team event. The forensics management would then be more controllable, the related risk-management easy, and ultimately, the integrity and quality of the forensics result more reliable.

All the above suggest that a cyclic model (just as in software development process) is more desirable for POSAR (than a linear sequential model), and that all the related success stories (see Pressman, 2001, p.34-47, p.603-627, p. 671-714) of software development process can be brought down to POSAR as well.

7.3.2 Incorporating the investigation of post-piracy modifications

The impact of post-piracy modifications in software piracy forensics (see Baboo and Bhattathiripad, 2010) has already been discussed (see chapter 4 above). Due to post piracy modifications, questionable differences from the original can be found in many parts of the pirated, as in source codes, databases, embedded images, fingerprints and so on. A proper study of these questionable differences will contribute substantially to the reliability of software piracy forensic investigation. In order to sieve out post piracy modifications in the pirated, the researcher needs to compare each of the above parts of
the original and the pirated and create a list of questionable differences due to post-piracy modifications. The current procedure of AFC generally filters out and thus discounts the essential post-piracy modifications. Thus, the active contribution of post-piracy modifications towards valuable evidence to establish piracy is not accounted for in AFC. This needs to be changed. While filtering out and discounting non-copyrightable items from the original and pirated, all suspected post-piracy modifications need to be separated out and thus made available for final analysis and comparison (See Operationalization and Separation phases, below).

7.3.3 Incorporating the elements of design and programming patterns

As mentioned in chapter 5, design patterns and programming patterns need to be explicitly made part of general software piracy forensics (Baboo and Bhattathiripad, 2009). Even though both these presumably come under some of the “doctrine” exceptions in AFC, to date it does not seem to have been explicitly examined even by AFC researchers. Moreover, AFC does not properly explain, how the increased complexity of analyzing a program that is written by one programmer and updated at various instances later, by other programmers (that means, increased complexity due to multiple thumb impressions), is dealt with. As far as program logic is concerned, most programs are very likely to have the thumb impressions of many programmers due to the frequent turnover of employees in the software industry and these multiple thumb impressions, compatibly identified in both the ‘original’ and the ‘pirated’ sources, can prove to be vital in establishing piracy.

Design and programming patterns are generally visible in program data structures, functions, data bases, and data base procedures, along the parameters like; (1) Nomenclature (names given to functions, program variables, loop variables, parameters, macros, ‘included’ objects, databases, fields, tables, etc.); (2) database field properties; (3) Programming Logic; (4) Programming remark; (5) Programming blunders; (6) Programming errors; (7) Finger prints; (8) Included objects; and (9) Sequence of appearance of similar-looking items. Forensic investigation needs to be carried out in
POSAR along each of these parameters with the objective of finding out potential evidence for piracy. This means that forensics steps need to be designed and incorporated in POSAR to investigate items like the occurrence of programming blunders in both programs in identical contexts. For this purpose, in POSAR, the original and the pirated need to be analysed with the objective of identifying, say, a programming blunder in the original and then, establishing its presence verbatim in an identical context in the pirated. Such an analysis of the original as well as the pirated need to be carried out along every instance of each of the 9 parameters mentioned above. Moreover, such a forensic analysis is inevitable in software piracy forensics because otherwise, evidence concerning all these parameters would not be available for final comparison. (Absence of such a forensic step is one of the basic fallibilities of AFC). In short, POSAR needs to be designed in such a way that instances of occurrences of design and programming patterns verbatim in original and pirated are properly made available for final comparison.

7.3.4 Incorporating external evidence

Some of the pieces of external evidence that can become potential and tangible evidence for software piracy are; (1) sundry complainant-specific data found in the seized computer system of the alleged pirate (that means, materials irrelevant for the functioning of the pirated); (2) any proof from the DNS server of the alleged pirate’s website; (3) systems analysis or business reports (documents that initiate the design and development of the software); (4) log books (or documents for the software modifications done); (5) government directives (or documents initiating modifications in the software). Not only the AFC test but also all the existing and established approaches and products would fail when the scope of investigation of the software piracy suit is expanded to include the above, that is, beyond the comparing of just the two software packages. POSAR needs to forensically analyse these potential external evidence and bring them to the notice of the court. However, this is a difficult task and cannot be easily incorporated into POSAR.
7.3.5 **Incorporating reporting formats**

AFC does not offer any standard format, let alone judiciary-friendly, for presenting the results of comparison even as the US judiciary seems to have been legally interpreting the technical results complacently for nearly two decades. Any judiciary system with a sense of responsibility and accountability would like to ensure that they are properly informed in matters that are outside their area of expertise. This makes it incumbent on computer experts not only to provide the judicial system with such expertise but also to create general awareness for the proper utilization of such expertise. In actual terms, what the court might expect from a technical expert would be an informative report that is self-explanatory, jargon-free and non-esoteric in format. Thus, the format of expert’s report becomes a matter of prime importance. The addition of a totally new stage that will bring about the presenting of the results of the whole investigation in a judiciary-friendly, less-technical, jargon-less, and non-esoteric format that explain similarity, preferably in numerical terms is not just desirable but necessary.

7.4 **The 3-stage AFC versus 5-stage POSAR**

The new enhanced protocol for AFC will have five phases instead of the three steps in AFC. They are: (1) Planning phase; (2) Operationalization phase; (3) Separation phase; (4) Analysis phase and; (5) Reporting phase. This new protocol (POSAR) will reformulate the software piracy forensic (copyright infringement investigation) procedure from the old 3-stage AFC to a 5-stage process. The details of POSAR are shown diagrammatically below in Fig. 7.1.
Fig 7.1 Process flow charts of POSAR forensic cycle as against AFC’s linear sequential process
7.4.1 The Planning phase of POSAR

The planning phase is an enhancement of the Abstraction step of AFC. One of the motivations for the present enhancement has its root in the US judiciary’s tenth circuit 1993 elaboration on the Computer Associates v Altai,\(^1\) case, where the Abstraction step of AFC (in fact, the AFC itself) was first introduced and applied. The tenth circuit’s elaboration on AFC’s abstraction process is reported by Hollar (2002). “In Gates Rubber v Bando Chemical,\(^2\) a case decided following Computer Associates v Altai, the Tenth Circuit added a little meat to the sparse description from the Second Circuit on how to abstract a program, taking much of its description of the possible levels of abstraction from a law review article written by a computer scientist and law student\(^3\) (Hollar, 2002): The judgment of Gates Rubber v Bando Chemical, as cited in Hollar (2002) is reproduced below and this forms the root of POSAR.

Application of the abstractions test will necessarily vary from case-to-case and program-to-program. Given the complexity and ever-changing nature of computer technology, we decline to set forth any strict methodology for the abstraction of computer programs. Indeed, in most cases we foresee that the use of experts will provide substantial guidance to the court in applying an abstraction test. However, a computer program can often be parsed into at least six levels of generally declining abstraction. (i) the main purpose; (ii) the program structure or architecture; (iii) modules; (iv) algorithms and data structures; (v) source code; and (vi) object code.

The main purpose of a program is a description of the program function or what it is intended to do. When defining a program’s main purpose, the court must take care to describe the program’s function as specifically as possible without reference to the technical aspects of the program.

The program’s architecture or structure is a description of how the program operates in terms of its various functions, which are performed by discrete modules, and how each of these modules interact with each other. The architecture or structure of a program is often reduced to a flowchart, which a programmer uses visually to depict the inner workings of a program. Structure exists at nearly every level of a program and can be conceived of as including control flow, data flow, and substructure or nesting. Control flow is the sequence in which the modules perform their respective tasks. Data flow describes the movement of information through the program and the sequence with which it is

\(^1\) 982 F.2d 693, 23 USPQ2d 1241 (2\(^{nd}\) Circuit, 1992)

\(^2\) 9 F.3d 823, 28 USPQ2d 1503 (10\(^{th}\) Circuit, 1993)

\(^3\) John W. L. Ogilvie, Defining Computer Program Parts Under Learned Hand’s Abstraction Test in Software Copyright Infringement Cases, 91 Mich. L. Rev. 526 (1992)
operated on by the modules. Substructure or nesting describes the inner structure of a module whereby one module is subsumed within another and performs part of the second module’s task.

The next level of abstraction consists of the modules. A module typically consists of two components: operations and data types. An operation identifies a particular result or set of actions that may be performed. For example, operation in a calculator program might include adding or printing data. A data type defines the type of item that an operator acts upon such as a student record or a daily balance.

Algorithms and data structures are more specific manifestations of operations and data types, respectively. An algorithm is a specific series of steps that accomplish a particular operation. Data structure is a precise representation or specification of a data type that consists of (i) basic data type groupings such as integers or characters, (ii) values, (iii) variables, (iv) arrays or groupings of the same data type, (v) records of groupings different data type, and (vi) pointers or connections between records that set aside space to hold the record’s values.

The computer program is written first in a programming language, such as Pascal or FORTRAN, and then in a binary language consisting of zeroes and ones. Source code is the literal text of a program’s instructions written in a particular programming language. Object code is the literal text of a computer program written in a binary language thorough which the computer directly receives its instructions.

These generalized levels of abstractions will not, of course, fit all computer codes. Ordinarily, expert testimony will be helpful to organize a particular program into various levels of abstraction. In any event, as pointed out earlier, the organization of a program into abstraction levels is not an end in itself, but it is only a tool that facilitates the critical next step of filtering out unprotectable elements of the program.4

As is clear from the above, the abstraction need not be performed in one specific way and that different ways of abstractions can be considered legal so long as the abstract is operationable in the subsequent phases. Although the Abstraction step was introduced and applied in a fairly good manner in Computer Associates v Altai, there was a general doubt whether abstraction of software can be carried out in any manner or should necessarily be in the way in which it was applied / used in in Computer Associates v Altai. The tenth circuit decision clarified this doubt. The abstraction process in (the planning phase of) POSAR sees the above as the starting point. Additionally, POSAR introduces certain added measures.

4 9 F.3d at 834-836, 28 USPQ2d at 1509-1510 (citations omitted)
As the starting point of the planning phase, the following suggestions (on the Abstractions step of AFC) of the five court-appointed experts (computer scientists) in Computer Associates v Altai have been considered. These suggestions (Hollar, 2002) are

   
   …… these abstractions concern the behavior of the program. Because programs can be viewed in terms of both their behavior and their text (treating the source code as a body of text), we can also describe the organization of the textual code itself at several levels of abstraction, ranging for example, from individual routines, to files containing multiple routines, to directories containing multiple files.

b. **Abstractions Should be Specific and Precise.**
   
   As one indicator of such precision, there should be no ambiguity about what behavior is being described and what body of literal code is being abstracted. This enables evaluation of the accuracy and completeness of the abstractions. Courts should thus require: (a) that labels on abstractions exhibits clearly specify the program behavior at issue; and (b) that each component of an abstraction exhibit refer clearly either to more detailed exhibits or to literal code…

c. **Abstractions At Any Given Level is complete.**
   
   Such completeness will ensure that the exhibits present an entire and accurate picture of the program at any chosen level of detail. The abstraction effort can be focused on the code relevant to the case at hand by cutting off the abstraction process below a certain level for the irrelevant parts of the code.

d. **Parties Should Adopt Explicit and Consistent Graphical Conventions for All Exhibits.**
   
   The adoption of consistent graphical conventions will permit the trier of fact to make sensible comparisons of the exhibits….To a trained eye, some of the graphical organization of the abstraction exhibits is informative, while other elements are accidental (e.g., the left-to-right order of certain of the boxes). If the parties cannot agree on a set of conventions, the court’s expert should assist the parties and the court in reaching a uniform set of standards for the exhibits.  

These basic legal suggestions on abstracting software are incorporated into POSAR too. Additionally, POSAR emphasizes that at any level of abstraction, certain valuable evidence like post-piracy modifications, programming blunders, programming errors etc. are not abstracted but are kept as such to be taken to the final phases of analysis, comparison and reporting as valuable evidence.

---

The objectives and rationale of introducing the Planning phase are basically five-fold: (1) To make POSAR a team-event rather than AFC (can be done only by single person handling); (2) To ensure that all the forensic areas are covered; (3) To ensure that all the functional areas in the software are covered; (4) To ensure that all the software parts are covered and; and (5) To ensure that nothing that could be evidence is filtered out.

The ultimate objective of the planning phase is to create two abstracted raw sets of items (one on the original and the other on the pirated) which can be taken to the next phase. Even though this is also the objective of the abstraction step of AFC, the way in which the abstraction of the two software packages is carried out in the planning phase (explained below) of POSAR is quite different from the abstraction that is carried out in AFC. The difference is primarily as follows. The abstraction step of AFC is generally performed in accordance with the US judiciary Tenth Circuit elaboration that “…a computer program can often be parsed into at least six levels of generally declining abstraction. (i) the main purpose; (ii) the program structure or architecture; (iii) modules; (iv) algorithms and data structures; (v) source code; and (vi) object code.”6. But POSAR recommends that the process of abstraction requires further elaboration and needs to be performed along the areas of investigation mentioned in Table 7.1, below. The Planning phase here insists that abstraction needs to be carried out taking into account also factors like programming remarks, programming blunders, programming errors, sequence of appearance of similarly looking items, database field properties, program manuals and documents.

---

6 9 F.3d at 834-836, 28 USPQ2d at 1509-1510 (citations omitted)
1. Functional area
2. Reports generated
3. Screens generated (Screen display and GUI are mentioned as part of AFC)
4. Menu structure (Menu command hierarchy)
5. Object Code strings / structure
6. **Source code (Design and Programming patterns)**
   a. Data Structure
      i. Nomenclature: Names given to functions, program variables, loop variables, function parameters, macros, included objects (JPEG files, songs, video clippings etc.)
      ii. Sequence of appearance of similarly looking items
   b. Programming Logic
   c. Programming remark
   d. Programming blunders
   e. Programming errors
   f. Finger prints
   g. Included objects
   h. Globally unique thumb impressions / finger prints
7. **Data base (Design and Programming patterns)**
   a. Nomenclature: Names given to databases, database tables, database fields etc
   b. Field properties: Field type, field length, field aliases etc.
   c. Sequence of appearance of similarly looking items
8. Data base procedures **(Design and Programming patterns)**
   a. Same as 6.a to 6.h, above
9. Documentations / manuals
10. The object code

| Table 7.1. Positive list of forensic investigation areas for the Planning phase of POSAR. |

Secondly, the planning phase of POSAR treats blunders differently during the investigation. A programming blunder or “copied misbehaviour”\(^7\) is sufficient to support the complaint (Hollar, 2002, p.___) but AFC’s abstraction step does not take any measure to ensure their retention. POSAR ensures that this valuable evidence stays for final comparison.

\(^7\) Amicus Brief of Computer Scientists, Harbor Solutions v. Applied Systems No. 97-7197 (2\(^{nd}\) Circuit, 1998) at 8-9 (citations omitted)
Thirdly, there is an important issue of “mining” to be addressed. This “mining” problem is a significant side-effect of the implementation of a suggestion that “*The Plaintiff Must have Access to the Defendant’s Code in Order to Focus the Inquiry*” during abstraction. This side effect has been cited by the court thus:

> If the plaintiff is permitted to examine the defendant’s code before abstracting his own code, the plaintiff may “mine” the defendant’s code for potential foci of alleged copying, thereby distorting the inquiry. But we believe some form of difficulty is unavoidable. If plaintiff instead must perform the abstraction and filtration before seeing the defendant’s code, it may feel that the only way to protect the claim is to abstract the entire program, resulting in an impractical—and wasteful—amount of work for both the plaintiff and the court. If, in the interests of economy, the plaintiff abstracts only those parts of its code that it suspects at the outset were copied by the defendant, the plaintiff’s exhibits are likely to be incomplete and require augmentation after it has had the opportunity to examine the defendant’s code, in which case the possibility of “mining” reappears. In any event, we believe that manufactured claims of copying become evidence at the filtration stage, where the court’s expert can raise such observations and the court can deal appropriately with them at that time.\(^8\)

Traditionally, AFC treats this problem as “unavoidable”. POSAR has built in measures that ensure that this “mining” problem does not happen during abstraction. The abstraction process is systematized in an algorithmic way along a specified list of parameters thus ensuring that plaintiff cannot “mine” defendant’s code. The step to be followed in the planning phase and the data flow within the planning phase are detailed in Fig. 7.2.

---

\(^8\) Amicus Brief of Computer Scientists, Harbor Solutions v. Applied Systems No. 97-7197 (2\textsuperscript{nd} Circuit, 1998) at 8-9 (citations omitted)
Thus, the outputs of the planning phase are the two raw sets of items which can be taken to the next phase. These two raw sets are further refined in the next two phases (operationalization and separation phases) to turn them into two sets of fully comparable items.
7.4.2 The Operationalization phase of POSAR

All software will contain elements that are universal and global which will automatically create similarity between any two software packages. Such similarities need to be set aside and filtered out in order to properly identify the similarities that can be explicitly assigned to piracy. The operationalization phase of POSAR tries to address this issue. Currently AFC does this by removing the following 4 types of global elements; (a) elements that are dictated by efficiency considerations (idea-expression merger considerations); (b) elements whose use are dictated by external factors such as functional area, calling sequence for a library routine, operating system function etc; (c) elements formed out of materials in the public domain; and (d) elements that are facts (except the particular selection or arrangement of facts). POSAR maintains that the filtering out of the above needs to be sensitive to two very important elements that currently unaddressed by AFC and these are; (i) all suspected post-piracy modifications (See chapter 4 above) and; (ii) all related design and programming pattern considerations (see chapter 5 above). This means that those elements which are either suspected post piracy modifications or whose design aspects can in some way provide supporting evidence to establish piracy should not be filtered out. In order to make the filtration process sensitive to the above two considerations, the original AFC list of items to be filtered out needs to be modified. The modified and re-structured list of filtration for POSAR is given in table 7.2., below and the de-globalisation process is diagrammatically shown in Fig 7.3, below.

---

9 982 F2d at 707-708, 23 USPQ2d at 1254 (citations omitted)

10 982 F2d at 709-710, 23 USPQ2d at 1255 (citations omitted)

11 9 F 3d at 836-837, 28 USPQ2d at 1511 (citations omitted)

12 9 F 3d at 837, 28 USPQ2d at 1511 (citations omitted)
1. Elements dictated by external factors. For instance, the calling sequence for a library routine, Operating system functions / commands etc. The standards adopted in AFC will be of help here.

2. Globally common mnemonics / expressions / names (except those misspelled, for instance, ‘stdunt_id’ for ‘student_id’)

3. Codes / expressions due to shared nature of global software education. For instance, ‘print_func’, ‘prn_function’ etc.

4. Codes / expressions that follow global notation standards. For instance, any function name like PI, DELTA etc. However, any rare function / variable names that have the string ‘PI’ or ‘Delta’ as a part, are not included here.

5. Codes / expressions due to shared nature of the globally common functional area jargons. For instance, function names in a banking software can be simply ‘SB’ for Savings Bank, ‘CA’ for current Account etc. However, any rare function name that has the string ‘SB’ or ‘CA’ is not included here.

6. Formatting styles in codes / expressions which are unique to a particular syntax-directed.

7. Codes due to programming language conventions. For instance, the self-imposed (self-generated) codes/expressions with the help of language-wizards, mandatory programming conventional codes/expressions. For example, ‘printf’ and ‘scanf’ in C++.

8. Materials that are available in the public domain. For instance, all library function names like rand( ), random( ) in C++, the API extensions etc. The standards adopted in AFC will be of help here.

9. Codes / expressions, which are unique (idiosyncratic) to the idea expressed.

10. Codes / Expressions that are part of the compatibility requirement with another program, with which this program is required to interoperate. The standards adopted in AFC will be of help here.

11. Codes / expressions / data base table names / data base names used because the software design globally demands it. The standards adopted in AFC will be of help here.

Table 7.2. Subtractive list for the Operationalization phase of POSAR
Fig. 7.3. Data flow in the Operationalization phase of POSAR
The output of this phase is a set of two filtered abstractions. According to the traditional AFC, these form the data to be compared in order to establish possible piracy. However, POSAR protocol believes that these are not yet ready for final comparison and should necessarily go through another phase for separating post-piracy modifications.

7.4.3 The Separation phase of POSAR

This phase deals with the suspected post-piracy modifications in the above two filtered abstractions (which form the input to this phase). Post-piracy modifications cause apparent surface differences that would decrease the similarity and thus favour the defendant (See chapter 4). By eliminating these differences, the actual similarities in the ideal original and the pirated will be more clearly visible. In order to identify these actual similarities, the idealized forms of both versions of the software need to be obtained. So both sets of software, where there are suspected post-piracy modifications (potential evidence), are identified and listed. Further, the two versions of the software are then purged of these modifications and idealized versions are obtained in that both are reverted to their pre-modified (idealized) form. This phenomenon is absent in AFC because post-piracy modifications and their role are not given due consideration in AFC.

The output of this phase is in three sets. The first two sets contain a pair of comparable items. The third is derived from the first two and is a list of suspected post-piracy modifications. These three sets are explained below. [The abbreviation SO, below, represents the source codes, data base structure, images etc. that form the original (of the complainant), while the abbreviation SP represents that of the pirated.]

i. Set $A=\{SO_1, SP_1\}$ of two comparable source codes with marked, suspected post-piracy modifications. $SO_1$ and $SP_1$ are formed out of the two filtered abstractions (which form the input to this phase). Comparison of $SO_1$ & $SP_1$ during the next phase will yield a list of actual similarities between the original and the pirated.
ii. Set $B = \{SO_2, SP_2\}$ of two comparable source codes with idealized (pre-modified) form of the suspected-post piracy modifications. SO2 and SP2 are formed out of SO1 and SP1. Unlike the result of comparison of SO1 & SP1, above, comparison of SO2 & SP2 will very likely yield a different (but never a smaller) and quite possibly an increased list of similarities due to the absence of interference from suspected post-piracy modifications.

iii. Set $C = \{SO_3, SP_3\}$ of two lists of post-piracy modifications; one for the original and the other for the pirated. SO3 is likely to be a null set because the investigation agency may be able to obtain an unmodified source code from the complainant. Set C is not a class similar to A & B. Unlike sets A & B above, set C is not used for comparison but for general analysis and the results of this analysis are expected to add credibility to the results of comparison of the SO2 & SP2.

The above three sets are the inputs to the next phase (the Analysis phase) for comparison and analysis.
Fig. 7.4. Data flow in the Separation phase of POSAR
7.4.4 **The Analysis phase of POSAR**

This phase is dedicated to the final comparison of the original and the pirated and further, the analysis of the suspected post-piracy modifications. The inputs to this phase are the three sets $A=\{SO1, SP1\}$, $B=\{SO2, SP2\}$ and $C=\{SO3, SP3\}$, mentioned in the last phase.

In this phase, there are four tasks to be carried out and they are: (i) $SO1$ and $SP1$ are compared (along the valid features mentioned in Annexure-III) for enlisting all apparent similarities / commonalities; (ii) the marked elements which are dictated by design and programming consideration (see operational phase) are compared for enlisting suspected piracy of thumb impressions, programming errors, programming blunders etc. in the pirated (See chapter 4) and; (iii) the idealized pre-modified form of the suspected post-piracy modifications marked in $SO2$ and $SP2$ are separately compared (iv) the elements in $SP3$ and $SO3$ are separately analysed to find if there is any contributing element in them to add credibility to the results of comparison of i to iii above. In AFC, only task (i) above is really carried out.

The first of the above four tasks is part of the AFC process as explained below:

The third and final step of the (AFC) test for substantial similarity that we believe appropriate for non-literal program components entails a comparison. Once a court has sifted out all elements of the allegedly infringed program which are “ideas” or are dictated by efficiency or external factors, or taken from the public domain, there may remain a core of protectable expression. In terms of a work’s copyright value, this is the golden nugget. At this point, the court’s substantial similarity inquiry focuses on whether the defendant copied any aspect of this protected expression, as well as an assessment of the copied portion’s relative importance with respect to the plaintiff’s overall program.\(^{13}\)

This new protocol would like to underscore that the remaining three tasks, though not part of AFC, are nevertheless of key importance and will certainly help enhance the final results. The result of the first task above will determine whether the protectable elements of the programs at issue are substantially similar so as to warrant a finding of

\(^{13}\) 982 F2d at 710, 23 USPQ2d at 1256 (citations omitted)

135
infringement. The results, if any, of the second, third and fourth tasks above will act as valuable supporting evidence and might contribute to the credibility of the result of the first task. The results of the four tasks above will be approximately in the following form.

7.4.4 A list of apparent similarities

7.4.5 A list of similarities from design and programming pattern considerations (for instance, thumb impressions, programming errors, programming blunders etc.)

7.4.6 A list of similarities arising from suspected post-piracy modifications factor and an analysis report of suspected post-piracy modifications in the pirated

These four results are severally made over to next phase (Reporting phase) of POSAR.
Fig. 7.5. Data flow in the Analysis phase of POSAR
7.4.5 The Reporting phase of POSAR

Inputs to this phase are the three lists of similarities and commonalities and the detailed report of the suspected post-piracy modifications. Also, their source, the sets A={SO1, SP1}, B={SO2, SP2} and C={SO3, SP3} are passed on to this phase as input. In this phase, these lists, report and sets are further studied for the purpose of reporting similarities and commonalities, presented in numerical and non-numerical terms. Several formats are postulated for reporting piracy in numerical terms (See Table 5.1 and 5.2 in chapter 5, for report formats already discussed. Also, see Baboo and Bhattathiripad (2009, 2010)).

Unlike AFC, POSAR recommends that there be a separate reporting phase in order to stress the importance of the presentation of the findings in a professional manner and in standard formats. It is a fact that due to the lack of standards in reporting digital evidence items, investigators are facing difficulties in efficiently presenting their findings (Bariki et al, 2010) and AFC suggests no reporting standards for presenting software piracy forensic investigation results. Bariki et al believe that any reporting standards should suggest formats which uphold legal integrity of the investigation results. The reporting formats suggested by Baboo and Bhattathiripad (2009, 2010) would be of help here in properly reporting software piracy investigation results.
Fig. 7.6. Data flow in the Reporting phase of POSAR
7.5 POSAR juxtaposed with AFC

With the help of the above five-phase protocol, software copyright infringement can be established in a better way than that of AFC. The features and phases of POSAR, some of which have roots in AFC, are summarized below in table 7.3, 7.4 and 7.5.

<table>
<thead>
<tr>
<th>Features of the Planning phase</th>
<th>Relation to AFC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Six levels of generally declining abstraction. (i) the main purpose; (ii) the program structure or architecture; (iii) modules; (iv) algorithms and data structures; (v) source code; and (vi) object code</td>
<td>Taken from the Abstraction step of AFC</td>
</tr>
<tr>
<td>Routine abstractions include control structures, data structures, data flow information, architecture and textual organisation of the code</td>
<td>Taken from the Abstraction step of AFC</td>
</tr>
<tr>
<td>Abstractions should be specific and precise</td>
<td>Taken from the Abstraction step of AFC</td>
</tr>
<tr>
<td>Abstractions at any given level is complete</td>
<td>Taken from the Abstraction step of AFC</td>
</tr>
<tr>
<td>Parties should adopt explicit and consistent graphical conventions for all exhibits</td>
<td>Taken from the Abstraction step of AFC</td>
</tr>
<tr>
<td>Listed areas of forensic coverage</td>
<td>AFC list modified</td>
</tr>
<tr>
<td>Elements in the software that are dictated by design and programming considerations are made available for final comparison</td>
<td>Newly added feature</td>
</tr>
<tr>
<td>Elements in the software that are dictated by suspected post-piracy modifications are made available for final comparison</td>
<td>Newly added feature</td>
</tr>
<tr>
<td>The plaintiff must have access to the defendant’s code in order to focus the inquiry</td>
<td>Taken from the Abstraction step of AFC</td>
</tr>
</tbody>
</table>

Table 7.3. The Planning phase’s relation to AFC
<table>
<thead>
<tr>
<th>Features of the operationalization phase</th>
<th>Relation to AFC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Elements that are dictated by efficiency considerations (idea-expression merger considerations) are removed</td>
<td>Taken from the Filtration step of AFC</td>
</tr>
<tr>
<td>2. Elements whose use are dictated by external factors such as functional area, calling sequence for a library routine, operating system function etc are removed</td>
<td>Taken from the Filtration step of AFC</td>
</tr>
<tr>
<td>3. Elements formed out of materials in the public domain; and (d) elements that are facts are removed</td>
<td>Taken from the Filtration step of AFC</td>
</tr>
<tr>
<td>4. Elements that are facts (except the particular selection or arrangement of facts) are removed</td>
<td>Taken from the Filtration step of AFC</td>
</tr>
<tr>
<td>5. Suspected post-piracy modifications are not filtered out</td>
<td>Newly added feature</td>
</tr>
<tr>
<td>6. Elements dictated by design and programming pattern considerations are not filtered out</td>
<td>Newly added feature</td>
</tr>
<tr>
<td>7. Globally common mnemonics / expressions / names that are misspelled (for instance, ‘stdnt_id’ for ‘student_id’) are not filtered out</td>
<td>Newly added feature</td>
</tr>
<tr>
<td>8. Codes / expressions due to shared nature of global software education (For instance, ‘print_func’, ‘prn_function’ etc.) are filtered out</td>
<td>AFC list modified</td>
</tr>
<tr>
<td>9. Codes / expressions that follow global notation standards are filtered out</td>
<td>AFC list modified</td>
</tr>
<tr>
<td>10. Codes / expressions due to shared nature of the globally common functional area jargons are filtered out</td>
<td>AFC list modified</td>
</tr>
<tr>
<td>11. Formatting styles in codes / expressions which are unique to a particular syntax-directed are filtered out</td>
<td>Taken from the Filtration step of AFC</td>
</tr>
<tr>
<td>12. Self-imposed (self-generated) codes/ expressions with the help of language-wizards, mandatory programming conventional codes/expressions are filtered out</td>
<td>Taken from the Filtration step of AFC</td>
</tr>
<tr>
<td>13. Codes / Expressions that are part of the compatibility requirement with another program, with which this program is required to interoperate are filtered out</td>
<td>Taken from the Filtration step of AFC</td>
</tr>
<tr>
<td>14. Codes / expressions / data base table names / data base names used because the software design globally demands it, are filtered out</td>
<td>Taken from the Filtration step of AFC</td>
</tr>
</tbody>
</table>

Table 7.4. The Operationalization phase’s relation to AFC
<table>
<thead>
<tr>
<th>The last three phases of POSAR</th>
<th>Features of the phase</th>
<th>Relation to AFC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Separation phase</td>
<td>Suspected post-piracy modifications are separated</td>
<td>Newly added feature</td>
</tr>
<tr>
<td></td>
<td>Idealised pre-modified form of the software is obtained</td>
<td>Newly added feature</td>
</tr>
<tr>
<td>2 Analysis phase</td>
<td>The two filtered abstractions are compared</td>
<td>Idea taken from the Comparison stage of AFC</td>
</tr>
<tr>
<td></td>
<td>The elements which are dictated by design and programming consideration are compared for enlisting suspected piracy of thumb impressions, programming errors, programming blunders etc. in the pirated</td>
<td>Newly added feature</td>
</tr>
<tr>
<td></td>
<td>The pre-modified form of the suspected post-piracy modifications are separately compared and the results, if any, are used for adding value to the final result</td>
<td>Newly added feature</td>
</tr>
<tr>
<td></td>
<td>The post-piracy modifications are separately analysed the results, if any, are used for adding value to the final result</td>
<td>Newly added feature</td>
</tr>
<tr>
<td>3 Reporting phase</td>
<td>Reports are presented in newly created formats</td>
<td>Newly added feature</td>
</tr>
<tr>
<td></td>
<td>Some of the results of investigations are presented numerically</td>
<td>Newly added feature</td>
</tr>
</tbody>
</table>

Table 7.5. The last three phases’ relation to AFC

7.6 Algorithmic (subjective) steps for carrying out POSAR

The software piracy forensic investigation using POSAR needs to be carried out along the following phases.

7.6.1 The Planning phase: The ultimate objective of the planning phase is to create two abstracted raw sets of items (one on the original and the other on the pirated) which can be taken to the next phase.

7.6.1.1 Input to the planning phase: The pirated software and the original software received from the investigation agency.

7.6.1.2 Steps in the Planning phase for the pirated
Step 1: Select one area of forensic investigation (see Table 7.1. for the positive list of areas where forensic investigation needs to be carried out).

Step 2: Collect data to investigate.

Step 3: Compile investigation tools.

Step 4: Abstract the pirated based on (in terms of) the selected area of investigation.

Step 5: Do step (i) to (iv) on the pirated until all the areas in Table 7.1. are covered. Thus a set of abstractions are obtained from the pirated.

7.6.1.3 Steps in the Planning phase for the original

Step 1: Select one area of forensic investigation (see Table 7.1. for the positive list of areas where forensic investigation needs to be carried out).

Step 2: Collect data to investigate.

Step 3: Compile investigation tools.

Step 4: Abstract the original based on (in terms of) the selected area of investigation.

Step 5: Do step (i) to (iv) on the original until all the areas in Table 7.1. are covered. Thus a set of abstractions are obtained from the original.

7.6.1.4 Output of the planning phase: Two raw sets of items to be compared (to be passed on to the Operationalization phase). (These sets are raw because the globally common items as well as the post-piracy modifications are yet to be filtered out of these sets).

7.6.2 The Operationalisation phase: In this phase, all filterable items are removed (listed in Table 7.2.) from the above two raw sets.
7.6.2.1 **Input to the Operationalization phase:** Two abstracted raw sets of items obtained in the planning phase (Raw, because there are filterable elements in these sets).

7.6.2.2 **Steps in Operationalization phase for the pirated**

Step 1: Select the abstracted set of the pirated obtained from the Planning phase.

Step 2: Mark all suspicious post-piracy modifications in this set by comparing this set with the respective portion of the abstracted set of the original. Ensure that these marked codes are not removed during the subsequent stages of the Operationalization phase.

Step 3: Mark all suspicious elements (in this set) that are dictated by the design and programming considerations. Ensure that these marked elements are not removed during the subsequent steps of the Operationalization phase.

Step 4: Remove from this set, the elements dictated by efficiency considerations. Codes / expressions, which are unique (idiosyncratic) to the idea expressed are removed here. Here AFC’s “merger” doctrine is used. This doctrine states that if there is only one way of effectively expressing an idea (a function), this idea and its expression tend to “merge”. Since the idea itself would not be protectable, the expression of this idea would also escape from the field of the protection.

Step 5: Remove from this set, the elements dictated by external factors.

Step 6: Remove from this set, the elements formed out of materials in the public domain.

Step 7: Remove from this set, the elements that are facts (except a particular selection or arrangement of facts).
7.6.2.3 Steps in Operationalization phase for the original

Step 1: Select the abstracted set of the original obtained from the Planning phase.

Step 2: Mark all suspicious post-piracy modifications in this set by comparing this set with the respective portion of the abstracted set of the pirated. Ensure that these marked codes are not removed during the subsequent stages of the Operationalization phase.

Step 3: Mark all suspicious elements (in this set) that are dictated by the design and programming considerations. Ensure that these marked elements are not removed during the subsequent stages of the Operationalization phase. Remove from this set, the elements dictated by efficiency considerations.

Step 4: Remove from this set, the elements dictated by efficiency considerations. Codes / expressions, which are unique (idiosyncratic) to the idea expressed are removed here. Here AFC’s “merger” doctrine is used. This doctrine states that if there is only one way of effectively expressing an idea (a function), this idea and its expression tend to “merge”. Since the idea itself would not be protectable, the expression of this idea would also escape from the field of the protection.

Step 5: Remove from this set, the elements dictated by external factors.

Step 6: Remove from this set, the elements formed out of materials in the public domain.

Step 7: Remove from this set, the elements that are facts (except a particular selection or arrangement of facts).

7.6.2.4 Output of the Operationalization phase: Pass on the two sets obtained from 7.15.2.2 and 7.15.2.3 to the Separation phase. These are two sets of partially comparable items (‘partially’ because the post-piracy modifications are yet to be removed from the pirated)
7.6.3 **The Separation phase:** In this phase, all suspected post-piracy modifications are sieved out from the pirated and enlisted separately so that this list can be separately analysed and used while arriving at the final result in the Separation phase.

7.6.3.1 **Input to the Separation phase:** Two sets of partially comparable items obtained from the Operationalization phase.

7.6.3.2 **Steps in the Separation phase for the pirated**

Step 1: Select the abstracted, localized set of the pirated and call it SP1. Take a copy of SP1 and call the new set as SP2. Also create a new, blank set by name SP3.

Step 2: Separate (take a copy of) all the marked elements (suspected post-piracy modifications) from SP1 and append SP3 with these elements. Also, re-engineer the respective elements in SP2 to their idealized pre-modified form.

Step 3: Do step (ii) until all the marked elements in SP1 are separated.

7.6.3.3 **Steps in the Separation phase for the original**

Step 1: Select the abstracted, localized set of the original and call it SO1. Take a copy of SO1 and call the new set as SO2. Also create a new, blank set by name SO3.

Step 2: Separate (take a copy of) all the marked elements (suspected post-piracy modifications) from SO1 and append SO3 with these elements. Also, re-engineer the respective elements in SO2 to their idealized pre-modified form.
Step 3: Do step (v) until all the marked elements in SO1 are separated.

7.6.3.4 **Output of the Separation phase:** Pass on SP1, SP2, SP3, SO1, SO2 & SO3 to the Analysis phase. SP1 and SO1 are comparable sets with the marked post-piracy modifications. SP2 and SO2 are comparable sets with idealized pre-modified form of SP1 and SO1, respectively. SP3 and SO3 are the suspected post-piracy modifications in SP1 and SO1, respectively.

7.6.4 **The Analysis phase:** In this phase, the two fully comparable sets (SP1 and SP2) are compared to identify and enlist the possible similarities / commonalities, if any. (along the valid features mentioned in Annexure-III). Also, the suspected post-piracy modifications (elements in SP3 & SO3) are separately analysed with the help of the marked, idealized pre-modified form of the post-piracy modifications (elements in SP2 and SO2).

7.6.4.1 **Input to the Analysis phase:** Two sets A={SP1, SO1}, B={SP2, SO2} of fully comparable items with the marked post-piracy modifications. Also, a third set C={SP3, SO3} of suspected post-piracy modifications.

7.6.4.2 **Steps in the Analysis phase**

   Step 1: Compare the respective elements in SP1 and SO1 and enlist the similarities and commonalities in the original and the pirated

   Step 2: Compare the marked elements which are dictated by design and programming considerations and enlist piracy of, for instance, thumb impressions, programming errors, programming blunders etc. (See chapter 4 for modus operandi of this comparison)

   Step 3: Analyse the suspected post-piracy modifications (elements in SP3 & SO3) with the help of the marked, idealized pre-modified form of the
post-piracy modifications (elements in SP2 and SO2) (See chapter 5 for modus operandi of this analysis).

7.6.4.3 Output of the Analysis phase: One list of similarities / commonalities between the original and the pirated and another list of piracy of thumb impressions, programming errors, programming blunders etc. Also, a third list of suspected post-piracy modifications along with the idealized, pre-modified form of these suspected post-piracy modifications.

7.6.5 The Reporting phase: In this phase, various statistical reports are prepared based on the two lists obtained from the analysis phase.

7.6.5.1 Input to the Reporting phase: One list of similarities / commonalities between the original and the pirated and another list of piracy of thumb impressions, programming errors, programming blunders etc. Also, a third list of suspected post-piracy modifications along with the idealized, pre-modified form of these suspected post-piracy modifications.

7.6.5.2 Steps in Reporting phase

Step 1: Study the two lists, the report (both created in the Analysis phase) and the sets SP1, SP2, SP3, SO1, SO2, and SO3 and arrive at similarities and commonalities in numerical / non-numerical terms.

Step 2: Prepare reports of similarities and commonalities using the report formats (See the table 7.6 & 7.7 below. However, the respective forensic expert should be at liberty to customize these report formats by not diluting the core aspects of these two formats).

Step 3: Submit these reports to the law enforcement agency.

7.6.5.3 Output of the Reporting phase: Reports with fewer jargons and in a manner that is less-technical and non-esoteric
1. **Similarity in the table names:** ___% commonality.

2. **Length of the ‘original’ table:** a

3. **Length of the ‘pirated’ table:** b

4. **Percentage of similarity in lengths:** (a/b)*100

5. **Field count of the ‘original’:** c

6. **Field count of the ‘pirated’:** d

7. **Percentage of similarity in field count:** (c/d)*100

8. **Perfect commonality in the names of fields:** ___ out of ___ names of fields in the ‘original’ are found in ‘pirated’ also. So, ___% commonality

9. **Perfect commonality in name and data type among fields:** ___ out of ___ fields have the common name and data types. So, ___% commonality in name and data type.

10. **Perfect commonality in name, data type and length among fields:** ___ out of ___ fields have same names, data types and length. So, ___% commonality

11. **Perfect commonality in name, data type, length and the default values set in the fields:** ___ out of ___ fields have same names, data types, length and default values. So, ___% commonality.

12. **Perfect commonality in sequence of the fields with same name:** ___ out of ___ fields with same name, do occur in the same sequence. So, ___% commonality.

13. **Perfect commonality in sequence of the fields with same name, data type, length and default values:** ___ out of ___ fields (with same name, data type, length and default values) do occur in the same sequence. So, ___% commonality.

14. **Count of comparable (mappable) fields including suspected-post-piracy modified / created fields:** ___ out of ___ fields in the ‘pirated’ can be perfectly or approximately mapped (in terms of names) to at least one field in the ‘original’. So, ___% comparable fields in the ‘pirated’.

15. **Count of non-mappable but suspected-post-piracy fields, including ending fields:** ___ out of ___ fields in the ‘pirated’ could not be properly mapped to any of the fields in the ‘original’ but they can be suspected to be post-piracy modifications. So, ___% incomparable but suspected fields in the ‘pirated’

16. **Count of non-mappable, non-suspected fields:** ___ out of ___ fields in the ‘pirated’ could not be properly mapped to any of the fields in the ‘original’ and do not provide any clue to be suspected as post piracy modification. So, ___% incomparable, non-suspected fields in the ‘pirated’

17. **Inference:** Piracy is confirmed / largely suspected / loosely suspected / not suspected.

---

Table 7.6: The proposed format for presenting the result of comparison of two database tables (Bhattathiripad, 2009) (To be prepared for each database structure as part of the software)
**Procedure / code segment comparison report:** The reason for comparing these two procedures / code segments is that both are doing the same job in the respective software packages ('original' as well as 'pirated'). Moreover, no other procedure / code segment in the respective software package was seen doing this job. Thus, it can be ascertained that they were made for executing the same process.

Name of the procedure in the ‘original’ software: ______

Name of the procedure in the ‘pirated’ software: ______

**Results of comparison of procedure / code segment:**

i. **Similarity in the name of the procedure:** ___% similarity

ii. **Evidence, in general:** The ___ variables / other parameters in the ‘original’ are found exactly named and declared (except length, in case of NUMERIC definition) in the same manner in the ‘pirated’ also. The parameters, namely, _____, _____ (populate here the list of variables / parameters, including type and length, if any) occur in both ‘original’ and ‘pirated’ in the same position and in the same sequence. Considering the variety in styles of nomenclature and thinking process of programmers, this is / is not (strike off whichever not applicable) extra ordinary. The ‘pirated’ has ___ more additional parameters namely _____, _____ (populate here the list of post-piracy modified / added variables / parameters, including type and length, if any), which may have added after the ‘piracy’ has occurred, as their positions come after all the ___ parameters in the ‘original’.

Further analysis of these ___ parameters (names of variables / parameters) have the produced the following statistics, in percentage:

iii. **Similarity in names and data types:** __%. 

iv. **Similarity in names, data types & length:** ___% ( ___ out of ___)

v. **Similarity in position of appearance:** ___% 

vi. **Similarity in the sequence of appearance:** ___% 

vii. **Perfect similarity in statements:** There are exactly ___ statements present in these two procedures in the ‘original’ as well as the ‘pirated’. All of them look exactly similar in styles, positioning and sequence of appearance of codes (except the inclusion / exclusion of the ___ (additional) parameters in the ‘pirated’). The ‘pirated’ neither has anything extra nor has anything lacking. So, the ___ statements in these two procedures are 100% similar. This can be / cannot be considered as pieces of evidence for copyright infringement.

viii. **Considerable similarity in statements:** There are exactly ___ statements present in these two procedures in the ‘original’ as well as the ‘pirated’. They look somewhat similar in styles, positioning and sequence of appearance of codes (except the inclusion / exclusion of the ___ (additional) parameters in the ‘pirated’). The ‘pirated’ neither has anything extra nor has anything lacking (strike off if this sentence is not applicable). So, these ___ statements in these two procedures are ___% similar. This can be / cannot be considered as pieces of evidence for copyright infringement.

ix. **Inference:** Copyright infringement / Piracy is confirmed / largely suspected / loosely suspected / not suspected is suspected / not suspected (strike off whichever is not applicable).

Table -7.7: The proposed format for presenting the results of comparison of two data base procedures / code segments (Bhattathiripad, 2002) (To be prepared for each such procedure / code segment / subprogram in the software)
7.7 Conclusion

The need for the new protocol (as a modified form of AFC) has been addressed in this chapter. The new protocol, POSAR, offers something more to the legal and the judicial domain than what AFC offers. The efficiency and praxis of POSAR will be verified and validated in Chapter 8.