Journal of Computational Design and Engineering Vol. 1, No. 3 (2014) 161~172
www.jcde.org

A comparison of three design tree based search algorithms for the detection of engineering parts constructed with CATIA V5 in large databases

Robin Roj[1,*]

[1] *University of Wuppertal (FB D Mechanical Engineering, Mechanical Engineering Informatics, D 42097 Wuppertal, Germany)*

(Manuscript Received January 26, 2014; Revised April 24, 2014; Accepted April 24, 2014)

Abstract

This paper presents three different search engines for the detection of CAD-parts in large databases. The analysis of the contained information is performed by the export of the data that is stored in the structure trees of the CAD-models. A preparation program generates one XML-file for every model, which in addition to including the data of the structure tree, also owns certain physical properties of each part. The first search engine is specializes in the discovery of standard parts, like screws or washers. The second program uses certain user input as search parameters, and therefore has the ability to perform personalized queries. The third one compares one given reference part with all parts in the database, and locates files that are identical, or similar to, the reference part. All approaches run automatically, and have the analysis of the structure tree in common. Files constructed with CATIA V5, and search engines written with Python have been used for the implementation. The paper also includes a short comparison of the advantages and disadvantages of each program, as well as a performance test.

Keywords: CAD; CATIA V5; Classification; Database; Dat mining; Design tree; Feature recognition; Knowledge Discovery; Python; Search engine

1. Introduction

Contemporary construction techniques of engineering parts are heavily influenced by the usage of computer-aided methods for the virtual design of new products, as well as for the management of whole projects. The appropriate software is available for the different demands, like e.g., the conception and technical draft of single parts, or the administration and coordination of several other engineering tasks, and can be summarized as Computer-Aided Design (CAD) and Computer-Aided Engineering (CAE), respectively. The shift from manually drawn sketches and sophisticated manufacturing of prototypes in the development phase of new components to Virtual Product Development (VPD) supported by simulations or computations has yielded many advantages, in terms of cost effectiveness and quality improvement.

But new challenges and tasks have arisen with the execution of computer-aided technology. One major difficulty is the handling of large amounts of data that are produced during the usage of software belonging to Product Lifecycle Management (PLM). Just a few examples are specialized files for virtual models, technical drawings, FEM-calculations and assemblies, or additional data, like tables, images, presentations, and even videos. A common strategy in large companies like car manufacturers, who produce a lot of the mentioned information, is storage in large databases with company-wide access.

An important question is the setup and the structure of such a database, depending on the desired objectives. These intentions could be e.g., good documentation of the accomplished work or might even include all possibilities of comfortable reaccess to the stored information. Because of the many imaginable styles of the warehousing of data in databases, like alphabetical or chronological order, as well as sorting depending on the different departments or branches of a company, it can be a challenging task to regain once stored information.

For this research area, the keywords Knowledge Discovery in Databases (KDD) and Data Mining are introduced. According to Fayyad, Piatetsky-Shapiro and Smyth KDD, can be defined as a procedure [1]: "*KDD is the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data.*"

The same authors describe the approach of KDD, as follows [2]: "*KDD focuses on the overall process of knowledge discovery from data, including how the data are stored and accessed, how algorithms can be scaled to massive datasets and still run effectively, how results can be interpreted and visualized, and how the overall man-machine interaction can*

*Corresponding author. Tel.: +49-202-439-2090, Fax.: +49-202-439-2091
E-mail address: r.roj@uni-wuppertal.de

doi: 10.7315/JCDE.2014.016

usefully be modeled and supported."

Frawley, Piatetsky-Shapiro and Matheus consider the three most important aims of KDD to be summarization, discrimination and comparison, referring to a clear distinction and categorization of the properties of extracted data [3].

Data Mining describes a similar strategy and the etymology developed according to Petersohn from colloquial language for the exploitation or extraction of rare materials, compared with valuable information from large amounts of data [4]. A good overview is provided by Gorunescu [5]. Hilderman and Hamilton describe an evaluation of the extracted information for the measurement of interestingness. They consider classification, association, clustering and correlation as the four most important techniques for the extraction of data [6].

This publication presents a specific case of data extraction. For the implementation of the algorithms, the CAD program CATIA V5, and the programming language Python have been chosen. Due to the fact that Python is connectable with CATIA via the COM-interface both programs are predestinated for the creation of macros, which are able to automate certain steps that are usually done by hand. The examination of a large database filled with engineering parts constructed with CATIA is demonstrated. As already mentioned the aim is a certain kind of classification of random components that should work as automatically and independent as possible.

At this point, the constitution of CAD models generated by most CAD systems, as well as CATIA should be explained briefly. Usually, the constructor builds the virtual model of the engineering part by the usage of several functions, like e.g., extrusions, rotations, drillings, roundings, or chamfers, to gain the intended shape. These properties of each part are called features. According to Vajna et al. [7], features can not only be labeled as geometrical elements, but also as relevant informational elements like relations and constraints.

The set of all features, and the unambiguous determination of every feature a model consists of, forms the precise definition of the part; and the combination of both leads to a certain singularity. All common CAD systems save these features in the created files, and also in the so called construction tree or design tree, where the user is able to comprehend the contained specifications. According to Kornprobst, the structure tree illustrates all the construction steps, which lead to explicit geometry or rules in a chronological order [8].

In Sections 3 and 4, three search algorithms are introduced, which examine the structure trees of given CAD models. The information contained in the structure trees are outsourced from CATIA, and stored in the form of XML-files for easier access, and subsequent classification. In Section 5, the advantages and disadvantages of the three search algorithms are compared, and the cases they are more or less appropriate for are determined.

2. State of the art

For the setting described in Section 1, many scientific and industrial concepts already exist. Here it should focus on engineering applications, and thus databases that are filled with CAD and CAE files, respectively. Approaches dealing with the general structure of such information are presented by Ester et al. [9], who focus on spatial databases; as well as Haffey and Duffy [10], who connect the topic with design issues. The dissertation of Angkasith concentrates on modular design [11]. In particular, the management of engineering products, which are manufactured by several suppliers and only mounted by the principal, is quite ambitious, regarding the administration of the generated data, and the coordination of every single working step. For such cases, a separation of the final product into modules might be time and cost reducing.

The representation of knowledge, and the interaction of the single elements with each other, can be visualized e.g., by directed graphs. Also, the above mentioned structure tree of CATIA is considered as a graph, and therefore outsourced into the XML-format, which is able to illustrate hierarchical structures. An example of the usage of graphs for the description of complex circumstances is presented by Kizu et al. [12]. They show a method for CAD Data Mining, and the detection of two-dimensional objects.

For engineering applications, not only the virtual construction of new products is important, but also a well elaborated production plan. Consequently, a strict separation of CAD features and manufacturing features take place, and has to be taken into account during the automatic feature recognition. A comprehensive review of Data Mining in manufacturing is given by Harding et al. [13].

Babic, Nesic and Miljkovic list the three main problems of Automated Feature Recognition (AFR) as 1. Extraction of the geometric primitives from a CAD model, 2. Defining a suitable part representation for form feature identification, and 3. Feature pattern matching/recognition [14]. In another review by Iyer et al. [15], the following techniques for the detection of shapes are structured in six categories: 1. Global feature-based techniques, 2. Manufacturing feature recognition-based techniques, 3. Graph-based techniques, 4. Histogram-based techniques, 5. Product information-based techniques, and 6. 3D object recognition-based techniques.

Two vivid examples of the automated recognition of three-dimensional objects are suggested by Min and Bowyer [16], who detect edges and reconstruct surfaces by image segmentation; and Cucchiara et al. [17], who use visual constraint graphs for an analysis of spatial components. Relational graphs are also used by Flynn and Jain [18]. They connect their topic with the storage of gained information in a database, which might later be used as a basis for manufacturing. While their publication deals with a library of proprietary CAD files, Cybenko, Bhasin and Cohen plan a global system for the representation, and in particular the reuse of once detected shapes [19]. By applying the examination techniques already in the design phase, and the usage of voxel-

Table 1. Overview of all programs.

Program	Description	Input	Output
tree.py	Outsources the structure trees of CATIA to XML-files	Database with CATPart-files	Database with XML-files
norm.py	Searches for standardized parts in the given database	Database with XML-files and CATPart-files, name of a norm part	Names of the located files that are similar to the selected norm part
category.py	Searches for parts in the given database that fit to the parameters	Database with XML-files and CATPart-files, search parameters	Names of the located files that fit the given parameters
match.py	Searches for parts in the given database that match to the reference	Database with XML-files and CATPart-files, reference XML-file and CATPart-file input-part	Names of the located files in three categories (perfect, very good, good)

based approximations, the current part is compared to all related files in a database, to identify similarities, and to integrate new objects into that library.

Of course, all of these techniques should reduce costs, and increase efficiency in the design and manufacturing process. Ip and Regli [20] focus on the recognition of fabrication features in particular. With such an automated recognition of features, and the separation into production steps, the sequence of machine application can be generated without any human assistance. Horváth and Rudas [21] describe the necessity of such interdisciplinary cooperation.

With once extracted features and any other information, respectively, a systematic and automated search in a database can take place. The following publications deal with search engines, which use the user's input as search parameters to find certain models in a database. The strategy of Min et al. [22] can be partitioned into three steps: (1) Acquisition: 3D models have to be collected from the web, (2) Analysis: They have to be analyzed for later matching, and (3) Query processing and matching: An online system has to match user queries to the collected 3D models.

Another approach of Data Mining techniques connected with a search engine is presented by Wei and Yuanjun [23]. For the retrieval of parts in a large database, they use voxel-based technologies; whereas Ansary, Daoudi and Vandeborre focus on two-dimensional views, to implement a method they call adaptive views clustering [24]. With that procedure, they are able to retrieve three-dimensional models from a database.

At this point, it should be referred to the Princeton Shape Retrieval and Analysis Group [25], where several publications, like e.g., the dissertation of Min [26], and the journal paper of Funkhouser et al. [27], deal with the topic of three-dimensional detection of shapes in databases in detail.

Another strategy for the handling of gathered information from databases, or feature parameters from single parts, is the categorization into classes. For engineering parts in particular, an order of CAD models into different types is recommendable. If the database of an automobile manufacturer is considered, an imaginable separation of all components into the various car units is a common practice for the division into different departments in the company, as well as an order from different suppliers. For this task, an algorithm called K-means is a common tool. Michalik, Štofa and Zolotová test the properties of the K-means algorithm on Data Mining applications [28].

A neat example of the classification of symmetric rotational parts by the recognition of surface features is given by Wang and Chang [29]. They suggest a method that makes any human assistance redundant.

All mentioned requirements that are necessary for user-friendly operability of software are realized in the following three programs, which are designed for application in industry. First a program called Geolus Search by Siemens is introduced [30]. Basically, it can be considered as a three-dimensional search engine for tessellated CAD models. The user predetermines a reference part, and provides the database, which should be examined. Further options can be chosen by refining the search for identical, very similar or similar parts. The graphical user interface shows all matching files stored in the memory, and enables a selection of the located models for possible update or continue constructions. The intention is a reduction and elimination, respectively, of redundant models that are already available in the storage. Thus, the time for the creation of new components can be reduced, and thereby the labor costs.

The second program that deals with the automatic classification of parts is called simus classmate by Lino [31]. It specializes in the automatic analysis and sorting of CAD models to gain information of interest, such as the maximum sizes, the number of drillings, the materials, or the different thread parameters of each element. With this extracted information, a categorization into classes, like turned parts or sheet metal parts, is enabled and realizable with a few mouse clicks. Another advantage is the complete integration of the program into the established CAD programs from different publishers.

The last considerable program is split up into several packages, programmed by the company CADENAS [32]. Software called PARTSolutions is designed for the management and influence of construction parts already in the creation phase, to limit all actions only to the necessary steps. The other package, called PURCHINEERING, treats all challenges arising with the purchase of the required construction parts.

In summary, it can be stated that many diverse approaches

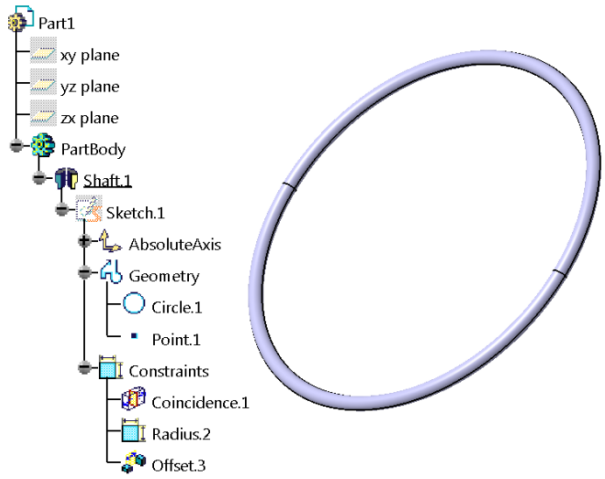


Figure 1. CAD-model and design tree of an o-ring in CATIA.

for the difficulty of CAD management already exist, and the development of specialized software depends on the given requirements for the particular cases of application.

3. Approach

As already stated in Section 1, the method presented here can be separated into three Python programs (norm.py, category.py, and match.py), which search with different strategies for CAD-parts in a given database. Table 1 shows in the dark grey segment a short description of these programs, as well as the input, which has to be given by the user, before the actual search can begin, and the output. The light grey segment consists of the preparation program (tree.py) that is necessary as an initializing step to create the XML-files.

3.1 Outsourcing the structure tree

To explain the meaning of this first preparation program tree.py, the situation mentioned in Section 1 should be reiterated. If the database of a large company, like e.g., a car manufacturer is considered, large amounts of CAD-models are stored in many imaginable patterns. A common method is a key number as a filename for every part, which is not very significant at first sight, but might be very substantial if it is decoded. Another conceivable warehousing strategy is a directory system with an arrangement into several categories, like departments, machine units, or construction dates. But also in such a case, the filenames of the CAD-models are probably not very informative, regarding the shape, size or function of each part. Usually, the constructor has to open each file manually, to obtain some detailed information about it, which is time and cost consuming. Especially, the search after a specific component using only visible filenames can be very sophisticated and intensive.

The program tree.py provides the conversion from automatically unreadable CAD-files, to easy examinable XML-files. As already mentioned, the structure tree of CATIA consists of many entries divided into a hierarchical order. All user actions, together with some additional information, are contained in it. Thus, the related models are replicable with a known structure tree; and without any other data an unambiguous recreation is ensured. The XML file format is adequate for the requirements of an extracted text-based structure tree. Each file is built by so-called tags, which are arranged in several levels, and can contain beside a meaningful name also attributes and element contents. With these preconditions, all the data a CAD-model consists of can be conveniently stored into a mentioned XML-file.

A good example of such a hierarchy is a point, which has three coordinate parameters, is a geometric primitive, and part of a sketch. In this sketch, some constraints are also included and it builds a foundation for a three-dimensional element. This element might also be a small part of many other elements that build the full body of a CAD-model. Already with this short example, the successive architecture of every file can be outlined.

Figure 1 shows a simple o-ring and the associated structure tree in CATIA. Here, the construction steps the user has conducted are recognizable. Only a small circle (Circle.1, Radius.2), with a center point (Point.1), at a certain distance (Offset.3) from the rotation axis, was necessary to create this shape. The command shaft (Shaft.1) has been chosen for the turn of the sketch around a selected axis. It is also observable that this file consists of only one body (PartBody). In more complex files, where e.g., Boolean operations are used, several bodies are needed.

At this point, the internal procedures of the program tree.py are only explained in summary form. During the processing of the program, Python accesses the running CATIA interface, opens every file that is contained in the given directory, and analyses the appropriate structure trees. The control of the program is very easy for the user. Before the outsourcing begins, he or she only has to specify one input folder, where one or many CAD-files are saved (the database), and one output folder, where the created XML-files should be stored. For the generation of the text-based XML-files the Python package Beautiful Soup was implemented into the source code. It can be labeled as a parser and is able to simplify the handling of XML structures.

In Figure 2 the XML-file of the o-ring is depicted. Beside the already explained structures, the purple specifications at the beginning and the black numbers in the middle are particularly remarkable. Here, it should be emphasized that the XML-file contains even more information than the structure tree holds at first sight. At the beginning, also the name of the file, the maximum measurements in the x-, y- and z-directions in mm, the volume in mm^3, the surface in mm^2, the centroid of the part from the absolute zero in mm, and the moment of inertia matrix, with nine entries in $kg*m^2$, are listed. In the middle part the coordinates of the point (the z-coordinate is not defined, because the sketch is two-dimensional), the radius of the circle, and the distance of the

circle from the axis of rotation are specified.

With the creation of one XML-file for every CATPart in the database, several advantages take place. On the one hand, the program runs completely independently and automatically. If huge amounts of files should be outsourced, the whole process might take a long time; but beside the provision of the input information at the beginning, human interaction is no longer necessary. Another great advantage is the redundancy of the expensive CAD-program after the generation of the XML-files. The information is then accessible with every text editor, and a license is not needed any more. The three search programs focus only on the analysis, and the evaluation of these XML-files. Because of this, tree.py can be labeled as a preparation program for the actual search.

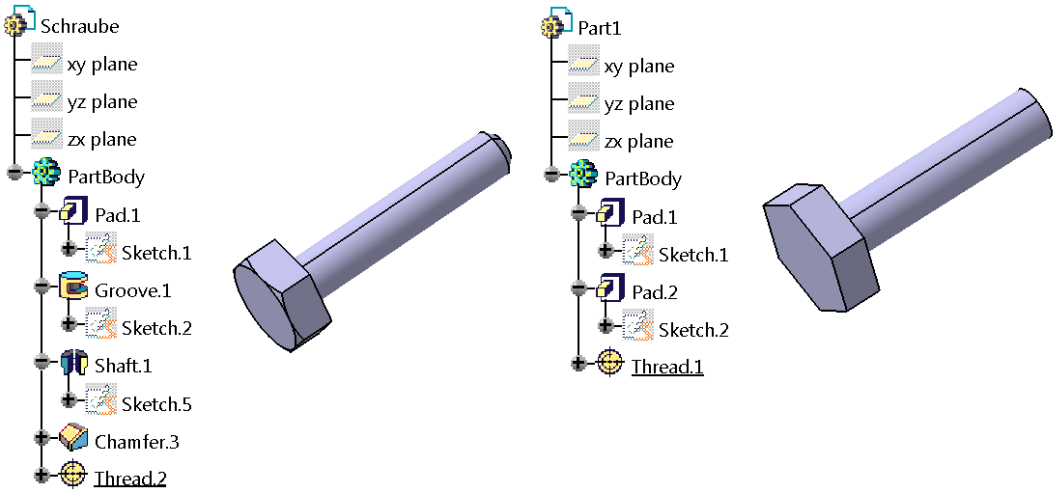
3.2.1 Searching for standard parts

In this section, the program norm.py is presented in detail. The intention is the retrieval of standardized parts in the well explained scenario of a large database. Standard parts are small and simple components, which are usually used for the mounting of assemblies. Often they are in accordance with norm systems, like the ISO (International Organization for Standardization), DIN (Deutsches Institut für Normung), or EN (Europäische Normen) standards. Until now, in the source code of the program the following eight standard parts are included: (1) Hexagon screws, (2) Cylinder head screws, (3) Square keys, (4) Hexagon nuts, (5) Dome nuts, (6) Washers, (7) O-ring seals, and (8) Disc springs.

The search algorithm is rather simple. To create these small standard parts with CATIA, there are always only a few ways to implement the shapes. With the first part in the list, the hexagon screw, the approach should be demonstrated. As it can be seen in Figure 3, a hexagon screw might have different sizes, and more or less varying structure trees. To the screw on the left hand side, some details like a chamfer at the shaft and a groove at the top have been added. But for a hu-

```
<treeview>
 <PartDocument
 name="O-Ring.CATPart"
 sizex="82.775"
 sizey="82.775"
 sizez="3.0625"
 volume="1776.5287922"
 surface="2368.70505626"
 centroid="(
 5.1195044277815595e-15,
 2.047801771112624e-15,
 6.3993805347269499e-17)"
 momentofinertia="(
 1.4237212773708933e-06, 0.0, 0.0,
 0.0, 1.4237212773708933e-06, 0.0,
 0.0, 0.0, 2.8454439598505666e-06)">
  <AxisSystems>
  <bodies name="Hauptkörper">
   <Shaft name="Welle.1">
    <Sketch name="Skizze.1">
     <GeometricElements>
      <AbsoluteAchse>
      <Kreis.1>
      <Punkt.1>
       (40.0, 0.0, None)
      </Punkt.1>
     </GeometricElements>
     <Constraints>
      <Kongruenz.1>
      <Offset.3>
       40.0
      </Offset.3>
      <Radius.2>
       1.5
      </Radius.2>
     </Constraints>
    </Sketch>
  </bodies>
  <GeometricalSets>
 </PartDocument>
</treeview>
```

Figure 2. Outsourced XML-file of the o-ring.



(a) (b)

Figure 3. Comparison of two standard hexagon screws.

man observer, both parts are clearly identifiable as screws at first sight, because both have a hexagonal screw head, a shaft, and a thread (not visualized in CATIA) in common.

After the user has chosen which standard parts he or she wants to search for, the program analyses every XML-file in the database. Then the features saved in the structure tree are inspected. Usually, every three-dimensional feature contains at least one two-dimensional sketch. Instead of writing out the values or the positions of the geometric primitives, the program identifies the quantity of components, like points, lines, or circles, as well as their constraints, like coincidences, parallelisms, or perpendicularities. Also, the number of three-dimensional features, like pads, threads, shafts or roundings is counted (cf. Section 3.2.2).

To produce e.g., the shaft of a screw, basically just two possibilities indicated in Figure 3 exist. Either it is generated by one circle, which is extruded along the axis of the screw (Figure 3, right hand side, Pad.2); or a rectangle, which is as long as the shaft, is rotated around its axis (Figure 3, left hand side, Shaft.1). The program norm.py takes both possibilities into consideration. The same principles are valid for the screw head. Either the constructor can use the CATIA command for polygons in the basic sketch, or he models a hexagon consisting of six lines, on his or her own. Again, both ways are recognized by the search algorithm.

If a part in the database should be detected as a hexagon screw, the following three preconditions must be implied: (1) The part must own the screw head realized with one extrusion and one sketch. To gain the hexagonal shape of the sketch, either the CATIA command for polygons could have been used, or the creation was performed manually. (2) The part must own a shaft created by the command Shaft or the command Pad. (3) The part must own exactly one thread.

The parts in Figure 3 are both identified as hexagon screws by the program, because all the mentioned conditions are fulfilled. With this strategy, the sizes and other details, like chamfers or roundings are not important, and can be neglected.

The algorithms for the discovery of the other standard parts listed above work in a similar way, and need not be explained in detail. But also simpler parts, like square keys or o-rings (cf. Figure 1), which are only generatable in one fixed way, are handled.

The program norm.py exploits the fact, that for the users of the CAD-software, there are not many ways to retrieve the simple shapes of standard parts. To gain a three-dimensional feature of a specific form, usually the command Pad is used, which is equal to an extrusion of a shape into the third dimension. With this command, many standard parts, like e.g., the here presented washers and square keys, can already be generated. Another important command of CATIA is Shaft. It is mostly used to produce round parts that are created by the spinning of a profile around an axis.

In fact, the mentioned shape of an ordinary washer can be gained either by the command Pad, or the command Shaft. If the command Pad is used, the designer has to draw two concentric circles into one sketch, and extrude it into the third dimension. With the command Shaft, a small rectangle has to be drafted into one sketch, and rotated around an axis that has a certain distance from this rectangle. All other ways to retrieve the simple shape of a washer need more than one three-dimensional command, and are therefore not regarded by the program norm.py. Only these two described possibilities are considered.

On the one hand, the algorithm is not very flexible with the analysis of other approaches, but on the other hand, the process of rediscovery of such standard parts is drastically simplified. All the program has to do is to either look for the pattern of the command Pad, which includes two concentric circles in one sketch in the XML files, or the pattern of the command Shaft that includes the rectangle, which is built up by four points and four lines in one sketch. If a standard part possesses some other details, like e.g., an imprint or a fin, all geometric primitives that are needed to create that detail are neglected.

This kind of algorithm works only for simple shapes that are fulfilled by standard parts. Every standard part that is contained in the software needs its own source code, and its own consideration. If in the future, automated searches for more standard parts should be implemented, every possible approach for the construction has to be analyzed first. If more complex engineering components are considered, too many ways for the creation performed by the designer are possible, and it would be speculative to detect those kind of parts only by their number of features.

3.2.2 Personalized user queries

This second subsection deals with the program category.py, which focuses on search parameters that are entered by the user. Internally, the structure trees and the XML-files, respectively, of all parts in the database are examined, and again in particular the number of commands that have been necessary to create the parts are determined. To decide automatically whether a part matches with the user input or not, it is compared to certain value ranges. These can be entered when the program is started.

At first, the user has to decide how many categories should be taken into consideration during the search. In the next step, it has to be chosen, in which categories should be investigated. Finally, the ranges of every category in their appropriate units are necessary, to complete the input information. If the corresponding parameter of the current part is within these user-defined limits, the query counts as a match. If the user defined the parameters for several categories, all checked categories must be valid, to find an overall match displayed in the shell. For proper results, the input has to be adapted to the physical units of each category.

The following list shows all 56 categories, and their related units:

1. Size in x-direction in mm
2. Size in y-direction in mm
3. Size in z-direction in mm
4. Moment of inertia xx in kg*m²
5. Moment of inertia xy in kg*m²
6. Moment of inertia xz in kg*m²
7. Moment of inertia yx in kg*m²
8. Moment of inertia yy in kg*m²
9. Moment of inertia yz in kg*m²
10. Moment of inertia zx in kg*m²
11. Moment of inertia zy in kg*m²
12. Moment of inertia zz in kg*m²
13. Shape in min/max to min/mid
14. Minimum size in mm
15. Maximum size in mm
16. Volume in mm³
17. Surface in mm²
18. Volume/surface in mm
19. Number of sketches in quantity
20. Number of points in quantity
21. Number of lines in quantity
22. Number of circles in quantity
23. Number of splines in quantity
24. Number of ellipses in quantity
25. Number of pads in quantity
26. Number of shafts in quantity
27. Number of grooves in quantity
28. Number of pockets in quantity
29. Number of ribs in quantity
30. Number of slots in quantity
31. Number of holes in quantity
32. Number of threads in quantity
33. Number of chamfers in quantity
34. Number of edge fillets in quantity
35. Number of geometrical sets in quantity
36. Number of bodies in quantity
37. Number of coincidences in quantity
38. Number of fixations in quantity
39. Number of concentricities in quantity
40. Number of tangencies in quantity
41. Number of parallelisms in quantity
42. Number of perpendicularities in quantity
43. Number of symmetries (constraint) in quantity
44. Number of angles in quantity
45. Number of lengths in quantity
46. Number of distances in quantity
47. Number of radiuses in quantity
48. Number of equidistant points in quantity
49. Number of translations in quantity
50. Number of rotations in quantity
51. Number of thickened surfaces in quantity
52. Number of symmetries (design) in quantity
53. Number of multi-sections surface in quantity
54. Number of automatic fillets in quantity
55. Number of constraints in quantity
56. Number of geometric elements in quantity

To achieve meaningful results, the user of the program should have an idea which kind of parts he is looking for. An easy separation strategy is e.g., the search by minimum (Category 14) or maximum (Category 15) size. A demonstrative example are sheet metal parts, which might have a large maximum size, but a very small minimum size.

The number of points, lines, and circles of one part might be very significant for the complexity of the sketches. If a model only has one extrusion based on one sketch, but many points and lines, this can be an indication of a relatively sophisticated shape.

Another helpful search parameter is the shape in Category 13. For the computation of the range, the minimum, medium, and maximum sizes are extracted from the XML-file, and the mentioned parameters are calculated. The sizes are saved in mm, so that the parameters have no physical units.

At this juncture, a small example of the detection of o-rings, shown in Figure 1, is explained in brief. Even if the structure tree of the o-ring is not known to the user, he or she can guess that only one sketch with exactly one circle is necessary to produce the shape with the command shaft. So the user can choose four categories (19. number of sketches, 20. number of points, 22. number of circles, and 26. number of shafts) for his or her search. The range of every category is selected as exactly one, because only one sketch with one point and one circle is necessary to produce the shaft. Of course, there is a certain probability that not all detected parts are o-rings; but even with only four search parameters, the approximation to an o-ring is quite good.

At this point, a more complex example should also be demonstrated. In Figure 4, some kind of crank is depicted, which is at this point named stabilizer. The construction of a

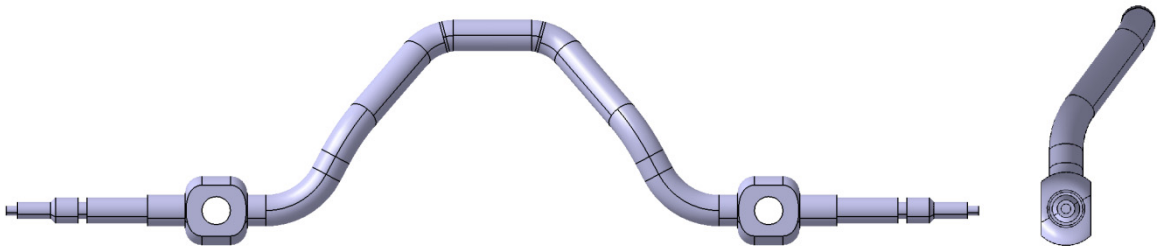


Figure 4. Front and side view of a stabilizer.

part like this is much more complex than the mentioned standard parts, but the search engine can benefit exactly from these kinds of part properties. In this example, 25 sketches (cf. Category 19) have been necessary to gain the final shape. Due to the fact that this part is symmetric, only the details on one side have been created, and then mirrored at the middle. Although the bounding box (cf. Categories 13 and 18) is quite large in all directions, it is not filled with much material. Thus, compared to the volume of the bounding box, this component is very light, and the volume is very small (cf. Category 16). A few commands that are seldomly used, like Rib (cf. Category 29), are included in the structure, tree as well as in the respective XML-file.

If the user has some knowledge about all the mentioned properties, it should be easy to define the search parameters the program category.py is using in a unique way, to sieve out this stabilizer, or similar parts from a large database. If there are components that contain similar features, of course the query may deliver additional results, but all stabilizers and engineering parts with similar properties will definitely be displayed in the output.

The usage of this program and its accuracy becomes certainly more difficult, the more complicated the sought parts are; but with a precise idea of the desired results in advance of the search process, many parameters should be utilizable for the detection of any kind of parts. After the computation is completed, the names of the located files are displayed in the shell.

3.2.3 Detection of similar parts

The third program match.py is comparable with the publication of Linghao et al. [33]. They present a method that analyzes two similar CAD-files, and indicates all differences to the user. In this way, e.g., all changes in a construction update are recognizable at first sight. The program match.py can instead be used for the detection of files in a database that are identical, or similar to, a reference file. The name of this reference has to be supplied by the user at the beginning. When the search process is started, the source code analyses the XML-file of the reference, in all the categories listed in Section 3.2.2. Then, the same happens with the XML-files of every part in the database.

The gradation of the matched parts is separated into the three classes: (1) Perfect match, (2) Very good match, and (3) Good match. Especially, the first class is very easy to find. If in the database, a XML-file is detected that is equal to the XML-file of the reference part, a perfect match is located. More complicated is the answer to the question, how similarities and differences could be identified.

The following scenario should clarify the problem. Figure 5 illustrates three simple CAD-models. On the left hand side the reference part is shown. To both other parts, in the middle and on the right hand side, only one feature has been added (middle: a small rectangular pocket, right: a large rectangular pad). But nevertheless, the part in the middle can be identified as a very similar match, compared to the reference part. Despite the fact that to the model on the right hand side, also only one feature has been added, this one feature changed the appearance of the part heavily, and it cannot be interpreted as a very good, or a good match any more. So the program match.py has the purpose of locating such differences, and only finding perfect, very good, or good matches to the reference. All other parts are skipped, and not displayed for the user.

If the structure trees of the parts shown in Figure 5 are considered, an easy mathematical distinction between the different categories can take place. Of course, the characteristic physical values of both updated parts has been changed e.g., in volume (Category 16), and surface (Category 17). Also, the number of graphical elements, like sketches (category 19), points (Category 20), lines (Category 21), and constraints (Category 55) has been modified, compared to the original part. The only obvious variation between a very good match (middle part) and a bad match (part on the right hand side) can occur through a big difference in sizes (Categories 1, 2, 3, 13, 14 and 15). While the part in the middle has exactly the sizes of the reference, the measurements of the model on the right are much larger, compared to the original dimensions.

The program match.py takes advantage of this fact. The deviation of every category is measured by the analysis of both values. If the divergence is very small, an internal counter is raised by the value of one. If all categories of the examined part are identical with the reference, the part is classified as a perfect match. If all categories except three or less are identical, the part belongs to the category with very good matches. All parts with similar categories between three and five count only as good matches.

Also, the problem of a similar category is not easy to solve.

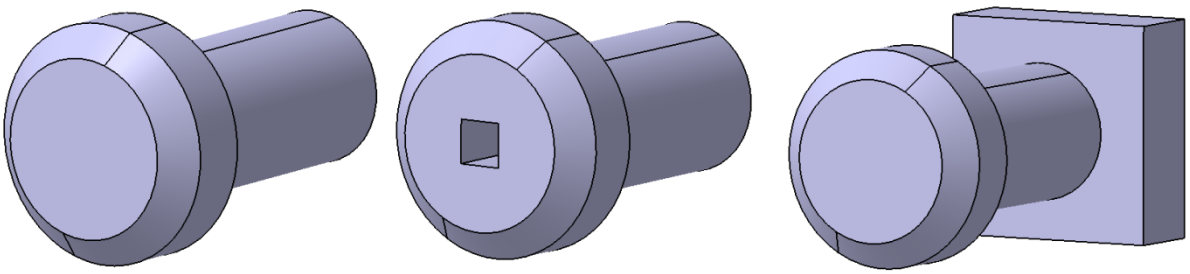


Figure 5. A reference part with two similar matches.

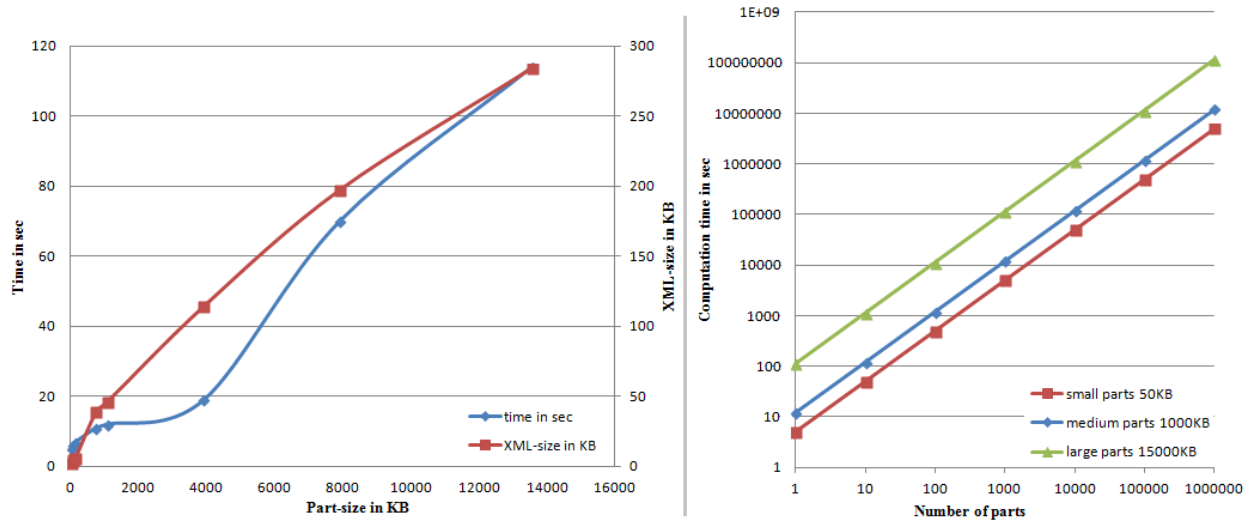


Figure 6. Performance of tree.py.

If for example, the maximum measurements are considered, a variety in the range of 5 mm is identified as a similar match, and the counter is raised by one. Thus, a potential candidate for a very good match might be 5 mm smaller or larger in every direction, to count as a similarity. The same principle is used for the categories dealing with geometric primitives. The number of points can vary by three, while the number of lines might differ not more than by two, to count as a similar category.

If again the stabilizer shown in Figure 4 is considered as a reference part and in the database another component with the same dimensions and outer shape is contained, which has only the difference that the through-hole drillings do not exist, the algorithm detects that component as a good match. The internal approach of the source code is exactly the same, as for simple parts. Of course, the volume and the surface are a little bit different, but, compared to the same sizes and the large dimensions, the variation is very small. Due to the mirroring, also the number of sketches will increase only by one, as well as the number of circles. Nevertheless, it will not change the fact that the overall similarity between both parts is quite large, and the program will identify a stabilizer with filled holes as a good match.

With this approach, the variation of CAD-models in shape and other physical properties has been transferred to a mathematical analysis system, which allows an automatized comparison of technical components. Also this program runs, after the user input, independently and without any further human assistance.

4. Performance

In this section, the performance of the programs is discussed. For the intended application in industry with large amounts of data, the running times, the necessary computation capacities, and manpower for handling are important. Exact results, after an elaborate test of every program with many CAD-models, are presented by Vadlamani [34]. Here in this publication only some keywords are mentioned.

The first important characteristic is the ability for an independent processing of every program. That implies that for the control of the software, no human-computer interaction is needed. The only necessary operation is the start preparation, before the actual process of examination takes place. Of course the path of the CAD-models and the folder of the database, respectively, must be entered into the program. Then, no matter how many parts have to be machined down, each of the four presented programs runs without any further assistance.

By far the most complex operations are executed by tree.py, and therefore it is the most time-consuming program. As already mentioned in Section 3.1, every part has to be opened in CATIA, and the physical properties, as well as the structure trees, have to be outsourced to the XML-files. Thus the speed of the processing depends on the connection between CATIA and Python, and the execution times of the different commands. Some tests with a only few parts showed that the outsourcing of ten parts takes approximately one minute, depending on the complexity of the models. Starting with this parameter, the transfer of one million parts from CATPart- to XML-file would take about 70 days of nonstop processing, if only one computer is used. The performance values of whole datacenters, including the application of parallel algorithms, would be of great interest.

In Figure 6, two diagrams are shown, which visualize the performance of the program tree.py. The left one shows, with the help of the left ordinate, the processing time the program needs to create the XML-file for one part, of the size depicted on the abscissa. The blue graph shows the behaviour of the software, depending on the increasing memory space in kilobyte. The ordinate on the right hand side contains an indication of the sizes of the created XML-files, also depending on the input file. Here it can be recognized that the XML-file

only needs a fraction of the space that is necessary for the initial CATPart-file.

The second diagram on the right has been created to show the performance for large databases. The scale on both axes is logarithmic, and it shows the computation time in seconds, over the number of input parts changed to XML-files. The three graphs indicate three different part sizes that are assumed to be constant for all the parts in the database. In reality, usually a mix of small and large parts is contained in a database of e.g., a company, and the behaviour of the program would indeed range between the red and the green line. At this point, it should also be considered that these results are related to one computer. If a company is transferring a database of several million parts to XML-files, not one, but many computers should be used, which might have even higher efficiencies.

The other programs norm.py, category.py, and match.py are all based on the investigation of the XML-files. These are only text-based and therefore easily searchable. Due to the quantity of only a few seconds for many parts, the processing time can be neglected, and only the indication and the printout, respectively, of maybe thousands of results might take some capacities. The reason for the fast execution is caused by the size of the XML-files. While the original CATPart-files are many kilobytes in size, the XML-files only need fractions of this memory. The o-ring in Figure 1 is of 59KB size. The respective XML-file is of just 2 KB. Larger and more sophisticated CAD-models might reach a memory of several megabytes, with correspondingly sized XML-files.

The CATPart-file of the stabilizer depicted in Figure 4 occupies 559 KB, while the XML-file is only 28 KB large. Nonetheless, the computation times of all three search programs are not the critical values. Due to the text-based search, the investigation of the XML-files in a large database takes for each program and one related query only a fraction of the time the program tree.py needs, and can therefore be disregarded. Even if a user needs to start several attempts to find a product group with the program category.py, the algorithm only needs seconds to a few minutes, always depending on the number of XML-files.

In summary, the usability of the suggested software for industrial application can be evaluated positively. Even if huge data should be treated, the automatic execution allows an independent accomplishment of all programs.

5. Comparison

At this point, the advantages, as well as the disadvantages of every program should be considered. The greatest benefit of the search engines is their independence from the CAD-software, as long as the outsourcing of any needed information has taken place previously. Thus, the usage of a license, which is quite expensive, is not necessary any more, after the process. As already mentioned in Section 4, the processing times, and therefore the industrial applicability, is similar for every search program, and can be disregarded.

Nevertheless, the properties of the three search programs depend on the location purposes of data. The scope of the first program norm.py is self-explanatory. If the user wants to filter special standard parts from a large database, all three programs could be used; but for more exact results, the first one is preferred. A big handicap is the search algorithm, which examines only the geometric primitives, the features of the CAD-model, and the physical properties of the parts, but not the actual shape, or the quality of the numeric values. As a result, also parts could be matched, which are not searched intendedly. The applicability of this search strategy becomes more prone to errors, the more complicated the parts are. Also, the adding of more details to a standard part could be problematic, because simultaneously the structure tree gets filled with more sketches and in particular constraints (cf. Figure 3).

The usage of the second program category.py can also become more or less difficult. To search with many categories and parameters at the same time, an exact vision of the desired results should exist, before the actual query is executed. If e.g., the number of certain features are not known in the first place, the search parameters have to be coarsened, which leads to a poor filtering, and thus to too many results. Of course, also the structure of the database plays a key role. If a company is specialized in a certain product group, many CAD-models might look quite similar. Then, the number of features and the whole structure trees will appear with the same characteristics, and a meaningful sieving becomes more complex. But the program category.py is still a powerful tool. If too many results are displayed, an additional refined iteration might become necessary.

The fields of application for the third program match.py are also widespread. Of course, a big disadvantage is the necessity of a reference part. If a special component is to be located, and only the shape of the part, but not the model is known, the program category.py has to be preferred. Another dubious factor is the mechanism for the recognition of similarities between the reference and the current part in the database. If the user wants to locate less similar models, one option could be a manual adjustment of the ranges described in Section 3.2.3. In the next segment, the possible tasks for an improvement of this software are named.

6. Future work

A suitable possibility for the expansion of the software presented here is certainly the transfer to other popular CAD systems. CATIA has been chosen, because it provides a convenient connectivity and automatability with Python. In spite of that, software like Inventor, NX, or Pro/E possesses similar functionalities, and the ideas of the methods are also realizable with other programs. Also, for the usage of the XML standard, some alternatives exist. An important aspect should be the improvement of the applicability in industry with huge

amounts of data, and the reduction of the processing times, and memory capacities. The keyword of parallel computing has already been mentioned, to represent one possible attempt. Once again, to the documentation of Vadlamani is referred to [34], in which the source codes of all programs have been analyzed.

Furthermore, every program could be improved and adjusted, as well. While the improvement of the computation time should be focus on tree.py, the first search engine norm.py could easily be extended by the addition of other standard or more complex parts. Here especially, the several ways of construction of one part could be considered, and analyzed in detail, to realize more ways of implementation.

The program category.py is less extensible, but the search strategies could be developed in detail. The most complicated, and therefore most analyzable program is match.py. The automatic recognition of features and similarities, respectively, can be considered as research areas in themselves, where many other possibilities could be taken into account. One alternative is support by the analysis of two-dimensional images of every part. The overall target is the automatic classification of CAD-models, with many alternatives to search engines.

7. Summary

In this publication, the state of the art of research areas like data mining and automatic feature recognition has first been presented. Beyond this, a method consisting of four comprehensive programs that can be separated into one preparation code, and three search engines for different fields of application, have been described. For the implementation of the strategies, the CAD-software CATIA V5 and the open source programming language Python have been used. The analysis of the performance, as well as the advantages and the disadvantages of every program, concludes that the intended industrial applicability with large amounts of data is indeed realizable and economical, in relation to the time, memory capacities, and necessary manpower. Nevertheless, all programs are software prototypes, which are in certain cases fault-prone and improvable.

References

[1] Fayyad UM, Piatetsky-Shapiro G, Smyth P. Advances in knowledge discovery and data mining. 1st ed. Menlo Park: American Association for Artificial Intelligence; c1996. Chapter 1, From data mining to knowledge discovery: an overview; p. 1-34.

[2] Fayyad UM, Piatetsky-Shapiro G, Smyth P. From datamining to knowledge discovery in databases. Artifical Intelligence Magazine. 1996; 17(3): 37-54.

[3] Frawley WJ, Piatetsky-Shapiro G, Matheus CJ. Knowledge discovery in databases: an overview. Artificial Intelligence Magazine. 1992; 13(3): 57-70.

[4] Petersohn H. Data mining - verfahren, prozesse, anwendungsarchitektur. 1st ed. Munich: Oldenbourg Wissenschaftsverlag GmbH; 2005. 342 p.

[5] Gorunescu F. Data mining - concepts, models and techniques. 1st ed. Berlin Heidelberg: Springer Verlag; 2011. 357 p.

[6] Hilderman RJ, Hamilton HJ. Knowledge discovery and interestingness measures: a survey. 1st ed. Regina: Department of Computer Science, University of Regina; 1999. 27 p.

[7] Vajna S, Weber C, Bley H, Zeman K. CAx für ingenieure - eine praxisbezogene einführung. 2nd ed. Berlin Heidelberg: Springer Verlag; 2009. 564 p.

[8] Kornprobst P. Catia v5 - volumenmodellierung grundlagen und methodik in über 100 konstruktionsbeispielen. 1st ed. Munich: Carl Hanser Verlag; 2007. 300 p.

[9] Ester M, Kriegel H-P, Sander J. Spatial data mining: a database approach. In: 5th International Symposium on Large Spatial Databases; 1997 Jul 15-18; Berlin, Germany; p. 47-66.

[10] Haffey M, Duffy A. Knowledge discovery and data mining within a design environment. From Knowledge Intensive CAD to Knowledge Intensive Engineering. 2002; 79: 59-74.

[11] Angkasith V. An intelligent design retrieval system for module-based product. 1st ed. Columbia: Faculty of the Graduate School, University of Missouri-Columbia; 2004. 269 p.

[12] Kizu H, Yamamoto J, Takeda T, Gyohten K, Sueda N. 2D CAD data mining based on spatial relation. In: 10th International Conference on Document Analysis and Recognition; 2009 Jul 26-29; Barcelona, Spain; p. 326-330.

[13] Harding JA, Shabaz M, Srinivas S, Kusiak A. Data mining in manufacturing: a review. Journal of Manufacturing Science and Engineering. 2006; 128(4): 969-976.

[14] Babic B, Nesic N, Miljkovic Z. A review of automated feature recognition with rule-based pattern recognition. Computers in Industry. 2008; 59(4): 321-337.

[15] Iyer N, Jayanti S, Lou K, Kalyaranaman Y, Ramani K. Three-dimensional shape searching: state-of-the-art review and future trends. Computer-Aided Design. 2005; 37(5): 509-530.

[16] Min J, Bowyer KW. Improved range image segmentation by analyzing surface fit patterns. Computer Vision and Image Understanding. 2005; 97(2): 242-258.

[17] Cucchiara R, Lamma E, Mello P, Milano M, Piccardi M. 3D object recognition by VC-graphs and interactive constrain satisfaction. In: 10th International Conference on Image Analysis and Processing; 1999 Sep 27-29; Venice, Italy; p. 508-513.

[18] Flynn PJ, Jain AK. CAD-based computer vision: from CAD models to relational graphs. Transactions on Pattern Analysis and Machine Intelligence. 1991; 13(2): 114-132.

[19] Cybenko G, Bhasin A, Cohen KD. Pattern recognition of 3D CAD objects: towards an electronic yellow pages of mechanical parts. International Journal of Smart Engineering System Design. 1997; 1(1): 1-13.

[20] Ip CY, Regli WC. Manufacturing classification of CAD models using curvature and SVMs. In: International Conference on Shape Modeling and Applications; 2005 Jun 15-17; Cambridge, MA; p. 363-367.

[21] Horváth L, Rudas IJ. Self adaptive product definition using captured specification and knowledge. In: 5th International

Symposium on Logistics and Industrial Informatics; 2013 Sep 5-7; Wildau, Germany; p. 17-22.

[22] Min P, Halderman JA, Kazhdan M, Funkhouser TA. Early experiences with a 3D model search engine. In: 8th International Conference on 3D Web Technology; 2003 Mar 09-12; St. Malo, France; p. 7-18.

[23] Wei L, Yuanjun H. Representation and retrieval of 3D CAD models in parts library. International Journal of Advanced Manufacturing Technology. 2008; 36(9-10): 950-958.

[24] Ansary TF, Daoudi M, Vandeborre J-P. A Bayesian 3-D search engine using adaptive views clustering. Transactions on Multimedia. 2007; 9(1): 78-88.

[25] Princeton University [Internet]. Princeton: Princeton University, Computer Science Department, Graphics & Geometry Group, Princeton Shape Retrieval and Analysis Group; c2013 [cited 2013 Nov 29]. Available from: http://gfx.cs.princeton.edu/proj/shape/

[26] Min P. A 3D model search engine. 1st ed. Princeton: Princeton University, Department of Computer Science; 2005.139 p.

[27] Funkhouser T, Min P, Kazhdan M, Chen J, Halderman A, Dobkin D, Jacobs D. A search engine for 3D models. Transactions on Graphics 2003; 22(1): 83-105.

[28] Michalik P, Štofa J, Zolotová I. Testing the properties of k-means algorithm for data mining applications. In: 5th International Symposium on Logistics and Industrial Informatics; 2013 Sep 5-7; Wildau, Germany; p. 99-102.

[29] Wang H-P, Chang H. Automated classification and coding based on extracted surface features in a CAD data base. International Journal of Advanced Manufacturing Technology. 1987; 2(1): 25-38.

[30] Siemens [Internet]. Cologne: Siemens Geolus Search, Search Engine for 3D Data; c2013 [cited 2013 Nov 29]. Available from: http://www.plm.automation.siemens.com/de_de/products/open/geolus/index.shtml

[31] Lino [Internet]. Mainz: Lino simus classmate, Software für das Daten-Prozess-Management (DPM); c2013 [cited 2013 Nov 29]. Available from: http://www.lino.de/simus-classmate.html

[32] CADENAS [Internet]. Augsburg: CADENAS, Strategisches Teilemanagement, Purchineering, Geometrische Ähnlichkeitssuche; c2013 [cited 2013 Nov 29]. Available from: http://www.cadenas.de/produkte

[33] Linghao Z, Dongming G, Hang G. A method to analyze the difference of 3-D cad model files based on feature extraction. Journal of Mechanical Science and Technology. 2011; 25(4): 971-976.

[34] Vadlamani S. A performance validation and improvement of three different search engines for CAD-models using Python and CATIA V5. 1st ed. Wuppertal: University of Wuppertal, Department of Mechanical Engineering Informatics; 2014. 32 p.