Title:
CAD Refactoring and the Art of Computer Aided Design Model Maintenance

Authors:
Peter Rosso, peter.rosso@bristol.ac.uk, University of Bristol, UK
James Gopsill, james.gopsill@bristol.ac.uk, University of Bristol, UK
and Centre for Modelling and Simulation, UK
Stuart Burgess, stuart.burgess@bristol.ac.uk, University of Bristol, UK
Ben Hicks, ben.hicks@bristol.ac.uk, University of Bristol, UK

Keywords:
Computer Aided Design (CAD), Editability, Graph Representations, Technical Debt, Refactoring

DOI: 10.14733/cadconfP.2022.339-343

Introduction:

Solid modelling is a common way to communicate and store geometric information. Solid models allow drafters to abstract complex constructs enabling them to be better understood and shared. Leveraging the computer-based nature of models affords continuous improvement and refinement of models via version control and the ability to store the history of a model's evolution. The abstractions also support the use and reuse of part geometry resulting in more efficient, less error-prone and compressed product development cycles [9, 12]. However, with particularly long-life products, solid modelling still poses challenges for the long-term management of product data (among which geometry data). Kasik et al. [5] discusses a range of challenges for which two underlying themes of interoperability and reuse are evident.

In addition to the two longer-term underlying themes, the type of abstraction applied by the engineer during the initial design of the CAD model will also impact current activities. For example, artefacts, such as intersections between surface geometry, can cause issue in the subsequent generation of Computer Aided Manufacturing (CAM) code and/or meshes for simulation. Given that geometry may be constructed with different design intent [7], it is important to understand which construction can be reused at a lesser cost. A consequence will be alleviating the drafter of further cognitive load, so that the drafter can continue to design in the way that works best for them - or they believe works best. The subject of changing structure of a system/code base without affecting its external behaviour is of great interest in programming and is referred to as refactoring. This field also cites the need for refactoring to simplify source code for long-term maintenance and support, providing underlying performance enhancements introduced by new techniques and copying with the growing complexity of a project. All of which are analogous to the issues and aims of engineers/engineering management.

Given the parallels in both the problem and solution, the authors propose refactoring as an approach to re-structure the relationships between design entities in a CAD model. The following sections discuss relevant literature in CAD, Technical Debt, Refactoring, and Graph Representation which provide the foundations for CAD refactoring. Throughout the sections the same artefact is represented in the context of the topic discussed.

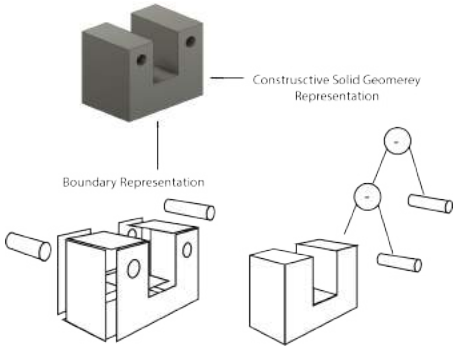

Fig. 1: BRep and CSG Model Representations

CAD:

Solid modelling continues to evolve, adopting different representations to match engineering needs and growing technological capabilities in order to include as many useful details as possible in the chosen digital representation of an artefact. The most common CAD representations are evolutions of Boundary Representation, Constructive Solid Geometry and Feature Based representations - Fig.1. "A Boundary Representation (BRep) of an object is a geometric and topological description of its boundaries. The object boundary is segmented into a finite number of bounded subsets, called faces. A face is represented in a BRep by its bounding edges and vertices' [1]. Constructive Solid Geometry (CSG) defines a family of schemes for representing solids as Boolean combinations of predefined volumetric primitives [11]. On the other hand, Feature Based (FB) representations depict a part or an assembly in terms of its constituent features. FB models are created by organising the constituent features in a structure that expresses the inter-relationships [4]. FB representations also allow the inclusion of Design intent in a CAD model. Design intent encapsulates sufficient knowledge regarding the manner in which the drafter generated the model to permit it to be modified by the original constructional procedure [10]. Rosso et al. [7] analysed the variability in design intent and CAD model building process which leads to the conclusion that the difference in structure might lead to files which are more or less editable. The accumulation of less editable files leads to an accumulation of what is referred to as Technical Debt that needs to managed by a engineering organisation.

Technical Debt:

Technical Debt (TD) is the dept that accrues when you knowingly or unknowingly make the wrong or non-optimal design decision [8]. TD is either repaid or leads to technical bankruptcy. Cunningham [2], talking about software development, introduces the metaphor of "debt" in the context of early shipping, where code at lower quality might be shipped to meet deadlines. Going into dept is a trade-off for faster growth and development. Short term solutions or quick responses to growing problems leads to non optimal solutions which are likely to lead to a growth in TD. TD can be observed across engineering from drafting, CAD, CAM, simulations and reporting. While in code the idea of smells identifies patterns that can be symptoms of TD, in CAD there are CAD smells [6]. Fig.2 portrays alternative constructions of the same artefact. While Fig.2(a) might be using too many sketches, Fig.2(b) and (c) offer alternative constructions using less sketches, by either condensing them, or using mirroring. Good organisation of code promotes good user behaviour by incentivising reusable and easily editable elements[3], seemingly

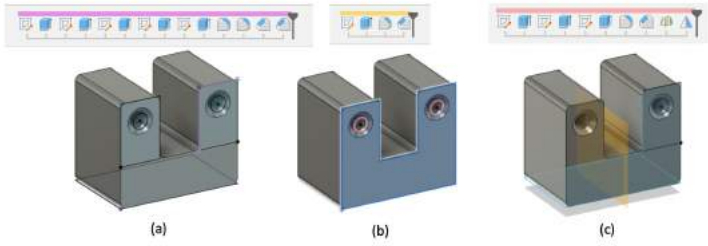

Fig. 2: Artefact modelled with different design intent.

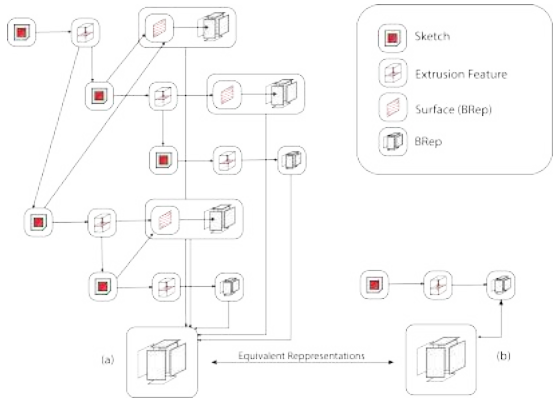

Fig. 3: Graph Representation of the example file for construction (a) and (b) in Fig.2

the same could be applicable to CAD given it's simiilarities to OOD [6]. It is also important to remember that poor organisation encourages bad user behaviour by either: disincentivising reuse, or by inciting "quick and dirty" solutions. In programming this problems is addressed using refactoring techniques.

Refactoring:

Refactoring is a form of restructuring which constitutes a change in the internal structure (to make it easier to understand and cheaper to modify, for example) without changing its observable behaviour [3]. Refactoring is performed in an incremental process that changes the structure of a model in steps. After each step, tests are performed to verify the observable behaviour has not changed. Refactoring covers a broad range of methods that act as treatments for different smells of poor coding practice. Smells can be classified in different groups that can be addressed individually and / or collectively [8] leading to a association between a smell and a correspondent refactoring method. A model can exhibit a combination of smells, and consequentially need a series of refactoring methods applied to it, verifying the external behaviour is the same at each step. While the relationship between elements in code is explicit, CAD has some implicit relationship which, at times, the drafter does not consider. A graph representation enables the visualisation of both implicit and explicit relationships.

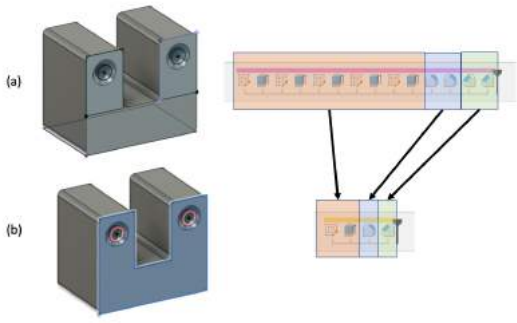

Fig. 4: Refactoring of construction (a) (see Fig.2a) to construction (b) (see Fig.2b)

Graph Representations:

A graph is defined as set of points, called nodes, joined by lines, called edges; each edge joins exactly two nodes. Graph representations have been used in depicting and addressing a variety of problems where there is a large degree of connectivity between entities. However, the connectivity often leads to increased computational complexity and it is only recently that advances in computation have enabled the deployment/production use of graph databases.Hypergraph and/or Typed-Attributed graphs are some of these representations that allow the embedding of more complex relationships between entities. A CAD model relies on a dense network of relationships of different types across different entities. An hypergraph structure will be able to encapsulate all of these relationships within the same representation. Representing relevant information in a graph model enables the user to utilise tools such as graph morphism, and to consider relationship across multiple representations of the same CAD model. At its heart, refactoring is a horizontal, exogenous transformation which can be applied to graphs where there is sufficient information to confirm that the CAD model before and after the process is still the same.

Refactoring with graph representations:

The methods offered by refactoring vary in complexity and in what entities they can move, merge, separate, etc. Simpler methods can be applied manually on the model, while more complex ones which need access to a larger number of variables can be implemented programmatically using graph transformations. Throughout this paper, the same simple model was presented using different representations Fig.1 and showing different design intent Fig.2. The graph representation in Fig.3 shows two different graphs of the same artefact; it is possible to move between these two representation using appropriate graph morphisms guided by pertinent constraints. Refactoring preserves observable behaviour, which is the behaviour that the user is interested in and impacts the system's output. This idea allows for changes in characteristics such as performance [3]. The observable behaviour of a CAD model relevant for a drafter and in this context it's the validity of the model in representing the associated artefact. It is possible to ensure this by maintaining the BRep for the file, only changing the design intent. The constraint of representing the same artefact is analogous to the constraint set in refactoring: "no change in external behaviour". Fig.4 shows two graphs that are the are associated with the same artefact. These two models are associated with the constructions in Fig.2 (a) and (b). The graph is able to provide sufficient information regarding the relationship of different elements in the boundary representation and design intent to refactor the model construction tree. Graph representations can bring all of these relationship on the same level of abstraction aiding more complex refactoring using existing methodologies.

Conclusions and further work:

This paper introduces the use of Refactoring for CAD models. Rosso et al. [6] discusses the parallel between CAD and OOD, consequentially some of the tools and methodologies in addressing TD in OOD could be tested in CAD to favour greater editability [7] and or interoperable. Given the large number of variable ranging from context of use, to user behaviour, CAD refactoring, at its early stages, is likely to become a tool to aid the drafters in improving their designs making them more reusable in the future. In future work the author hope to propose CAD refactoring methodologies in more detail and to study their effects on the editability of CAD models against other alternative constructions. The value of reusable CAD models goes beyond the resources saved in recreate another model, but as model are tighly coupled with product data that are leveraged throughout the product life cycle.

References:

[1] Bertolotto, M. 'Geometric Modelling and Spatial Reasoning'. In Artificial Vision: Image Description, Recognition, and Communication, 107-34. Elsevier, 1997. https://doi.org/10.1016/b978-012444816-2/50011-3

[2] Cunningham, Ward. 'The WyCash Portfolio Management System'. In Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum), 29-30. OOPSLA '92. New York, NY, USA: Association for Computing Machinery, 1992. https://doi.org/10.1145/157709.157715

[3] Fowler, Martin. Refactoring: Improving the Design of Existing Code. Addison-Wesley, 2018.

[4] Hoffmann, Christoph M., and Robert Joan-Arinyo. 'Parametric Modeling'. In Handbook of Computer Aided Geometric Design, 519-41. Elsevier, 2002. https://doi.org/10.1016/b978-044451104-1/50022-8

[5] Kasik, David J., William Buxton, and David R. Ferguson. 'Ten CAD Challenges'. IEEE Computer Graphics and Applications 25, no. 2 (1 March 2005): 81-92. https://doi.org/10.1109/MCG.2005.48

[6] Rosso, Peter, James Gopsil, Stuart Burgess, and Ben Hicks. 'Does CAD Smell Like Code? A Mapping Between Violation of Computer Science Design Principles and CAD [Manuscript Accepted for Publication]'. In 17th INTERNATIONAL DESIGN CONFERENCE. CAVTAT, CROATIA, 2022.

[7] Rosso, Peter, James Gopsill, Stuart Burgess, and Ben Hicks. 'Investigating and Characterising Variability in CAD Modelling and Its Potential Impact on Editability: An Exploratory Study'. Computer-Aided Design and Applications 18, no. 6 (26 February 2021): 1306-26. https://doi.org/10.14733/cadaps.2021.1306-1326

[8] Suryanarayana, Girish, Ganesh Samarthyam, and Tushar Sharma. Refactoring for Software Design Smells:Managing Technical Debt, 2014.

[9] Ulrich, Karl T., and Scott Pearson. 'Assessing the Importance of Design Through Product Archaeology'. Management Science 44, no. 3 (1998). https://doi.org/10.1287/mnsc.44.3.352

[10] Ulrich, Karl T., and Scott Pearson. 'Assessing the Importance of Design Through Product Archaeology'. Management Science 44, no. 3 (1 March 1998): 352-69. https://doi.org/10.1287/mnsc.44.3.352

[11] Voelcker, Herbert, and Aristides Requicha. 'Constructive Solid Geometry', 1977.

[12] You, Chun Fong, and Yi Lung Tsai. '3D Solid Model Retrieval for Engineering Reuse Based on Local Feature Correspondence'. International Journal of Advanced Manufacturing Technology 46, no. 5-8 (2010): 649-61. https://doi.org/10.1007/s00170-009-2113-9