INTERNATIONAL DESIGN CONFERENCE – DESIGN 2022
https://doi.org/10.1017/pds.2022.176

Does CAD Smell Like Code? A Mapping between Violation of Object Oriented Programming Design Principles and Computer Aided Design Modelling

P. Rosso [1,✉], J. Gopsill [1,2], S. C. Burgess [1] and B. Hicks [1]

[1] University of Bristol, United Kingdom, [2] Centre for Modelling & Simulation, United Kingdom

✉ peter.rosso@bristol.ac.ukik

Abstract

In objected-oriented design, "smells" are symptoms of code violating design principles. When a deadline is looming, decisions can affect the long-term quality of a code or CAD. Given this and the similarities between object-oriented code and CAD models, this paper introduces a set of CAD smells. These smells are derived from a top-down review of potential CAD smells mapped against the reported code smells that violate abstraction, modularity, encapsulation, and hierarchy principles. This list was further reviewed considering CAD systems and specific examples (some illustrated in the paper).

Keywords: computer-aided design (CAD), design practice, design management, model-based engineering (MBE), object-oriented design

1. Introduction

Digital engineering and design are dependent on code, models and a variety of artefacts including Computer-Aided Design (CAD), simulations, analysis for Computer-Aided Manufacturing (CAM) and other relevant model-based initiatives. The importance of CAD/CAM tools has increased over time enabling more significant levels of collaboration between users, integrated workflows, and greater reusability of existing models across multiple lifecycles of a product. Capture and storage have been paramount, with numerous tools and methods created for formal and informal design information. These include CAD repositories, PDM and PLM systems, and rational capture. As the quantity of data that individuals and industries must deal with is increasing, so do the challenges of model management and version control. Kasik et al. (2005) listed ten of the challenges that CAD will face in this century, and highlights reusability and management of CAD files and corresponding product data as one of the most important.

A common example involves the access to, and editing of, existing files, which may not always be possible. While access to a file might still allow designers to create a form of design history specifying the connection between historical models and newer ones, it does not offer the same robustness and efficiency as if the model could be reused. As model-based enterprises are becoming more common, CAD has evolved to describe physical objects in a digital space and associating these models with relevant design data. Thus, further drawbacks in redesigning models from scratch, range from mistakes (Jackson and Buxton, 2007) to time loss (Jackson and Prawel, 2013). CAD models are part of a growing ecosystem that includes a broader set of product data. Product Data refers to all data involved in the design and manufacturing life cycles such as CAD model data, process planning data, geometric dimensioning and tolerance data, PDM data and CAE/CAM data (Yang et al., 2006).

CAD models' design intent and the associated ability to reuse a CAD model have been important subfields in CAD research (Karsenty, 1996; Regli et al., 2000). Design intent aims to capture encapsulates sufficient knowledge of how the designer generated the model to modify it by the original construction procedure (Junhwan Kim et al., 2008). Design intent is commonly, but not always understood to describe a model's anticipated behaviour once it undergoes alteration (Otey et al., 2018), thus it's not only important to make a file editable, but to also reduce the complexity of interacting with a model, as most behaviours following a CAD file modification can be anticipated by the drafter. The value of being able to anticipate propagation of changes (at least within the extent of a single model) is important given the variability in design intent (Rosso et al., 2021, 2020). Researchers have explored the impact of quality of CAD models on reuse, given the central role of CAD models, the variability in design intent, the derived complexity, and the increasing need to reuse models and the associated data (Camba et al., 2020, 2014, 2018, 2017; Gerbino, 2003; Gunduz and Yetisir, 2018; Jackson and Buxton, 2007; Jackson and Prawel, 2013; Otey et al., 2018; Yu et al., 2012). While considerations of such issues is relatively new to the CAD field, programming research introduced the concept of "smells" as a way to address similar problems (Fowler, 2018). In programming the concept of "smells" has been developed for Object Oriented (OO) languages which, due to the common standards (IGES & STEP) and feature-based constructs used in today's CAD systems can be considered analogous where CAD models can be considered as objects.
It follows that the contribution of this paper is to explore the concept of "smells" within the context of CAD models. The paper begins with a review of relevant literature in programming and engineering design. It follows with a description of the methodology used in mapping design principles and smells from programming to CAD. In the result section the authors show the process by which a classification of CAD smells is proposed with a selection of illustrative examples. The proposed set of CAD smells and examples are reflected upon in the Discussion to consider implications and limitation for CAD tools, designers and workflows, and future outlooks and research areas.

2. Background and Literature

2.1. CAD

The structure of CAD models has seen lots of changes in the last six decades allowing users to create and manage increasingly complex structures. Starting from wireframes, then using Boundary Representation (BRep), individuals and industry could store and visualise a large variety of models increasing the complexity of the objects. Constructive Solid Geometry (CSG) was developed to provide a representation scheme which aligns better with mental models used by engineers to describe manufacturing process of component (subtractive and additive processes) (Voelcker and Requicha, 1977). Current representation schemes are an evolution of BRep and CSG. An implicit representation such as BRep enables quick and reliable representation of models making properties and rendered visualisations readily available. The explicit representation evolved from CSG schemes provides affordances for easy and rapid design that encapsulates design intent and facilitates model reuse.
To promote reusability, it is crucial to have good quality CAD models and good quality Product Data. "Good product data quality means providing the right data to the right people at the right time (ISO, 2006)". This becomes difficult when data is retrieved with missing, disorganised, or unnecessary information. It is widely accepted that the quality of product data impacts the quality of design processes, which consequentially affects manufacturing quality. Low-quality product data often delays product development pacts the manufacturing process and final product quality (McKenney, 1998). One of the most critical is the CAD model which will impact downstream processes because of their central position start of the workflow, highlighting the dependency of design and manufacturing processes on such product data (González-Lluch et al., 2017).
Design reuse the design and manufacturing process from an existing product (or design) improves product development efficiency by reutilising the existing product data (e.g. CAD file, manufacturing setup, time cost estimation...)(You and Tsai, 2010). Jackson and Buxton (2007) and Jackson and Prawel (2013) provided an in depth review of benefits for model reuse and exchange.

Feature based parametric design is widely used in industrial settings enabling the creation of interconnected structure which lead to reusable models (Otey et al., 2018). The parent child relationship created in the representation schemes can either lead to highly adaptable and reusable models or to different problems caused by inconsistencies used to define them (Camba et al., 2020). It follows that a set of design principles that promote the creation of design models that are more reusable will lead in saving time, resources and improving quality of manufacture, improvement and support for products linked to "better" CAD models. Reusability, along with other quality attributes are some of the arguments for object-oriented design approach in programming. To set the scene with respect to Object-Oriented Programming (OOP) it is advantageous to review some of its basic concepts.

2.2. Object Oriented Design

In dealing with complexity, computer scientists developed different approaches to "break down" the problem space, allowing a more systematic approach to code design. Algorithmic decomposition approaches play better alongside procedural programming thinking. While an algorithmic decomposition highlights the ordering of events, Object-Oriented (OO) emphasises agents that cause action or are the subjects on which these operations act. These agents are called objects, and objects do things as a response to a trigger message.

> *"Object Oriented Design (OOD) is a method of design encompassing the process of object oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system under design." (Booch et al., 2008)*

The ability of breaking down a complex system into objects that follows design principles that the field has developed, provide users with the ability to use reusable code, often reducing times and costs. OO systems are also more resilient to change and thus better able to evolve over time because their design is based on stable intermediate forms, and uncoupled objects that facilitate distribution of labour. OOD's large adoption was accompanied by substantial research in developing tool, methods, and practices to allow companies to expand in size and reliably use their code over time.

2.2.1. Smells in Programming

Beck et al. (1999) coined the term smell to indicate characteristics of code likely to negatively impact quality of code. Despite not raising runtime errors (they are not bugs), smells are likely to lead deeper problems affecting code's quality attributes such as understandability, changeability, extensibility, reusability, testability, and reliability (e.g., low understandability or low testability could make code harder to use/fix code leading to negative impacts to the code's stakeholders).
Awareness of smells is important, because they are "symptoms" to a deeper design problem which further increases the cost of change of a project (Highsmith, 2009). (Suryanarayana et al., 2014) highlights that awareness of these "symptoms" can improve reusability of code along other quality attributes. Consequentially they offer an exhaustive and comprehensive classification of Design smells. Suryanarayana (2014)'s classification utilises a name scheme based on the four "major elements" of the object model discussed in (Booch et al., 2008): Principles of Hierarchy, Abstraction, Modularity and Encapsulation (PHAME). This classification provides a frame of reference for a comprehensive documentation of smells reviewing existing classifications.
This paper focus more specifically on design smells; a subset of smells caused by violation of design principles (Suryanarayana et al., 2014). The term smell will be used to refer to design smell for the rest of the paper.

2.2.2. Principles underpinning the Object Model

There are many OO principles that are used to optimise code quality. The most famous set of principles is known by the acronym of SOLID (Single-responsibility principle, Open-closed principle, Liskov substitution principle, Interface segregation principles and Dependency inversion principle). The authors will use an alternative (Table 1). The choice of reviewing these principles is also motivated by the need to be consistent with terminology used in (Suryanarayana et al., 2014)'s classification.

https://doi.org/10.1017/pds.2022.176 Published online by Cambridge University Press

Table 1. Definitions of PHAME principles as in (Suryanarayana et al., 2014).

OOD Principle	Description
Abstraction	The principle of abstraction advocates the simplification of entities through reduction and generalization: reduction is by elimination of unnecessary details and generalization is by identification and specification of common and important characteristics
Encapsulation	The principle of encapsulation advocates separation of concerns and information hiding through techniques such as hiding implementation details of abstractions and hiding variations.
Modularisation	The principle of modularization advocates the creation of cohesive and loosely coupled abstractions through techniques such as localization and decomposition.
Hierarchy	The principle of hierarchy advocates the creation of a hierarchical organization of abstractions using techniques such as classification, generalization, substitutability, and ordering

3. CAD vs Object-Oriented Design

Design and programming set themselves the challenge to map problems in the solution space following their respective fields rules. Despite their constraints, these processes afford significant scope for creativity, particularly at the point of initial construction. Some of the approaches used in one field are very similar to the ones in another field. An example is function decomposition (Dimarogonas and Lewis, 2001). Object decomposition in programming and function decomposition in Design aim to break down the problem addressed in smaller items that can be dealt with independently. Later the paper revisits this as reason for overlap between programming and axiomatic design principles.

Warman (1990) discusses the use of objected oriented approaches design of a CAD system. His argument lies on background in design theory and model interactions, considering design elements or features as objects that can have properties, and methods that allow them to handle messages. Arguing for an interface allowing to insert entities, as objects. For this analysis this is important as it gives us a reason to explore the use of other OO principles in the context of CAD model designs.

In a BRep, the information necessary to fully describe a model are entities (surfaces, lines, and vertices) and their relationship (adjacency). CSG-like representations are more complex as they offer different environments for a drafter to interact with the digital design space easily. In an assembly, the entities are parts and construction elements; relationships are joints and constraints. In a part/component environment, the entities are features, construction elements and sketches. When a command is instantiated, it requires inputs determining relative positions and parameters, creating relations in this environment. These environments provide patterning and mirroring to deal with repetitions in the design. Other relationships are determined by the order of the drafter's action (timeline/construction tree) associated with design intent. In a sketch environment, the entities to consider are vertices, lines, dimensions; the relationship between entities can be adjacency, projections, reflections, constraints, etc. Depending on the tool, all environments might be able to access the BRep of each feature using them as parameters. Explicit modelling therefore offers the possibility to have highly reusable models.

The overlap between OOD and CAD is the main reason to attempt to map OOD principles and smells in the attempt to develop a methodology to "bad design" in CAD. Changing context or design environment the drafter can change the level of **abstraction** they are designing. Whether breaking down a system in subsystems or a part into features to represent it, both for present and future designers' sake, there is value in having simplified entities that can be easily interacted with. Information axiom in AD argues for a similar idea, aiming for the minimum and essential information in a design.

Modularisation allows designers to break down complex problems into smaller systems easing the design task. Any CAD representation scheme had to break down a physical object into smaller entities that the GMS could organise. Implicit representations use Surfaces, Edges and Vertices; explicit representations use features. Explicit representations have more loose coupling than CSG, which is why they allow greater editability of CAD models. The independence axiom in AD aligns with modularisation.

https://doi.org/10.1017/pds.2022.176 Published online by Cambridge University Press

Encapsulation does not have the same importance that it has in programming. Depending on GMS, there are different levels of information hiding depending on the level of abstraction (environment) you are working in. Ideally, a model should use entities that are available in the level of abstraction that the designer is working in. Parametric design has gone beyond this, allowing designers to access data to the parent level (e.g. projections). At times there are models which access information from any environment in the same file without regard for the loss of modularisation. Encapsulation should be considered, but more loosely than other principles. (Arguably parametric design breaks encapsulation).

Hierarchical organisation of entities allows GMS to minimise repetitions and propagate changes to all the connected entities when necessary. In cad classification, generalisation and ordering are maintained in GMS, while substitutability does not apply as entities making a model cannot be used for that model. In programming hierarchy is either defined by *is-a* or *has-a* relationships. *Is-a* is a subsumption relationship, requiring satisfaction of Liskov substitution principle - i.e., child entities can be substituted for parent entities and satisfy the same responsibility. This cannot be warranted in CAD as a sketch child to a solid feature cannot be changed for the child feature in a construction tree. The *has-a* relationship is a composition relationship which is necessary to define both BRep and CSG-like representations.

4. Methodology

CAD research and practices focus on different directions dependent on their subfields and the motivations of the researchers, leading to various approaches geared towards optimal design. This variability in objectives, tools, and practices is compounded by a lack of literature that explicitly focusses on design principles for CAD models. Consequently, attempting to compare, contrast and correlate design smells in code and CAD is not straightforward due to the absence of a top-down taxonomy of, what could be thought of as CAD smells (quality issues). Given this lack of extant classification, one of two approaches can be adopted: 1) a bottom-up elicitation and theming of CAD smells by practitioners or 2) a critical evaluation of the applicability and transferability of OOD smells to CAD by experts. Both approaches have merit, but despite the long application of CAD technologies it is harder to follow a bottom-up approach as it was done in programming. CAD lacks large open-source projects and homogeneity of languages and tools that programming had while undergoing the same process. Hence the second approach is adopted in this paper.

Consequentially, given the lack of extant material, the start point is to review OO principles in the context of CAD and follow a deductive approach to map programming smells to, what the authors refer to as CAD smells where appropriate. Mapping two fields, this research followed (Nickerson et al., 2013)'s taxonomy development method to guide the process. As recommended by (Nickerson et al., 2013; Omair and Alturki, 2020), a conceptual-to-empirical approach led to a survey of existing taxonomies and classifications related to this research topic. Starting from a conceptual-to-empirical (top-down) approach leaning on an existing classification of programming smells. This was examined, leading to some generalisation and subtraction of fields. This first stage was carried out considering whether and why existing smells apply to CAD, if and how they could be prevented by a Geometrical Modelling System (GMS), and what environment could be considered. A second stage reviewed the remaining list of CAD smells, providing a description applicable to CAD an example, environment affected, impacted quality attributes, and practical considerations (i.e., when the occurring of this smell might be inevitable or desirable by the designer). Once the list was comprehensive, it was presented to a group of researchers which provided useful question and comments leading to a further review. To make the distinction between the proposed attempt in classifying smells in the field of CAD, the authors will call violations of CAD modelling principles, CAD smells.

The process was undertaken over three months from August 2021. (Suryanarayana et al., 2014) reviews over 90 design smells (including smells from other reviews which are referred as alias for the PHAME named smells). These smells are generalised and reduced to 25 smells named as violation of PHAME. Considering each of these smells, their associated alias, and CAD related parameters the list was further reduced to 19 CAD smells. Smells are summarised in brief herein and given in full in “CAD Design Smells Classification” available at data.bris.ac.uk. Properties of encapsulation and hierarchy made in

Section 3 lead to reduction of the smell which apply to CAD. The parameters considered in the reduction and classification of the proposed CAD smells were: entities causing the smell and the associated editing environment (sketch, part, assembly), role of GMS in preventing the smell, or introducing viscosity that would lead to the smell, further justifications against CAD practices and examples of when the smell would be applicable. Furthermore, akin to (Suryanarayana et al., 2014), this research considered practical considerations of when a smell could be unavoidable or wanted.

5. Results: Mapping form Programming Smells to CAD Smells

Table 2 presents a mapping of reviewed programming smells to CAD smells indicating in which environment these could be applicable. Some GMS have been constructed providing affordances for good design behaviour, alternatively they also preclude drafters from creating some instances that would violate PHAME principles, leading to smells. More in debt information about smells is given in full in "CAD Design Smells Classification" available at data.bris.ac.uk.

Table 2. Programming Smells, GMS intervention, environment affected and applicability to CAD

#	Programming Smells	GMS Preventing Smell	Sketch	Part	Assembly	CAD Smell?
1	Missing Abstraction	~	y			Y
2	Imperative Abstraction	*not applicable*				
3	Incomplete Abstraction	~	y	y		Y
4	Multifaceted Abstraction		y			Y
5	Unnecessary Abstraction		y	y	y	Y
6	Unutilised Abstraction		y	y	y	Y
7	Duplicate Abstraction		y	y	y	Y
8	Deficient Encapsulation	y				Y
9	Leaky Encapsulation	~				Y
10	Missing Encapsulation	*not applicable*				
11	Unexploited Encapsulation	y				N
12	Cyclically-Dependent Modularisation		y	y	y	Y
13	Hub-like Modularisation		y	y	y	Y
14	Broken Modularisation		y	y		Y
15	Insufficient Modularisation		y	y	y	Y
16	Missing Hierarchy	~		y	y	Y
17	Unnecessary Hierarchy		y	y		Y
18	Unfactored Hierarchy	*not applicable*				
19	Wide Hierarchy			y	y	Y
20	Speculative Hierarchy	*not applicable*				
21	Deep Hierarchy			x	x	Y
22	Rebellious Hierarchy	*not applicable*				
23	Broken Hierarchy	x		x	x	Y
24	Multipath Hierarchy	~	x	x	x	Y
25	Cyclic Hierarchy	~		x	x	Y

Table 2 considers the possible presence of a smell in different environments (Sketch, Part, Assembly) and their applicability to CAD using *y/Y* for yes and *n/N* for no; ~ is used for sometimes.

6. Illustrative Examples

This section illustrates a selection of the smells that are associated with editability of CAD models.

6.1. Duplicate Abstraction (Table 2, #7)

Description: This smell arises when two or more entities have identical type and common properties beside position in the 3D space. The presence of visual patterns that could have been expressed in the GMS is a possible sign of the presence of this smell.

Rationale: Duplication is common in CAD models despite GMS providing tools to avoid it (e.g. patterning, mirroring, and other commands allowing multiple instances). On the other hand, excessive

https://doi.org/10.1017/pds.2022.176 Published online by Cambridge University Press

use of the tools provided by the GMS would lead to another smell (wide hierarchy). Thus, it is essential to use these tools within the same abstraction and be mindful of the modularisation principle beside the abstraction principle, which is primarily violated in this case. Keeping in mind the design intent for an object, it is possible to avoid this smell. The appropriate use of duplication-prevention tools will imply a relationship between these entities as part of the design intent.

Examples: In sketches, regardless of their size and complexity, there are often entities that are repeated and should be associated using patterning or mirroring. **See** *Figure 1a,1b***.** Use of these implicit relationships (not shown on construction tree) can make editing of a model much less error prone and time consuming.

Impacted quality attributes: Excessive duplication decreases the understandability of a design, impacts its changeability, extensibility and editability. Within the design context, this impacts the reusability of the design.

Practical considerations: None

6.2. Broken Modularisation (Table 2, #14)

Description: This smell arises when data (dimensions and entities) that the drafter should have localised into a single entity are separated and spread across multiple abstractions.

Rationale: Drafters break an entity into multiple entities trying to avoid excessive complexity in one environment; this increases the number of related features and relations, increasing the complexity of the design and leading to a fragile design.

Potential causes: The drafter might not be aware of the tools offered by a GMS, or they might not have planned for certain entities to be associated in the long run.

Examples: In attempting to avoid a sketch with multiple complex shapes the drafter might attempt to break this down into multiple sketches which are all represented on the same surface and then they are used to build multiple features summing up to the same solid. See *Figure 1a,1c*. When the drafter will have to edit this solid, they will have to access different sketches and hope that projection and other parametric relations maintain the consistency across features.

Impacted quality attributes: Editability, and reusability of a model are impacted as properties that are related to one entity might be related to other entities upstream residing in different editing environments. Consequentially drafters are more prone to make mistakes or finding the correct property can be more time consuming.

Practical considerations: There might be industrial guidelines that invite drafters to separate specific sketches for future extension of work, despite these entities might start from the same surface and be associated with the same feature or the same solid.

6.3. Wide Hierarchy (Table 2, #19)

Description: This smell arises when a relationship hierarchy is "too wide: indicating that intermediate elements may be missing.

Rationale: There are circumstances in which CAD models present too many entities that are dependent upon a single entity. This leads to a brittle design as one single change applied to the highly connected entity can lead to unwanted/unpredicted/unaccounted changes in the related features. Given the high likelihood of a CAD model also having implicit relationship that do not show on the construction tree it is possible to have a child entity inheriting links to multiple parents via projections, or parametric design practices. As indicated in Camba et al. (2020), excessive use of these relationship can make the model hard to edit, change or extend.

Examples: Single entities that are connected to one another such as a different negative extrusion on a body created with a positive extrusion are likely to be constrained so that they are related to the size of the object and/or might have relative distance to each other. See *Figure 1a,1c*. When modifications are made to the initial body, children entity might come into conflicts or move in unexpected positions.

Impacted quality attributes: Editability, changeability and extensibility are the main quality attributes impacted. Entities connected to many other features lead to brittle structures as the modification or removal of a single entity will propagate to many dependent ones. Consequently, a drafter will have to be mindful of changes and might have to resolve this smell before further design activities.

https://doi.org/10.1017/pds.2022.176 Published online by Cambridge University Press

Practical considerations: As the drafter approaches the design space, the first entity is likely to start a wide hierarchy. This often happens as this feature needs to be the first in the design workspace. Origin axis and surfaces often have many related features as they are the only elements in the design space when a drafter starts, and they are required to provide a frame of reference.

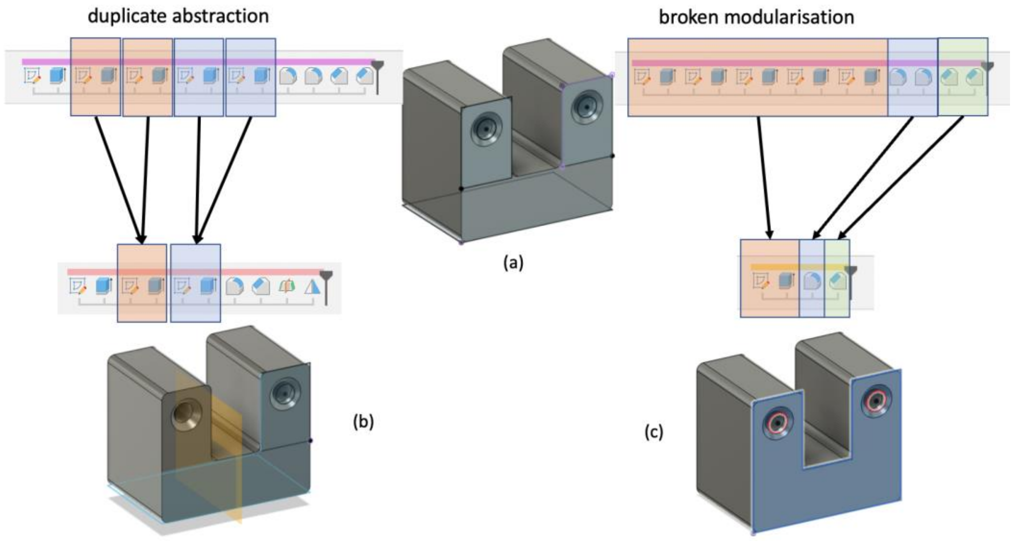

Figure 1. Illustrated Examples of CAD smells in Autodesk Fusion360 - Part (a) exabits both duplication abstraction (#7) and broken modularisation (#14) It's harder to get a wide hierarchy (#19) as the GMS associates sketches with the body rather than feature. Parts (b) and (c) show alternative representations avoiding smells

7. Discussion

In Table 2, some of the smells were considered not applicable. Despite drawing a parallel between CAD and OO Design, CAD has considerably less freedom than code. Consequentially some of the smells considered for programming do not apply to CAD because of the limited "degree of freedom" in which CAD can be utilised. For instance, no "Imperative Abstraction" in CAD has entities in models that will always represent an object to exist in a design environment. "Unnecessary, Rebellious and Speculative Hierarchy" are associated with *is-a relationship, which in Section 3, was discussed and ruled out as a relationship that drafters can find in CAD (within a design context)*. The differences between GMS are non-exhaustive but do associate with a similar problem in OOD, which is caused by differences in editing tools and languages. Different GMS have proprietary data structures. Thus, considering smells, a drafter should consider both environment and representation schema together, as they might limit or offer affordances leading to specific bad design approaches. This is not a critic of the tools themselves but of the misuse of the tools by the drafters.

This list of CAD smells is non-exhaustive and should be considered as guidelines. As the authors tried to show in the practical considerations section, there are examples in which what classify as a CAD smell is a good design choice required by the circumstances in which the drafter is operating. CAD smells like programming smells are non-exclusives but addressing the main smell might resolve the others. Additionally, CAD smells are named after the PHAME principles they primarily violated; what this implies is that a smell might violate multiple principles. Programming smells classifications depend on reviewing different professionals' experiences after years of awareness of such problems. Design does not yet have enough insight to offer a fully bottom-up approach in characterising and classifying CAD smells. The authors hope that in future, this work will be extended and challenged considering different bottom-up approaches and leading to larger data sets informing researchers on good and bad design approaches in CAD. The authors are aware that the naming schema might not be as intuitive as it could be for a computer scientist but developing a new naming schema goes beyond

https://doi.org/10.1017/pds.2022.176 Published online by Cambridge University Press

the purpose of this paper. Furthermore, there is hope that these names might reflect why they have been chosen and on an individual CAD practice.

8. Conclusions

This paper proposes a mapping between programming smells and CAD smells, considering the properties and behaviour of CAD models and their similarity to OO models. A list of possible CAD smells matching the naming schema offered by Suryanarayana (2014) has been developed through a review and comparison of established smells from programming. This process resulted in 90 programming smells that were filtered down to 25 smells and later pruned to 19 smells. A few CAD smells are demonstrated through illustrative examples. In accord with the field of programming, establishing the smells provides a basis for training (good practice), automatic detection, recommender/guidance systems and, where appropriate, the consideration of technical debt. Using this taxonomy of smells, the authors are currently exploring automated techniques for refactoring CAD models to eliminate smells. The proposed classification of CAD smells can be used to increase awareness of the impact of violation of design principles and bad practices in CAD models. While this characterisation might not be important for single use files, small to large projects are likely to have to reuse models. The importance of model reuse was raised in the paper.

Future work hopes to explore the importance of CAD smells as symptoms of growing technical debt in model-based enterprises. The characterisation of symptoms calls for identification methods (CAD smells detection) and a search for a cure. The increasing awareness of this problem and lack of possible solutions, combined with the relatively recent growth of Model-Based Engineering will act to focus industry's attention towards this problem, eventually pushing for a solution. Additionally, the lack of a framework to analyse CAD models systematically presents a challenge in taking a bottom-up approach to elicitation of CAD smells. Future work could propose representations that could aid such process. Ultimately if it is possible to identify a symptom, which has a considerable impact on the functioning of a system, it is essential to find a remedy to address the problem. While education might address this at its roots, there are plenty of CAD models which could become more valuable assets by removing CAD smells which are obstacles to reusability and taking full advantage of the model-based enterprises.

Acknowledgments

The work reported in this paper has been undertaken as part of a PhD project at the **University of Bristol**, which is founded by th**e EPSRC**, grant reference (**EP/N509619/1).** The authors would also like to thank **Chris Cox, Daniela Schamroth Rossade, George Barnaby, Harry Felton, Lorenzo Giunta, Lee Kent, Ric Real, Robert Ballantyne, and Shuo Su, for their comments during the Design and Manufacturing Futures Lab's show and tell.**

References

Beck, K., Fowler, M., Beck, G., 1999. Bad smells in code, in: Refactoring: Improving the Design of Existing Code. pp. 75–88.

Booch, G., Maksimchuk, R.A., Engle, M.W., Young, B.J., Connallen, J., Houston, K.A., 2008. Object-oriented analysis and design with applications, third edition. https://doi.org/10.1145/1402521.1413138

Camba, J., Contero, M., Company, P., Hartman, N., 2020. The Cost of Change in Parametric Modeling: A Roadmap. Comput.-Aided Des. Appl. 18, 634–643. https://doi.org/10.14733/cadaps.2021.634-643

Camba, J., Contero, M., Johnson, M., Company, P., 2014. Extended 3D annotations as a new mechanism to explicitly communicate geometric design intent and increase CAD model reusability. Comput.-Aided Des. 57, 61–73. https://doi.org/10.1016/j.cad.2014.07.001

Camba, J.D., Contero, M., Company, P., 2017. CAD Reusability and the Role of Modeling Information in the MBE Context 54.

Camba, J.D., Contero, M., Company, P., Pérez-López, D., Otey, J., 2018. Identifying High-Value CAD Models: An Exploratory Study on Dimensional Variability As Complexity Indicator. Presented at the ASME 2018 13th International Manufacturing Science and Engineering Conference, American Society of Mechanical Engineers Digital Collection. https://doi.org/10.1115/MSEC2018-6391

Dimarogonas, AD ,. Author, Lewis, G.,. Reviewer, 2001. Machine Design: A CAD Approach. Appl. Mech. Rev. 54, B65–B68. https://doi.org/10.1115/1.1383679

Fowler, M., 2018. Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional.

Gerbino, S., 2003. Tools for the Interoperability among CAD Systems, in: Undefined. p. 13.

González-Lluch, C., Company, P., Contero, M., Camba, J.D., Plumed, R., 2017. A survey on 3D CAD model quality assurance and testing tools. Comput.-Aided Des. 83, 64–79. https://doi.org/10.1016/j.cad.2016.10.003

Gunduz, M., Yetisir, T., 2018. A design reuse technology to increase productivity through automated corporate memory system. Neural Comput. Appl. 29, 609–617. https://doi.org/10.1007/s00521-016-2586-z

Highsmith, J., 2009. Zen and the art of software quality. Presented at the Agile 2009 Conference.

ISO, 2006. ISO/PAS 26183:2006 SASIG Product data quality guidelines for the global automotive industry (No. ISO/PAS 26183:2006 SASIG).

Jackson, C., Buxton, M., 2007. The Design Reuse Benchmark Report: Seizing the Opportunity to Shorten Product Development. Aberdeen Group.

Jackson, C., Prawel, D., 2013. The 2013 state of 3D collaboration and interoperability report, Lifestyle Insights and Longview Advisors.

Junhwan Kim, Michael J.Pratt, Raj G.Iyer, Ram D.Sriram, 2008. Standardized data exchange of CAD models with design intent. Comput.-Aided Des. 40, 760–777. https://doi.org/10.1016/J.CAD.2007.06.014

Karsenty, L., 1996. An empirical evaluation of design rationale documents, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. Association for Computing Machinery, New York, NY, USA, pp. 150–156.

Kasik, D.J., Buxton, W., Ferguson, D.R., 2005. Ten CAD Challenges. IEEE Comput. Graph. Appl. 25, 81–92. https://doi.org/10.1109/MCG.2005.48

McKenney, D., 1998. Model quality: the key to CAD/CAM/CAE interoperability. Int. TechneGroup Inc. Milford OH.

Nickerson, R.C., Varshney, U., Muntermann, J., 2013. A method for taxonomy development and its application in information systems. Eur. J. Inf. Syst. 22, 336–359. https://doi.org/10.1057/ejis.2012.26

Omair, B., Alturki, A., 2020. An Improved Method for Taxonomy Development in Information Systems. Int. J. Adv. Comput. Sci. Appl. 11. https://doi.org/10.14569/IJACSA.2020.0110470

Otey, J., Company, P., Contero, M., Camba, J.D., 2018. Revisiting the design intent concept in the context of mechanical CAD education. Comput.-Aided Des. Appl. 15, 47–60. https://doi.org/10.1080/16864360.2017.1353733

Regli, W.C., Hu, X., Atwood, M., Sun, W., 2000. A survey of design rationale systems: Approaches, representation, capture and retrieval. Eng. Comput. 16, 209–235. https://doi.org/10.1007/PL00013715

Rosso, P., Gopsil, J., Hicks, B., Burgess, S., 2020. Investigating and characterising variability in CAD modelling: An overview, in: CAD Conference and Exhibition. CAD Solutions, LLC, pp. 226–230. https://doi.org/10.14733/cadconfp.2020.226-230

Rosso, P., Gopsill, J., Burgess, S., Hicks, B., 2021. Investigating and Characterising Variability in CAD Modelling and its Potential Impact on Editability: An Exploratory Study. Comput.-Aided Des. Appl. 18, 1306–1326. https://doi.org/10.14733/cadaps.2021.1306-1326

Suryanarayana, G., Samarthyam, G., Sharma, T., 2014. Refactoring for software design smells : managing technical debt.

Voelcker, H., Requicha, A., 1977. Constructive Solid Geometry.

Warman, E.A., 1990. Object Oriented Programming and CAD. J. Eng. Des. 1, 37–46. https://doi.org/10.1080/09544829008901641

Yang, J., Han, S., Kang, H., Kim, J., 2006. Product data quality assurance for e-manufacturing in the automotive industry. Int. J. Comput. Integr. Manuf. 19, 136–147. https://doi.org/10.1080/09511920500171261

You, C.F., Tsai, Y.L., 2010. 3D solid model retrieval for engineering reuse based on local feature correspondence. Int. J. Adv. Manuf. Technol. 46, 649–661. https://doi.org/10.1007/s00170-009-2113-9

Yu, J., Cha, J., Lu, Y., 2012. Design synthesis approach based on process decomposition to design reuse. J. Eng. Des. 23, 526–543. https://doi.org/10.1080/09544828.2011.629316