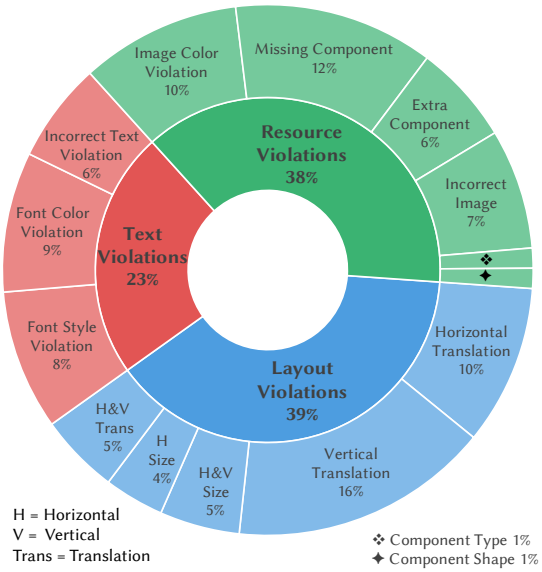

Figure 2: Distribution of Different Types of Industrial *DVs*

3.1 Study Setting & Methodology

The *goal* of this study is to derive a taxonomy of the different types of *DVs* and examine the distribution of these types induced during the mobile app development process. The *context* of this study is comprised of a set of 71 representative mobile app mock-up and implementation screen pairs from more than 12 different internal apps, annotated by design teams from our industrial partner to highlight specific instances of *resolved DVs*. This set of screen pairs was specifically selected by the industrial design team to be representative both in terms of diversity and distribution of violations that typically occur during the development process.

In order to develop a taxonomy and distribution of the violations present in this dataset, we implement an open coding methodology consistent with constructivist grounded theory [17]. Following the advice of recent work within the SE community [45], we stipulate our specific implementation of this type of grounded theory while discussing our deviations from the methods in the literature. We derived our implementation from the material discussed in [17] involving the following steps: (i) establishing a research problem and questions, (ii) data-collection and initial coding, and (iii) focused coding. We excluded other steps described in [17], such as memoing because we were building a taxonomy of labels, and seeking new specific data due to our NDA limiting the data that could be shared. The study addressed the following research question: *What are the different types and distributions of GUI design violations that occur during industrial mobile app development processes?*

During the initial coding process, three of the authors were sent the full set of 71 screen pairs and were asked to code four pieces of information for each example: (i) a general category for the violation, (ii) a specific description of the violation, (iii) the severity of the violation (if applicable), and (iv) the Android *GC* types affected (*e.g.,* button). Finally, we performed a second round of coding that combined the concepts of focused and axial coding as described in [17]. During this round two of the authors merged the responses from all three types of coding information where at least two of the three coders agreed. During this phase similar coding labels were merged (*e.g.,* "layout violation" vs. "spatial violation"), conflicts were resolved, two screen pairs were discarded due to ambiguity, and cohesive categories and subcategories were formed. The author agreement for each of the four types of tags is as follows: (i) general violation category (100%), (ii) specific violation description (96%), (iii) violation severity (100%), and (iv) affected *GC* types (84.5%).

3.2 Grounded Theory Study Results

Our study revealed three major categories of design violations, each with several specific subtypes. We forgo detailed descriptions and examples of violations due to space limitations, but provide examples in our online appendix [35]. The derived categories and subcategories of *DVs*, and their distributions, are illustrated in Fig. 2. Overall 82 *DVs* were identified across the 71 unique screen pairs considered in our study. The most prevalent category of *DVs* in our taxonomy are *Layout Violations* ($\approx$ 40%), which concern either a translation of a component in the x or y direction or a change in the component size, with translations being more common. The second most prevalent category ($\approx$ 36%) consists of *Resource Violations*, which concern missing components, extra components, color differences, and image differences. Finally, about one-quarter ($\approx$ 24%) of these violations are *Text Violations*, which concern differences in components that display text. We observed that violations typically only surfaced for "leaf-level" components in the GUI hierarchy. That is, violations typically only affected atomic components & not containers or backgrounds. Only 5/82 of examined violations ($\approx$ 6%) affected backgrounds or containers. Even in these few cases, the violations also affected "leaf-level" components.

The different types of violations correspond to different inequalities between the attribute tuples of corresponding GUI-components defined in Sec. 2. This taxonomy shows that designers are charged with identifying several different types of design violations, a daunting task, particularly for hundreds of screens across several apps.

4 THE GVT APPROACH

4.1 Approach Overview

The workflow of GVT (Fig. 3) proceeds in three stages: First in the *GUI-Collection Stage*, GUI-related information from both mock-ups and running apps is collected; Next, in the *GUI-Comprehension Stage* leaf-level *GCs* are parsed from the trees and a KNN-based algorithm is used to match corresponding *GCs* using spatial information; Finally, in the *Design Violation Detection Stage DVs* are detected using a combination of methods that leverage spatial *GC* information and computer vision techniques.

4.2 Stage 1: GUI Collection

4.2.1 Mock-Up GUI Collection. Software UI/UX design professionals typically use professional-grade image editing software (such as Photoshop[1] or Sketch[10]) to create their mock-ups. Designers employed by our industrial partner utilize the Sketch design software. Sketch is popular among mobile UI/UX designers due to its simple but powerful features, ease of use, and large library of extensions [11]. When using these tools designers often construct graphical representations of smartphone applications by placing *objects* representing *GCs* (which we refer to as *mock-up GCs*) on a *canvas* (representing a Screen S) that matches the typical display size of a target device. In order to capture information encoded in these mock-ups we decided to leverage an export format that