# A Survey of Patterns for Adapting Smartphone App UIs to Smart Watches

Zhilan Zhou
The University of North Carolina at Chapel Hill
Chapel Hill, United States

Jian Xu
Stony Brook University
Stony Brook, United States

Aruna Balasubramanian
Stony Brook University
Stony Brook, United States

Donald E. Porter
The University of North Carolina at Chapel Hill
Chapel Hill, United States

## ABSTRACT

Wearable devices, such as smart watches and fitness trackers are growing in popularity, creating a need for application developers to adapt or extend a UI, typically from a smartphone, onto these devices. Wearables generally have a smaller form factor than a phone; thus, porting an app to the watch necessarily involves reworking the UI. An open problem is identifying best practices for adapting UIs to wearable devices.

This paper contributes a study and data set of the state of practice in UI adaptation for wearables. We automatically extract UI designs from a set of 101 popular Android apps that have both a phone and watch version, and manually label how each UI element, as well as how screens in the app, are translated from the phone to the wearable. The paper identifies trends in adaptation strategies and presents design guidelines.

We expect that the UI adaptation strategies identified in this paper can have wide-ranging impacts for future research and identifying best practices in this space, such as grounding future user studies that evaluate which strategies improve user satisfaction or automatically adapting UIs.

## CCS CONCEPTS

• **Human-centered computing** → **Mobile devices**; **Ubiquitous and mobile computing design and evaluation methods**.

## KEYWORDS

wearable devices; smartphones; smartwatches; dataset; UI design patterns

**ACM Reference Format:**
Zhilan Zhou, Jian Xu, Aruna Balasubramanian, and Donald E. Porter. 2020. A Survey of Patterns for Adapting Smartphone App UIs to Smart Watches. In *22nd International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '20), October 5–8, 2020, Oldenburg, Germany.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3379503.3403564


*MobileHCI '20, October 5–8, 2020, Oldenburg, Germany*

ACM ISBN 978-1-4503-7516-0/20/10…$15.00
https://doi.org/10.1145/3379503.3403564

## 1 INTRODUCTION

Smart watches are rapidly growing in popularity. Smart wearables, including watches, increased almost 9% in 2018 to 125.3 million devices and it is predicted that the market will grow an average of 11% annually to reach 189.9 million devices in 2022 [26], which makes wearables be one of the tech industry's bright spots in next few years. Smart watches unlock new opportunities for software applications, both in communicating information to the user, or through additional sensors that obtain input from the user, such as monitoring heartbeat and temperature for personal health applications, or gesture tracking for a software tennis coaching application.

A common usage pattern for the first generation of watch apps is to serve as a companion app to a smartphone [29]. In the companion app model, an app runs primarily on the user's smartphone and the watch presents a subset of the phone's display. Since smartwatches have a smaller display than a phone, in to create a companion app, the app developer must subdivide or rearrange portions of the phone's user interface (or UI). The main motivation for this adaptation is to provide a user with an alternate interaction mechanism. For example, the *Spotify* music player app adapts a subset of the controls to the smartwatch, so a user can pause and play a song from their watch.

Currently, little is understood in terms of best practices for designing a companion app. When each app developer decides to add smartwatch support, the UI adaptation from the phone to the watch is done by hand, in an ad hoc manner. Developers lack any scientifically grounded guidance about which patterns are best for users, and further lack automation in this adaptation process. There are a number of ways to subdivide the same UI for a watch, and which strategy makes the most sense will likely vary depending on the nature of the app.

A prerequisite for evaluative studies of different adaptation techniques is to have a common *vocabulary* to describe different adaptation strategies. In other words, one cannot even ask the question of which UI adaptation strategy is best suited for application X, without a good menu of adaptation strategies to then compare. Moreover, there is some utility in understanding how common a given strategy is in practice, regardless of whether this strategy is best for the app or not.

The primary contribution of this paper is a dataset, similar in spirit to RICO [7] or Erica [8], and a study of 101 popular Android smartphone apps and how they are adapted to a smartwatch counterpart. This dataset draws specifically from free watch applications

that are a companion to a phone application, offer non-trivial functionality beyond a simple "watch face", and that does not require excessive effort to authenticate an end-user, such as banking applications. This dataset describes how UI elements on the phone correspond to the watch UI (or don't). The paper then identifies and describes common adaptation strategies, and draws insights from the trends in which apps use a given strategy.

The contributions of this paper are:

- A data set mapping smartwatch UIs onto smartphone UIs, drawn from 101 popular apps on the Wear OS Play Store.
- A categorization of the common phone and watch screen layouts into eight and five patterns, respectively.
- An analysis of the trends and outliers in the dataset. For instance, we observe that the overwhelming majority of *content* screens are mapped one-to-one onto the watch, with some subsetting of details, whereas *navigational* features are the most likely to introduce churn in the UI layout. Similarly, the importance, not the amount, of content is most likely to dictate the watch layout; for example, when a list will not fit on the watch display, this list will either reduce to summarizing the entries if the list itself is important, or drop to a fixed display if a few elements are the most important content.
- Additional design guidelines for developers extending a UI from a phone to a watch.

This study serves as a key building block for a comparative study of these strategies in future work. One immediate application of this work is in automating companion app generation. A recent framework, called UIWear [29], shows that one can reduce the effort to create a companion app by only requiring a developer to specify the UI adaptation design; the rest of the companion app is generated by the UIWear framework. The design patterns identified in this paper can likely be used to automatically choose the best adaptation design, to fully automate the companion app generation. The dataset in this paper is available at https://oscarlab.github.io/projects/ui-adaption/, which we hope will accelerate future work on wearable UI design.

## 2 PHONE UI LAYOUTS

This section describes common design patterns for phone UI screens. Note that we use the term **display** to refer to the graphical output device of a phone or watch, and use the term **screen** to refer to the contents of the display at one point in time. An app's UI can have multiple screens. A **UI component** is a general term, used to refer to any element that can appear on the user interface. Examples of UI components include, but are not limited to: an image, a paragraph of text, or a button containing a text label. A UI design pattern is defined by the combination and arrangement of components within the screen.

This section presents both design patterns described in the book *Mobile design pattern gallery* [22], which focuses on navigational patterns, and also presents content-based patterns we observe in practice.

### 2.1 Navigation UI Patterns

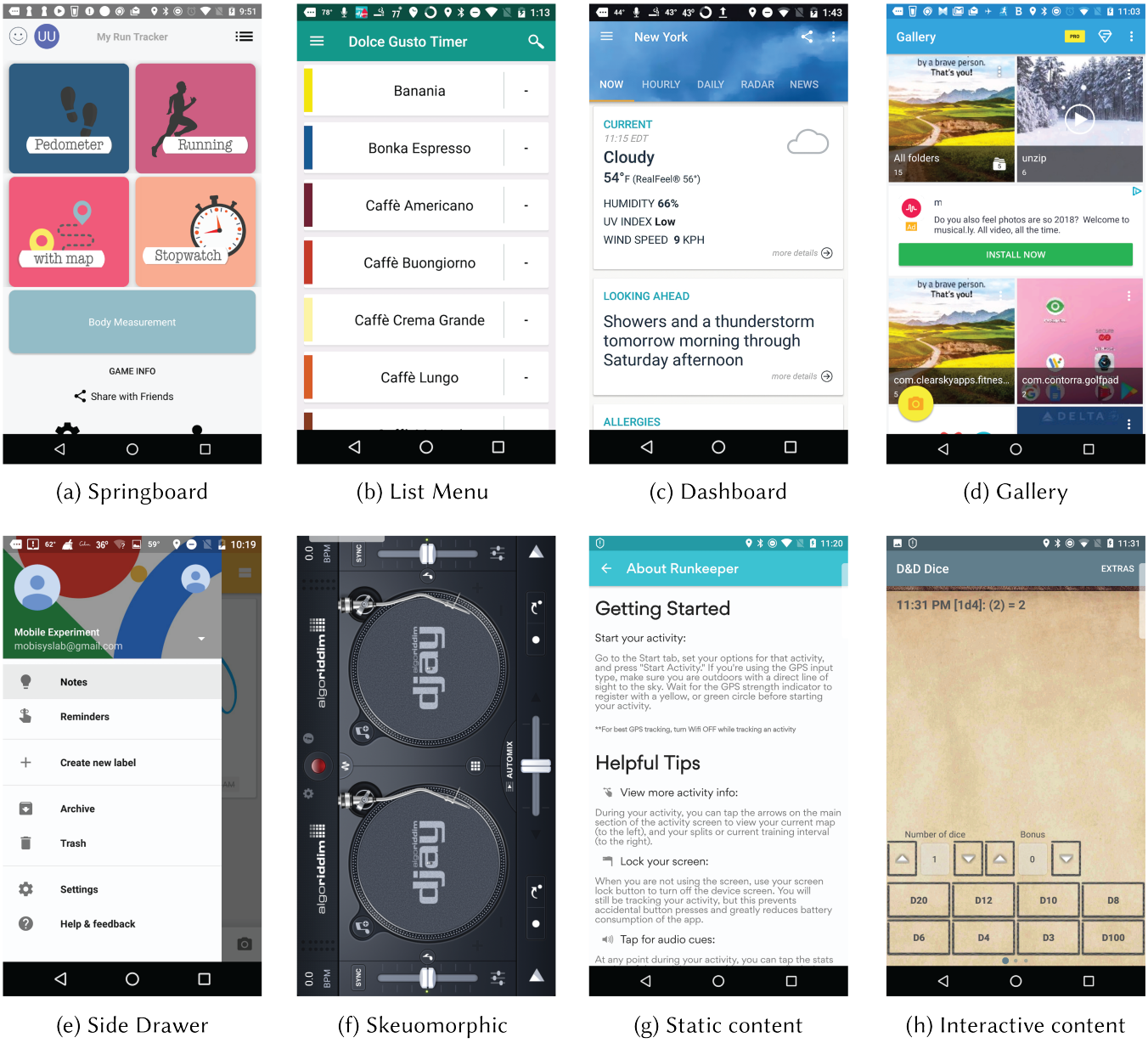
When all app content does not fit on a single screen, the app needs to organize navigation among multiple screens. These screens essentially form a graph, with the initial screen as the root of the graph. Each screen is a node and edges are interactions that transition to another screen, such as a swipe or a click. The vast majority of edges, if not all edges, are bidirectional. A screen that follows a **navigational UI pattern** primarily serves to organize navigation among other screens. An app may use multiple different navigational UI patterns within a hierarchy, such as clicking on a button in one screen leading to another screen showing a list.

The six UI patterns we adopted from *Mobile design pattern gallery* [22] cover almost all navigational UI organizations that we have observed in practice. There are additional patterns in the book that are no longer popular in phone apps, such as Cards and Pie Menus, or that are not suitable for mapping onto a watch, such as the Toggle Menu that is primarily used in browser apps. The book also mentions a Tab Menu as a separate pattern distinguished by the presence of a Bottom Navigations [13]; the Bottom navigation bar is now sufficiently ubiquitous that we no longer count this as a separate category. We will talk on the adaption strategies for several popular UI components including the bottom navigation bar in § 5.9.

(1) A **Springboard**, or Launchpad, pattern (Figure 1a) is often the first screen of the app, and is designed to help the user navigate to different locations within the app. A Springboard is typically a set of buttons, which can be either just icons or icons with text, and these buttons select sub-features of the app.
(2) A **List Menu** pattern (Figure 1b) displays all the navigation destinations to other parts of the app as a list. Each list item can be represented with icons and/or text. Unlike a Springboard, the appearance of all items in the List Menu is uniform.
(3) A **Dashboard** pattern (Figure 1c) displays detailed information at a glance in the form of a list. Unlike a List menu, each item on a Dashboard may have different styles and may contain information beyond icons and text. All the items on a Dashboard follow one another to form a long, vertical scrolling interface.
(4) A **Gallery** pattern (Figure 1d) displays dynamic content as a grid of images. Unlike a Springboard, a Gallery must contain images from the app's data, rather than an abstract icon.
(5) A **Side Drawer** pattern (Figure 1e) is displayed only on a part of the app screen, and is usually activated by selecting the Navigation drawer control button in a shortcut bar on the top of the app (on Android). The navigation destinations in the Side Drawer are app-wide, usually displayed as a list of icons and/or text labels.
(6) A **Skeuomorphic** pattern (Figure 1f) is a digital representation matching a real-world object or tool, with which the user can interact to navigate to other destinations in the app.

### 2.2 Content-based UI Patterns

In addition to navigational patterns discussed above, we observe that some app screens simply display static or dynamic information.




(a) Springboard (b) List Menu (c) Dashboard (d) Gallery

(e) Side Drawer (f) Skeuomorphic (g) Static content (h) Interactive content

**Figure 1: Example screenshot from each phone layout pattern. The *Accuweather* screenshot is edited from the Android Play Store Page [1].**

We categorize them as a **content-based UI pattern**, organizing only static or dynamic content, without any navigational components. In thinking of the UI as a graph, screens with the content-based UI patterns are nodes with only one edge. The only way to navigate to a different screen from a content-based UI screen is to return to the parent of the current screen. By definition, content-based patterns are distinct from navigational patterns, and thus, there are no screens in both categories.

We define two content-based UI patterns:

(1) A **static content** pattern displays a combination of static text and/or images. The user can only read the text or see the images; there is no interactive component beyond swiping up or down, or returning to the prior screen. For example, the "about" screen of *Runkeeper* (Figure 1g) displays static information, including the "getting started guide" and helpful tips. Even though the user can close the "about" screen, they cannot navigate to other destinations from this screen.

(2) An **interactive content** pattern displays dynamic information using a combination of text, images, and interactive components that modify the information displayed. The users can interact with the components using a tap or a sensor. For example, the *D&D Dice* app (Figure 1h) has buttons for the

user to roll one or more dice; when the user taps a button, the only update to the screen is to recalculate and display the points from the roll. In some, this pattern can also overlap with the skeuomorphic navigational pattern, but without the navigational destinations.

## 3 WATCH UI LAYOUTS

In mapping a phone UI with more components onto a smaller watch UI, one typically either removes some functionality, or breaks up one screen on the phone into multiple screens on the watch. We identify four general categories of watch UI patterns in our data set: List, Scroll, Interactive Card, and Informative Card. These four categories are determined based on the type of objects (`ViewGroup` and `View` in Android parlance) on the screen. We identify a fifth, special category of Skeuomorphic layouts, that can only be identified visually. Each watch UI view falls into precisely one of these categories, explained below:

- A **List** (Figure 2a) pattern is comprised of vertically scrollable components, evenly arranged list items. A list item in this pattern can be an image, a text label, or a combination of both. Examples of a List pattern in an app include options under a "settings" menu or items in a shopping list. In our data set, more than 90% of these items are clickable.
- A **Scroll** pattern (Figure 3) is comprised of vertically scrollable components, but not evenly arranged. The height and visual style of these components can vary widely, and developers tend to select the Scroll pattern when there is not as strong of a semantic relationship among the elements as items under a list. For example, *Google Fit* uses a Scroll to include such varied content as today's goals, heart rate table, detailed activity statistics, and settings, shown in Figure 3.
- An **Interactive Card** pattern (Figure 2b) is comprised of one or more clickable UI components, which can be images or text. An Interactive Card is not scrollable. One typical usage of this layout pattern is a music player. There is not a clear relationship between the use of an Interactive Card on the watch and any specific phone layout pattern above; later sections investigate how these patterns are translated in practice.
- An **Informative Card** pattern (Figure 2c) is comprised of one or more non-clickable UI components, and is not scrollable. This layout is typically used solely to display information, such as the weather and news. An Informative Card is similar to the Static Content pattern on the phone, as both only display non-interactive information. as the only functionality.
- A **Skeuomorphic** pattern (Figure 2d) is a special category that must be identified visually, where the UI mimics a real-world object or tool. This pattern is similar to a Skeuomorphic pattern on the phone. For instance, Figure 2d shows a compass app, which mimics the appearance of a physical compass.

## 4 DATA COLLECTION METHODOLOGY

In order to study the current state of practice in adaptation, we curated a dataset drawn from 101 Android apps with both a phone and a watch interface, from 24 app categories. From these apps, we collect both the view hierarchies of both the phone and watch, as well as screenshots of each screen, and manually labeled correspondences between individual UI elements. This Section explains how we compiled the dataset and its contents.

**App selection.** We select apps with both a watch and phone counterpart. There is no definitive count of watch applications on the Wear OS Play Store, but a popular mirror for free apps, APKMirror, counts roughly 600 free watch apps [3]. Our data set includes 101 companion apps, roughly 16.8% of this total population.

We selected these apps prioritized by overall popularity from the "Top Free Apps" chart in Google Play Store. We removed applications that were simple *watch faces*, which put app-specific content on the first "clock" screen of the watch; this was the most popular category of watch apps by far, so our data set likely covers a large fraction of the remaining apps. We also dropped apps that required significant measures to authenticate that the end-user was a real human and customer, such as banking apps and password managers. We did create accounts on free services that required minimal user validation, such as a working email. Finally, we excluded apps that simply did not work on our test watch, or that required additional in-app purchases. We used the Google Nexus 5, Nexus 6 phone, and Huawei Watch 2 Sport Smartwatch as test platforms.

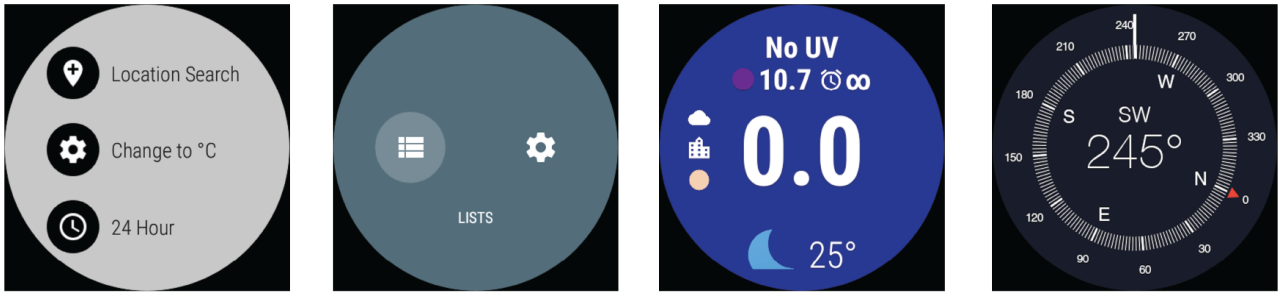
**Capturing Screenshots and XML.** We developed a tool, called **UICrawler**. UICrawler automatically explores every UI in each app and captures the screenshot and view hierarchy in XML.

UICrawler leverages the Android testing API, *uiautomator* [11], which can automate UI exploration by mimicking user events, such as clicking and inputting text. UICrawler does a depth-first-search traversal of all possible UI elements, starting at the initial screen of the app. During each UI traversal, UICrawler also saves a screenshot and the corresponding XML file for each UI. In UICrawler, each UI node maintains a list of child nodes to keep track of exploration and progress, as well as a parent node pointer for navigating back up the graph. UICrawler differentiates each UI by assigning each UI a unique signature, based on the UI class name and each UI's navigation path from the root UI.

UICrawler has a limitation that it can only navigate and crawl the UI on a single device. A screen on one device may only be reachable via interaction on another device, such as a watch screen that can only be reached by pressing a button on the phone. Typically this involves a dialogue on one device advising the user to take an action on the other device, and this manifests in UICrawler hanging. We observed about ten instances of this in our set of apps, and manually collected these interactions. It is possible that more subtle screens were overlooked in the process. We mitigated this issue by asking the students who labeled the data (described below) to spend time using the apps themselves and note any screens that were missing in the captured data.

At the end of this crawling process, we have an initial dataset of 716 phone and 310 watch screen captures (pairs of XML and images).

**Labeling Corresponding UI Elements.** We then manually labeled the design pattern each screenshot follows, according to the description in §2 and §3. We further identify each UI element that appears in both a phone and watch UI for the same app.




(a) List (b) Interactive Card (c) Informative card (d) Skeuomorphic

**Figure 2: Example screenshots for four watch patterns.**

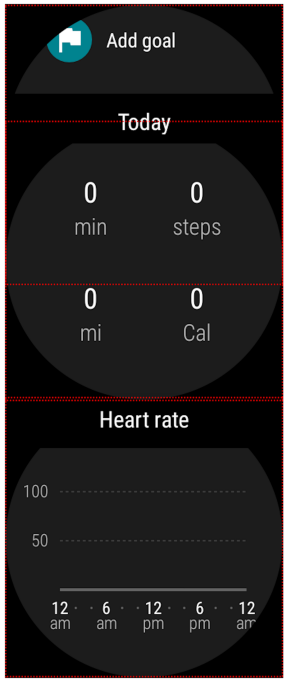



**Figure 3: Screenshots of the scroll interface in *Google Fit*. Each red dotted rectangle indicates how much is visible at one time.**

We had a team of three students manually do this categorization, including labeling outliers that did not follow these patterns or match in both the phone and watch. Each student was provided with a computer that had UIAutomatorViewer—a tool within the Android Studio developing environment that shows how portions of the app's XML correspond to portions of the screenshot. In addition to screenshots, each student was provided with a phone and watch that they could use to try each app and check whether elements correspond.

None of the students had any prior experience with smartwatches or Android development. We trained the students on using the UIAutomatorViewer tool before they start collecting data. On average, and excluding training time, each student spent around 15 hours labeling these screen captures and UI elements. This plan was reviewed by our IRB in advance of any work and exempted from review, as it does not involve human subjects.

Each screenshot and element was labeled by at least two of the three students, to detect any inconsistencies or subjective disagreements. In the case of a disagreement, we had additional participants (including senior researchers) review, independently label the items, and take the majority's opinion.

**Dataset.** The resulting dataset first includes a set of labeled pairs of watch and phone screenshots, where at least one UI element on the phone appears in the watch screenshot. This results in 140 phone UI screenshots and 165 watch screenshots. There were a number of reasons that we could not match some phone screens to watch screens:

- The watch app does not implement any overlapping functionality as the phone app. For example, the watch version of *Bitdefender* is actually called "Where is my phone" in the app list, and works as a button triggering the phone's alarm sound to reveal the position of the phone.
- The watch and phone apps contain a completely different presentation of the same information. For example, *ViewRanger* has turn-by-turn navigation in the text on its watch app, but an interactive map on its phone app.

The second part of our dataset is a set of records for each UI element that appears on both a phone and watch screenshot. Here, a record consists of the app's package name, a reference to the containing phone screenshot and the bounds of the element within that screen, and a reference to the containing watch screenshot and the bounds of the element within that screen. The boundary is used to identify the component within the View hierarchy XML.

In total, the dataset includes 426 unique phone-watch UI component pairs. Each app has an average of 4 UI component pairs.

## 5 UI ADAPTATION PATTERN STUDY

We organize the analysis of our dataset based on the eight phone UI patterns described in Section 2. For each UI pattern, we describe both trends and interesting exceptions in how these UIs are adapted to a watch. Screenshots of the example apps highlighted in this section can be found in our data set.

In order to create a consistent "look and feel", Android Wear Design Guidelines introduce components, visual styles, and interaction patterns in an Android wear app. These guidelines do not

| Phone Pattern | Sample Size | Adaption Strategy to the watch |
|---|---|---|
| Springboard | 7 | A List(4), an Interactive Card(2), a combination of Scroll and Interactive Cards(1). |
| List Menu | 37 | A List(18), one or multiple Cards(14), a Scroll(3), a List with Cards(2). |
| Dashboard | 21 | Informative Cards(9), Interactive Crads(7), a List(4), a List and Cards(1). |
| Gallery | 6 | A List(4), Interactive Cards(1), a List and an Interactive Card(1). |
| Side Drawer | 5 | A List(2), Interactive Cards(2), a List and an Informative Card(1). |
| Skeuomorphic | 6 | Skeuomorphic(5), an Interactive Card(1). |
| Static Content | 6 | One or more Informative Cards(5), and a Scroll(1). |
| Interactive Content | 52 | Interactive Cards(36), an Informative Card(8), a List(4), a List and Cards(4). |

**Table 1: The UI Adaption Strategies based on each distinct phone UI pattern**

specify patterns for realizing a given functionality or for porting an Android phone app on to a watch. This study addresses this gap by investigating the design patterns for overall content and functionality in both phone and watch apps.

The distribution of phone patterns in the dataset is different from the average distributions of phone patterns because some patterns are more likely to be mapped on to the watch. Watch apps tend to display content directly and with fewer navigation steps than the phone counterpart. In phone apps, it is common for important content to be reachable from multiple components, including the Bottom Navigation and Side Drawer; a corresponding watch app may simplify overall navigation to have one path to the same content, such as placing it on the initial screen of the app. As a result, one element on the watch app may take the place of multiple elements in the phone counterpart, leading to a different overall distribution.

## 5.1 Springboard

The Springboard pattern has declined in popularity over the last several years [22], and our data set only includes seven examples of the Springboard pattern. The most common watch pattern (four of seven examples), converts the Springboard to a List pattern on the watch.

Rather than map all options from a Springboard onto a List, the remaining apps in our sample map a subset of the phone's Springboard to other patterns on the watch (two to an Interactive Card and one to a combination of Scroll and Interactive Card). These apps place the features that are likely to be accessed on the watch in a more prominent position. For example, in the *Runkeeper* phone app, the button to start an exercise appears in a Springboard with options such as Activity, Music, and Statistics. On the watch, however, this start button is mapped separately in an Interactive card with easier access, on the assumption that users will more commonly use their watch to start and stop a workout than to view statistics. As another example, *Google Fit* also moves the "Start activity" to a separate Interactive Card, placing other options ("Add goal", "Add activity", "Start activity", "Log your weight") in a separate, Scroll pattern.

> The first screen on the app, both phone and watch, typically includes the most important content. The decision about subsetting content on the watch is driven by the relative importance or utility of each item.

## 5.2 List Menu

Our dataset includes 37 phone screenshots that use the List Menu pattern. As with the Springboard, the relative importance of items drives how the List Menu is mapped to the watch, which can legibly display only a few items at a time.

About half (18) of these samples map to a single List pattern on the watch. In general, the list contents are identical on the phone and watch, except when the list on the phone includes a preview of its content; on the watch, this preview is removed, as it is usually too small to see.

An additional 14 phone screenshots map onto one or more Cards on the watch. These samples are evenly split between Informative and Interactive cards, and this correlates with static versus dynamic content, respectively. There is no clear trend in whether the developer selects a List or a set of cards. An example of the list-to-card strategy comes from *Sports Tracker*. The UI on the phone shows a list of past activities, with detailed statistics directly embedded in the list item, including time, distance and speed; one can click on a list item to see additional details about that activity, like the actual running route on the map. When this list item mapped to the watch, the designers used an Informative Card pattern to show these detailed data on one card. Mapping to an Informative Card combines the list item and its detailed page together, while dropping less important content on the watch.

The more interesting exceptions are cases where it seems that the designers wished to break the uniform presentation of a list to draw attention to more important design elements, such as the most recent note or most frequently used item. There are three cases in our dataset where a List Menu is replaced with a Scroll on the watch, and two cases where a List Menu on the phone is mapped onto a combination of a List with one or multiple Interactive Cards on the watch. in a related list. For example, *Pandora* creates a single List Menu for both tabs showing my stations and browsing others, as well as then showing with the full list of stations below; on the watch, they separate the first tabs to two Interactive Cards, and the station list remains a separate List on the watch. Users need to change between Cards to reach different music station lists.

> Lists on the phone are typically mapped to Lists or a set of Cards on the watch. Cards give the developer more flexibility to display content non-uniformly, such as for emphasis or to provide different levels of detail in different items.

## 5.3 Dashboard

In our dataset, 21 phone screens use the Dashboard pattern, and this is typically the first screen on both the phone app and watch app. The most visually similar watch pattern is a Scroll; to our surprise, none of our samples implemented this mapping. We suspect the reason for this is that it is easy for a Scroll to become very long and unwieldy for users—undermining the goal of collecting key information in a single, curated screen. Across these patterns, more than three-fourths of these watch apps reduce the amount of content in adapting the Dashboard for the watch.

We found that most instances of the Dashboard on the phone were mapped to a single or multiple cards on the watch. On the phone, one Dashboard component usually occupies at least one-fourth of the display and shows detailed information. To realize the same functionality on the watch, the whole watch screen is typically used to display the same amount of information. Thus, an Informative card is the most popular pattern with nine samples used to translate a phone dashboard onto a watch. For the same reasons, another seven of the samples use an Interactive card to display dynamic elements. In terms of subsetting contents, if all of the phone's content will not fit on the watch screen, lower-priority content is dropped.

Four samples in our dataset turn a Dashboard into a List on the watch. In general, these were cases where one could use as easily to present the same content as a List Menu pattern on the phone, and, we suspect, there was less need to prioritize certain UI elements, but rather to display a summarized list. In some cases, the list may become less detailed so that more items can fit on a single watch screen without scrolling. For instance, in *Space Launch Now*, only spacecraft category, name, launch date, and location are mapped to watch. Other information that appears in the phone's Dashboard is not present on the watch, so that the user can see multiple spacecrafts launch information displayed in a List pattern.

Finally, *Accuweather* is a single case of mapping a Dashboard to a List and Cards. The phone version shows all important weather information (current weather, daily forecast, etc.) in a Dashboard, but the watch version uses a List for the daily forecast and an Informative Card for current weather and other information.

> Dashboards on the phone typically store the most important content in one screen. The most common watch mapping is to one or more cards. Three-fourths of watch apps subset content based on importance to fit a smaller display.

## 5.4 Gallery

Our dataset includes 6 phone screens that follow the Gallery pattern. Recall that a Gallery includes a grid of larger images (not simple icons); at most, one larger image will display well on a watch form factor, these images will often be illegible in a list or grid.

Four of the samples are mapped to a List pattern on the watch, and one to the Interactive Card pattern. When the images are mapped to different Cards, they cannot be larger, meaning that the preview Card and the actual content are the same on a watch. As a result, Galleries are more often mapped to Lists.

When a Gallery pattern is mapped to a List, the text under the image on the phone often becomes the list items on the watch, rather than the images. On the phone, *Streamago* uses a Gallery to list current streaming users and show options including "Go Live"; on the watch, this is mapped to a single Interactive Card that only shows the options, but no user images. Since users cannot watch live streams on their watches, the watch app acts more like a controller for the phone.

The final example, *Google Keep*, adopts a mixture of a List and Interactive Cards. The phone app uses a Gallery pattern to show all notes a user created, resized to fit the column width. Unless the note is longer than the screen length, a user can typically see the whole content of a note. In the watch version, this screen becomes a List to show all of the notes, but with a limited preview area where users can only see the first line of the note. *Keep* also uses an Interactive Card for creating new notes, but with fewer note categories than those on the phone. To fit a smaller display, *Keep* removes options like "drawing freely" on its watch version.

> Galleries are not a good fit for watches, as there is simply not adequate screen area to give a visual overview of content. Adaptation strategies vary widely, based on the functionality of the screen and app. For instance, a control interface is more likely to use text in a list, whereas a content display may use a series of cards including images.

## 5.5 Side Drawer

A Side Drawer pattern is visually similar to a List Menu pattern, but helps the user navigate among screens in an app. Side Drawers are usually not the primary navigation interface for the whole app, but rather, serve in a secondary, shortcut role. In translating a Side Drawer to a watch, however, this screen often becomes the primary navigation mechanism. Our dataset has five sample phone screens.

Two of these samples map onto a List on the watch. Mapping a Side Drawer to a List is similar to mapping a List Menu on the phone to a List on the watch, but without removing any data from the original UI. For example, in the *Bring!* app, the Side Drawer contains user-created shopping lists, as well as an editing UI with different kinds of shopping items. However, on the companion watch app, the first screen users see is a List containing all of their shopping lists. This change in navigation experience, from a secondary menu to the primary screen, also matches the typical use of this app: users often create shopping lists on their phone, and then check on the watch while in the store.

Two of the samples map onto a set of Interactive Cards. In these examples, the navigational content remains secondary—users do not see these cards when they launch the app, but they are brought up by swiping down. The Interactive Card that is brought up retains the same content as on the phone, and each option leads to another part of the app. This pattern of using a special gesture to bring up an Interactive Card most closely approximates the Side Drawer pattern.

The fifth screen, from *Google Keep*, uses a combination of a List and an Informative Card. In the phone app, a user can switch among multiple accounts by just tapping on the avatars on the top of the Side Drawer. However, a user can only see the current account first on a Card in the watch app. By tapping it, the user will see the switch account menu showing in a List.

The side drawer pattern on the phone often moves from a secondary to a primary navigation role on the watch, and is often approximated with a special icon or gesture, such as swiping down from the top.

## 5.6 Skeuomorphic

Our dataset includes 6 phone screens that follow the Skeuomorphic pattern. Five of these directly map onto a similar, Skeuomorphic pattern on the watch. One of these, *Compass2D*, also adds a new Informative Card containing current position information in text, in addition to a skeuomorphic compass showing on the watch.

The exception is a case where the original object does not fit within the watch display. Specifically, the app *SchoolBell*, which emulates a school bell on the phone following a skeuomorphic pattern, is converted to a simple button in an Interactive Card on the watch.

Skeuomorphic designs tend to be directly adopted on the watch, provided they fit in the form factor.

## 5.7 Static Content

The dataset includes six phone screens following the Static Content pattern. Five of them turn into one or more Informative Cards, and one is converted to a Scroll. The app *Woman Bible* uses a Scroll pattern to show the book's text. In other cases, a typical content that will be mapped from a Static Content to an Informative Card is a QR Code. In our dataset, the *Business Card* app actually mapped the QR Code on the business card it displays on the phone to the watch. Though not included in our dataset, we note this use of QR codes in an Informative Card is common in digital payment services, such as in the *Alipay* app.

## 5.8 Interactive Content

Our dataset contains 52 phone screens with Interactive Content. About three-fourths are mapped to a watch screen following the Interactive Card pattern. Five of them are a music player interface on both the phone and watch. We notice that Google provides a music player template in an Interactive Card for developers and in practice, all player apps use this template in their watch apps. Further, eight screens become an Informative Card, four are mapped to a List, and four are mapped to a List and Cards.

Mapping Interactive Content pattern to a List usually means that the designer creates a new navigation for the current contents. The *Runmore 5K Trainer* app shows the "start activity" button, a time indicator, and tabs containing the number of days on its phone app, but maps only the numbers of days on its watch version and each could start activity immediately. Sometimes new navigation options are derived by the watch itself. In app *Ride with GPS*, the "start a ride" button on the phone is mapped to a List containing two options "Start Ride on Phone" or "Start Ride on Watch".

There are examples where functionality is not subset, but Interactive Content is divided into multiple screens. For instance, the *Vivint* app on the phone has an Interactive Content with both a text message indicating the state of the home alarm and multiple sliders to open/close doors. The watch version places the text message in an Informative Card and the door controls in a different List.

## 5.9 Special Components

Most phone app UIs include special components, including a top and bottom bar.

A bottom navigation bar typically includes icons for quick navigation to major functionality in the app [13], and are present in most apps. An app typically has only one bottom navigation bar, which is the same across most of the screens. In Google's Wear OS Anatomy, there is a *navigation drawer* component that serves a similar purpose, although it is not visible by default—users need to perform a swiping down gesture to show the bar. In our dataset, in most cases a bottom navigation bar on the phone is mapped to a navigation drawer on the watch.

A floating action button performs the primary, or most common, action on a screen [15], such as drafting a new email in an email app. It appears in front of all screen content, typically as a circular shape with an icon in its center. Typically, a screen only has one floating action button. In our dataset, a floating action button is typically mapped to a single Interactive Card when it is on the opening screen on the phone. This is because such primary action is usually accessed most frequently on the watch as well. If it is not mapped to a Card separately, it is usually mapped to an inline action button on the top of the content view.

A top app bar [12] generally includes contents related to the current screen. The developer typically places branding, screen titles, navigation, and actions in the top app bar. Although a top app bar is an important component of an app on the phone, components in a top app bar are usually not mapped to the watch. When visual space is at a premium many of these components can be removed, and essential, contextual actions are instead added to the content view.

## 5.10 Mapping Phone Screens Onto Watch Screens

Because the watch has a smaller form factor than the phone, a significant number of phone screens do not have a one-to-one mapping to a watch screen; content may be subsetted, or one phone screen is divided into multiple watch screens. In practice, we find that there are three kinds of mappings between phone and watch screens:

- One phone screen to one watch screen (103 cases, 82%). This result matches our expectation that designers most commonly choose to map a single phone UI to 1 or more watch UIs, picking a subset of UI components from the phone UI. An important caveat is that some phone UIs are mapped onto multiple watch screens, where multiple watch List or Scroll screens are composed to form a single, scrollable list, transparently to the user. As long as these are the same logical list or scroll (i.e., users can access all of these screens by swiping up or down), we treat this as a one-to-one mapping. This happens most commonly when adapting the first screen of the phone app, and the first screen is mostly navigational elements.
- One phone screen to multiple watch screens (20 cases, 16%). In 16 of these cases, the original phone screen is a Navigational UI. On the phone, these navigation screens tend to

have the most important content, such as key navigational elements, so designers place these elements on multiple watch screens.

- Multiple phone screens to one watch screen (3 cases, 2%). Since the watch has a smaller screen, mapping components from multiple phone screens to one watch screen usually requires a significant restructuring. We expect that the additional design effort required makes this pattern rarer. For example, in the *Fishbrain* app, the "Log Catches", "Fishing Forcast", and "Map Navigation" appear in different navigation levels in the phone app, but they appear on a single watch screen, probably because users use these two features more frequently on their watches. We hypothesize that cases, where the most important information is not on one screen on the phone indicate a design flaw, which is exacerbated by a more constrained form factor; in other words, the phone UI would likely be improved by a similar restructuring.

# 6 UI ADAPTATION TRENDS AND DESIGN GUIDELINES

This section discusses trends in the dataset and elaborates possible reasons behind these trends. The section then presents design guidelines for adapting a phone UI to a watch. These trends describe basic watch patterns that are general enough for all wearable devices with a small display: lists and cards.

## 6.1 Trends among patterns

**The majority of top-level navigation patterns are mapped to Lists or multiple Cards.** Such adaption is straightforward, as the navigation options turn into list items. Users can easily understand the new UI on the watch by recognizing the same icon or text even if the layout is changed. This trend does not hold for the Skeuomorphic pattern.

**The decision to use Cards instead of a List appears to be motivated by prioritizing a particular function on the opening screen.** This can be the "Start activity" button in a fitness app, or "the most recent QR ticket" in a movie theatre app.

**When mapping patterns containing previews to the watch, designers need to balance between the amount of information and the overall clarity of the UIs.** Phone UI patterns like the Dashboard or Gallery include a preview of information, whereas the Springboard and List Menu do not. Removing part of the preview information is common in adapting to the watch. Further, legibility is important, especially for users in motion [4], and designers reduce content to keep fonts and images sufficiently large.

**There is a natural correspondence between Side Drawer on the phone and a navigation drawer on the watch; yet three of the five applications in our data set did not map to this pattern.** As with a side drawer, a navigation drawer is hidden by default and revealed when a user does a downward "swipe" gesture [14]. Two of the five apps used cards that were placed in a navigation drawer, whereas the other three moved content from a secondary navigation interface to a primary navigation interface.

## 6.2 Design Guidelines

Based on the observations of current practice, this subsection presents guidelines to assist developers in adapting a phone UI to a watch.

(1) **Most phone UI patterns have a natural watch counterpart, which should be used by default.** As detailed above, each pattern has a natural default counterpart on the watch. For example, designers typically map a phone Dashboard to a set of Cards on the watch. Although there are reasons to deviate from the default, the majority of cases do not.
(2) **Appropriately subset contents on the watch.** The watch has a much smaller screen, and subsetting is often necessary. This subsetting often requires a judgment call from the designer; in some cases there is redundant information to remove, such as a preview of the content, but in others, it requires an understanding of the relative utility of the content. This is probably the facet of UI adaptation least amenable to automation.
(3) **Use the navigation drawer on the watch properly.** §6.1 notes that watch apps are not using the navigation drawer, despite the fact that it is a natural counterpart to the Side Drawer on the phone. We believe secondary navigation interfaces likely have value and should likely be retained even on a smaller screen; this is best evaluated with user studies in future work.
(4) **Use a List for the Navigation pattern and a set of Cards for Content-based patterns.** The List pattern is the most common layout on watch apps, and users are familiar with this. Thus, it is a reasonable default for many navigation patterns. Similarly, for content-based patterns, the Card is best, as it avoids placing important content off-screen at the bottom of a long Scroll.

# 7 LIMITATIONS AND FUTURE WORK

We did not collect interactions related to components in our dataset. Adding UI components composition of patterns will increase both the complexity and the specificity of each category, leading to several sub-categories that will be less abstract and more close to the actual screens. We also believe the position of UI components is significant, especially for content-based UIs. For example, screens with buttons nearly evenly distributed (e.g., calculators) are often intended to convey a different relationship than screens with buttons having a clear hierarchy (e.g., music players). Further, interactions with components can trigger transitions between screens, or changes in part of the screen. Adding interactions could further help with categorizing components in context. We leave identifying strategies for mapping components for future work. Finally, our analysis does not factor in the semantics of an app's tasks or contents in identifying patterns; it is probable that apps with similar contextual or situational features may follow similar adaptation strategies. We leave the exploration of these various features for future work.

Another limitation of this project is the focus on free applications. It is possible that paid content has a significantly different pattern, although exploring this may be both labor-intensive and require personal information. It is possible that one might be able to address this in future work with a tool, such as a debugger, that

can manipulate control flow and data values to reach screens that one would not be able to reach without paid content, although this may require more manual validation than any time it is saving.

In future work, we hope to combine these observations with watch app generation tools, such as UIWear [29], to realize fully automatic, or an improved semi-automatic, watch app generation. On the phone side, users could choose UI components they want on a watch, and, based on the observations in this study, recommend watch UI templates for the watch.

## 8 RELATED WORK

This section surveys related work on GUI design patterns, with a particular focus on extending UIs to different platforms and existing UI datasets on these adaptation patterns.

**Cross-device UI platforms:** As wearable devices are increasing in popularity, a natural need arises to easily extend the UI of the phone onto a wearable, or adapt the UI to run the app independently of a phone, but on a different form factor. A number of research projects have looked at issues relating to the engineering complexity of building cross-device apps.

One category of this research assumes a human will design the interface, and the research contribution is reducing the complexity of coordinating a UI across multiple devices. For instance, UIWear [29] observes that extending a phone app's interface onto a wearable involves developing an ad hoc networking protocol that relays state between the devices. UIWear observes that most of the effort can be automated, except for the design itself. Similarly, Conductor [16] presents a framework for orchestrating a UI that spans a large array of devices. PageTailor [5] and Highlight [23] simplify the process of transferring from a PC version webpage to a smartphone version webpage. Gjerlufsen et al. [10] present a middleware for developing multi-screen apps, which decouples application behavior and data. Nebeling et al. [20, 21] provide platforms to generate UIs across devices semi-automatically, in which users only need to specify their preferred UI customization across devices. GUMMY [18] generates an initial design for a new platform from existing user interfaces by manually specifying the layout. WinCuts [28] presents a system that allows users to replicate arbitrary regions of existing windows into independent windows view of a region of the source window. WinCuts also allows users to share those windows across devices.

In the presence of heterogeneous form-factors, or devices that come out after the app is designed, some projects have investigated techniques to help developers place UI elements on the device(s) best suited for that element. For instance, Panelrama [30] identifies several key usability factors, such as device size or keyboard type, that a programmer assigned to each UI screen (or panel in their nomenclature). By matching these qualities to the underlying device, Panelrama distributes elements to each device automatically. Mori et al. [19] present a tool, TERESA, which abstracts user interfaces based on their task models [25], and then generates concrete user interfaces for different devices automatically.

In total, adapting an app's user interface from one form factor to another is very much an art. Broadly speaking, given a design, prior research simplifies the construction of creating interfaces that span devices, or provides some heuristics to assist the developer in placing elements. It is an open question the degree to which the design of UI adaptation can be automated; this paper lays a foundation for future studies by measuring the state of practice.

**Studies on UI design patterns:** A number of studies define and categorize UI design patterns, but these work does not consider patterns for adapting UIs to new devices with different form factors.

In particular, a number of studies have considered app design for smartphones. For instance, Neil et al. [22] introduces more than 90 mobile app design patterns crossing 11 categories, based on functionality, which is one of our inspirations when categorizing mobile and wearable UIs. Calvary et al. [6] introduces a framework for classifying UIs and usage contexts. Specifically, the framework expresses how a change of context is considered and supported in a context-sensitive user interface. Nilsson et al. [24] introduces a collection of UI design patterns for mobile applications. These patterns are grouped along three main axes: utilizing screen space, interaction mechanisms, and overall design. This study also introduces a general guideline on how to use the design patterns at suitable circumstances. Gallud et al. [9] introduces an emerging topic of the concept of Distributed User Interfaces (DUIs) that the UI can be split and composed, moved, copied or cloned among devices running the same or different operating systems. SahamiShirazi et al. [27] analyzed over 29K UI layout files so that it concluded the most frequent interface elements and identify combinations of interface components from 400 free Android apps. This study identifies general patterns for the combinations of layout styles.

**Building UI Datasets:** This paper is inspired in part by prior work that collected and analyzed datasets of UI design patterns. These studies provide useful data for the community, but answering different questions. The most relevant work analyzes data sets of smartphone app UIs and web pages. For example, Alharbi et al. [2] collected more than 25 million UI elements by tracking 24k smartphone apps for 18 months and analyzed how UI components changed though the period. Kumar et al. [17] collected millions of design elements from 100,000 Web pages to understand design demographics, automate design curation, and support data-driven design tools. RICO [7] builds a mobile app UI dataset that covers over 9.7k Android apps. Similarly, ERICA [8] captures how a UI changes in response to user interactions, automatically generating datasets of user interaction traces.

## 9 CONCLUSION

This paper presents the first dataset of real-world applications with both a phone and watch component, and analyzes the trends in how the UI is adapted. One-to-one mappings of phone screens onto watch screens are common, and it is likely that one can automatically produce at least a first draft of a watch UI from a phone UI for the majority of cases. Apps that spread content from one phone screen onto multiple watch screens tend to do this for important navigation elements, where buttons and other important elements are replicated for convenience, or promoted onto the first screen of the app.

Subsetting watch content and functionality is common, especially for non-essential details or decorations. In general, the subsetting process is driven by the importance of the content more than the amount of data on the screen. It is likely that, with additional

profiling of how end-users typically navigate the phone app, this step of subsetting could also be potentially automated.

In future work, these data and observations can ground future studies on the efficacy of automated or semi-automated UI layout strategies, as well as evaluating user satisfaction with different common design decisions.

## ACKNOWLEDGMENTS

We thank Bhushan Jain and the anonymous reviewers for their insightful comments on previous drafts. We thank Daniel Koenigsperger, Rachel Poppalardo, and Marigrace Seaton for help collecting the data set in this paper. This work was supported in part by NSF grants NSF CNS-1717973 and CNS-1718491.

## REFERENCES

[1] AccuWeather. 2019. AccuWeather Play Store Page. https://play.google.com/store/apps/details?id=com.accuweather.android&hl=en_US.

[2] Khalid Alharbi and Tom Yeh. 2015. Collect, Decompile, Extract, Stats, and Diff: Mining Design Pattern Changes in Android Apps. In *Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services* (Copenhagen, Denmark) *(MobileHCI '15)*. ACM, New York, NY, USA, 515–524. https://doi.org/10.1145/2785830.2785892

[3] APKMirror. 2020. APKMirror. https://www.apkmirror.com/.

[4] Apple. 2019. Apple WatchOS. http://www.apple.com/watchos/.

[5] Nilton Bila, Troy Ronda, Iqbal Mohomed, Khai N. Truong, and Eyal de Lara. 2007. PageTailor: Reusable End-user Customization for the Mobile Web. In *Proceedings of the 5th International Conference on Mobile Systems, Applications and Services* (San Juan, Puerto Rico) *(MobiSys '07)*. ACM, New York, NY, USA, 16–29. https://doi.org/10.1145/1247660.1247666

[6] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt. 2003. A Unifying Reference Framework for multi-target user interfaces. *Interacting with Computers* 15, 3 (2003), 289 – 308. https://doi.org/10.1016/S0953-5438(03)00010-9 Computer-Aided Design of User Interface.

[7] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017. Rico: A mobile app dataset for building data-driven design applications. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. 845–854.

[8] Biplab Deka, Zifeng Huang, and Ranjitha Kumar. 2016. ERICA: Interaction Mining Mobile Apps. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (Tokyo, Japan) *(UIST '16)*. ACM, New York, NY, USA, 767–776. https://doi.org/10.1145/2984511.2984581

[9] Jos A. Gallud, Ricardo Tesoriero, and Victor M. R. Penichet. 2011. *Distributed User Interfaces: Designing Interfaces for the Distributed Ecosystem.* Springer Publishing Company, Incorporated.

[10] Tony Gjerlufsen, Clemens Nylandsted Klokmose, James Eagan, Clément Pillias, and Michel Beaudouin-Lafon. 2011. Shared Substance: Developing Flexible Multi-surface Applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vancouver, BC, Canada) *(CHI '11)*. ACM, New York, NY, USA, 3383–3392. https://doi.org/10.1145/1978942.1979446

[11] Google. 2018. Android uiatomator API. https://developer.android.com/reference/android/support/test/uiautomator/package-summary.

[12] Google. 2018. App bars: top. https://material.io/design/components/app-bars-top.html.

[13] Google. 2018. Bottom navigation. https://material.io/design/components/bottom-navigation.html.

[14] Google. 2020. Android Wear System Overview – Anatomy. https://designguidelines.withgoogle.com/wearos/system-overview/anatomy.html.

[15] Google. 2020. Buttons: floating action button. https://material.io/components/buttons-floating-action-button/buttons-floating-action-button.html.

[16] Peter Hamilton and Daniel J. Wigdor. 2014. Conductor: Enabling and Understanding Cross-device Interaction. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. ACM, New York, NY, USA, 2773–2782. https://doi.org/10.1145/2556288.2557170

[17] Ranjitha Kumar, Arvind Satyanarayan, Cesar Torres, Maxine Lim, Salman Ahmad, Scott R. Klemmer, and Jerry O. Talton. 2013. Webzeitgeist: Design Mining the Web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Paris, France) *(CHI '13)*. ACM, New York, NY, USA, 3083–3092. https://doi.org/10.1145/2470654.2466420

[18] Jan Meskens, Jo Vermeulen, Kris Luyten, and Karin Coninx. 2008. Gummy for Multi-platform User Interface Designs: Shape Me, Multiply Me, Fix Me, Use Me. In *Proceedings of the Working Conference on Advanced Visual Interfaces* (Napoli, Italy) *(AVI '08)*. ACM, New York, NY, USA, 233–240. https://doi.org/10.1145/1385569.1385607

[19] Giulio Mori, Fabio Paternò, and Carmen Santoro. 2003. Tool Support for Designing Nomadic Applications. In *Proceedings of the 8th International Conference on Intelligent User Interfaces* (Miami, Florida, USA) *(IUI '03)*. ACM, New York, NY, USA, 141–148. https://doi.org/10.1145/604045.604069

[20] Michael Nebeling. 2017. XDBrowser 2.0: Semi-Automatic Generation of Cross-Device Interfaces. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. ACM, New York, NY, USA, 4574–4584. https://doi.org/10.1145/3025453.3025547

[21] Michael Nebeling and Anind K. Dey. 2016. XDBrowser: User-Defined Cross-Device Web Page Designs. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. ACM, New York, NY, USA, 5494–5505. https://doi.org/10.1145/2858036.2858048

[22] Theresa Neil. 2014. *Mobile design pattern gallery: UI patterns for smartphone apps.* O'Reilly Media, Inc.

[23] Jeffrey Nichols, Zhigang Hua, and John Barton. 2008. Highlight: A System for Creating and Deploying Mobile Web Applications. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology* (Monterey, CA, USA) *(UIST '08)*. ACM, New York, NY, USA, 249–258. https://doi.org/10.1145/1449715.1449757

[24] Erik G Nilsson. 2009. Design patterns for user interface for mobile applications. *Advances in engineering software* 40, 12 (2009), 1318–1328.

[25] Fabio Paterno. 1999. *Model-Based Design and Evaluation of Interactive Applications* (1st ed.). Springer-Verlag, Berlin, Heidelberg.

[26] Aaron Pressman. 2018. Why Smart Wearables Will Be One of the Tech Industry's Few Bright Spots. http://fortune.com/2018/12/17/wearables-smartwatch-apple-fitbit-garmin/.

[27] Alireza Sahami Shirazi, Niels Henze, Albrecht Schmidt, Robin Goldberg, Benjamin Schmidt, and Hansjörg Schmauder. 2013. Insights into Layout Patterns of Mobile User Interfaces by an Automatic Analysis of Android Apps. In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (London, United Kingdom) *(EICS '13)*. ACM, New York, NY, USA, 275–284. https://doi.org/10.1145/2494603.2480308

[28] Desney S. Tan, Brian Meyers, and Mary Czerwinski. 2004. WinCuts: Manipulating Arbitrary Window Regions for More Effective Use of Screen Space. In *CHI '04 Extended Abstracts on Human Factors in Computing Systems* (Vienna, Austria) *(CHI EA '04)*. ACM, New York, NY, USA, 1525–1528. https://doi.org/10.1145/985921.986106

[29] Jian Xu, Qingqing Cao, Aditya Prakash, Aruna Balasubramanian, and Donald E. Porter. 2017. UIWear: Easily Adapting User Interfaces for Wearable Devices. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking* (Snowbird, Utah, USA) *(MobiCom '17)*. ACM, New York, NY, USA, 369–382. https://doi.org/10.1145/3117811.3117819

[30] Jishuo Yang and Daniel Wigdor. 2014. Panelrama: Enabling Easy Specification of Cross-device Web Applications. In *Proceedings of the 32nd Annual ACM Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. ACM, New York, NY, USA, 2783–2792. https://doi.org/10.1145/2556288.2557199