Task Complexity Analysis: A Mobile Application Case Study

Sami Nieminen

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 25.5.2022

Thesis supervisor:

Prof. Antti Oulasvirta

Thesis advisor:

M.Sc. (Tech.) Timo Mätäsaho

Aalto University
School of Electrical Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Sami Nieminen

Title: Task Complexity Analysis: A Mobile Application Case Study

Date: 25.5.2022 Language: English Number of pages: 7+67

Department of Communications and Networking

Major: Human-Computer Interaction Code: SCI3097

Supervisor: Prof. Antti Oulasvirta

Advisor: M.Sc. (Tech.) Timo Mätäsaho

Interfaces are used to perform various tasks and in HCI particularly a human performs some task with the aid of a computer with the mediation of an interface. Tasks and interactions can be modelled as graphs, where the graph attributes contain information relevant to the understanding of the interaction task. Based on the interaction graph, it is possible to compute numerical task complexity measures that help compare the complexities of different tasks. However, determining the nature of tasks with manual evaluation is labor-intensive and does not work well with large-scale problems such as algorithmic design or evaluation of large datasets. In this work, we have shown that it is possible to algorithmically infer tasks structures from user interfaces and compute task complexity measures for the task structures represented by graphs. More specifically, the graphs contain descriptions of the components and the interaction modes associated with them, such as a tap. The graphs have been generated from Enrico dataset view hierarchies. The accuracy of the generated graphs is 53.5 % (90 % CI, 15 % ME). Majority of the errors are caused by issues in the underlying dataset. The computed task complexity measures include Wood's task complexity and Halstead's E measure. The task complexity measures behave in a fundamentally different way, and their applicability requires further validation. The results demonstrate that it is possible to computationally model and understand tasks performed by humans on interfaces based only on the interface structure. The ability to infer interface task structure as a graph and an adjacency matrix adds a novel perspective for analyzing and modeling user interfaces.

Keywords: computational interaction, graph analysis, mathematical modeling, mobile applications, task analysis, task complexity, user interfaces

AALTO-YLIOPISTO SÄHKÖTEKNIIKAN KORKEAKOULU	DIPLOMITYÖN TIIVISTELMÄ

Tekijä: Sami Nieminen

Työn nimi: Tehtävän Monimutkaisuusanalyysi: Mobiilisovellus Tapaustutkimus

Päivämäärä: 25.5.2022	Kieli: Englanti	Sivumäärä: 7+67

Department of Communications and Networking

Pääaine: Human-Computer Interaction	Koodi: SCI3097

Työn valvoja: Prof. Antti Oulasvirta

Työn ohjaaja: M.Sc. (Tech.) Timo Mätäsaho

Käyttöliittymillä suoritetaan lukuisia tehtäviä ja erityisesti HCI-alalla ihminen suorittaa tehtäviä tietokoneen ja käyttöliittymän avustuksella. Tehtäviä ja vuorovaikutuksia voidaan mallintaa graafeina, joissa graafin ominaisuudet sisältävät tehtävän ymmärtämiseen olennaista tietoa. Vuorovaikutusgraafin pohjalta on mahdollista laskea numeerisia tehtävien monimutkaisuusarvoja tehtävien monimutkaisuuksien vertailuun. Kuitenkin, tehtävien luonteen päättely manuaalisella evaluaatiolla on työlästä, jonka takia se ei toimi hyvin suuren skaalan ongelmiin kuten algoritmilliseen suunnitteluun tai suurien datasettien evaluointiin. Tässä työssä osoitettiin, että on mahdollista päätellä tehtävien rakenteet käyttöliittymistä ja laskea graafien esittelemille tehtävärakenteille tehtävän monimutkaisuusarvo. Tarkemmin graafit sisältävät kuvaukset käyttöliittymien komponenteista ja niihin liitetyistä vuorovaikutuksista, kuten sormella napautuksesta. Graafit on generoitu Enrico datasetin näkymähierarkioista. Generoitujen graafien tarkkuus on 53.5 % (90 % CI, 15 % ME). Enemmistö graafien virheistä johtuu datasetin ongelmista. Tehtävien monimutkaisuusarvoina on käytetty Woodin tehtävän monimutkaisuusmittaa ja Halsteadin E mittaa. Työn monimutkaisuusarvot käyttäytyvät fundamentaalisesti erilailla, ja niiden pätevyys vaatii lisävahvistusta. Tulokset osoittavat, että pystymme laskennallisesti mallintamaan ja ymmärtämään ihmisten käyttöliittymillä suorittamia tehtäviä pelkästään käyttöliittymän rakenteeseen nojaten. Kyky päätellä käyttöliittymän rakenne ja sillä suoritettavat tehtävät graafina ja vierekkäisyysmatriisina lisää uuden näkökulman käyttöliittymien analysoimiseen ja kehittämiseen.

Avainsanat: grafianalyysi, käyttöliittymät, laskennallinen vuorovaikutus, matemaattinen mallinnus, mobiilisovellukset, tehtävä-analyysi, tehtävän monimutkaisuus

Acknowledgements

If you initially do not know what is interesting or enjoyable to you in life, a good way to discover it is through design thinking. In life design thinking, you generate parallel lives as design concepts and rapidly try out many of the alternative life prototypes that are both interesting and feasible. This thesis is the culmination of such a process applied over several years, when many alternative life directions were tried out. During this time, the surface of the prism has become clearer, offering better guidance on which direction to take for the future. No one completes a journey worth undertaking alone. As a consequence, I will name some of the people who have made this journey worthy of undertaking.

Prof. Antti Oulasvirta for his excellence in teaching and guidance on performing research. What he teaches prepares his students well for future challenges and for solving problems both in academia and in working life. It has been a pleasure to be able to study under his instruction. I also want to especially thank Timo Mätäsaho for taking up the task of being my thesis advisor despite his other commitments in life and career. Seeing the intense effort he puts towards solving challenging problems at work has been inspiring, and it sometimes baffles me how he can switch so fast to talking about other things during coffee breaks to wind down from the intense effort.

My family, including six younger siblings, who make it worth it to strive for better in the hopes of showing them a good example. In the long run, a body that is housed in a healthy body tends to be more productive than the same mind housed in an unhealthy body. Thus, I want to thank the people at TrainTrue gym in Korea, who have created an environment and a community that supports getting both healthy and strong. I also want to thank all the people who joined me on hikes to the countless mountains in Korea, as this greatly helped me recharge mental energy after a week of researching and writing while doing exchange studies for another higher degree on the side. Lastly, I want to give a shoutout to my childhood friends who for some reason have stuck around through all these years while offering accountability, advice and humor.

Seoul, Republic of Korea, 25.5.2022

[illegible]

Sami Nieminen

Contents

Symbols and abbreviations

Symbols

E	Edge set
V	Vertex set
E	Halstead's E measure
H	Shannon's entropy
ρ	Autocorrelation
W	Information cue
f	Period
r	Precedence relation
$\alpha\beta, \gamma$	Weight factor

Operators

$\lvert \cdot \rvert$	Absolute value
$\surd$	Square-root
$\sum$	Sum

Abbreviations

ID	Identifier
HCI	Human-computer interaction
HTA	Hierarchical task analysis
TC	Wood's task complexity
UI	User interface

1 Introduction

Here we define the context and motivation for task complexity analysis in human-computer interaction (HCI), the research problem, contributions made by solving the problem, constraints of the research, and finally the thesis structure.

1.1 Motivation

In industrial projects where part of the product includes a mobile application or a website, a particular concern is how to make the interface easy to use or in similar terms less complex. In the field of HCI, various techniques and methods exist to improve an interface, where they can also be employed to make the interface less complicated. The importance of interface improvement is reflected in the many techniques employed by HCI experts. These techniques and methods include but are not limited to

- online controlled experiments like A/B testing [1, 2, 3, 4],
- expert based usability evaluations [5] such as heuristic evaluation, cognitive walkthrough, and think aloud method [6],
- various surveys and interviews that help understand the quality of an interface possibly with the help of predesigned scales such as system usability scale (SUS) [5].

A particular limitation with these methods is their employability for computational and algorithmic approaches. In particular, many of the methods require considerable amount of manual work to find ways to improve interfaces. In computational interaction [7] a focus is placed on using mathematical models and algorithms with capacity to explain and improve interaction. In this work, we aim to

1. demonstrate a mathematical modeling methodology for algorithmically inferring task and interaction structures in the form of interaction graphs from interfaces view hierarchies.
2. evaluate several task complexity formulas for their suitability to be used with the mathematical methodology used to infer the task and interaction structures.

Developing a mathematical method to consistently improve and analyze the task complexity presented by an interface can be used to further the field of computational interaction. Additionally, it may help various industrial projects where continuous development practices [8] especially in design and interface development areas could be sped up by having a consistent and reliable way to quantify the difficulty experienced by users when they want to perform a task with the aid of the interface. Once the difficulty of the task is quantifiable, it will be possible to determine which interface may be superior for aiding the user in accomplishing the intended task.

1.2 Research Problem and Questions

Within the scope of task complexity modeling under the field of computational interaction [7] there are two interesting initial research directions that can be pursued. The first research direction is constructing task models of already existing interfaces and obtaining task complexity measures for them. Firstly, by obtaining the models and measures, it will be possible to compare the interfaces against each other objectively in terms of how difficult accomplishing tasks on them will be. Secondly, by constructing task complexity models of UIs algorithmically, we will be able to study where the algorithms may face limitations when compared to human evaluation of a correct task model.

The second initial research direction of interest ties to constructing new UIs algorithmically. When developing interfaces algorithmically, a particular topic of interest is which objective function to use for maximizing certain aspects of user performance [7]. Task complexity measures can serve as an objective function to be optimized for when constructing interfaces algorithmically. Both the first and second research directions enable us to generate task complexity measures for interfaces, which can later be studied against task complexity experienced by humans.

In this work, we choose to focus on the first option of constructing task complexity models for already existing interfaces and computing task complexity measures for them algorithmically. More specifically, we aim to study how well a basic algorithmic approach is able to infer the task and interaction model when compared to human evaluation. In order to do this we will use Enrico [9] which is a dataset of mobile interfaces with guarantees of matching between UI screenshots and the view hierarchy. We aim to answer two questions in this work that are complementary:

1. What types of problems exist for inferring interaction graphs from view hierarchies in terms of dataset quality and inference algorithm performance?

2. Evaluating several task complexity measures for their suitability to be employed with the interaction graphs.

When evaluating the limitations with inferring interaction graphs from view hierarchies, we measure a numerical estimate of the frequency of the inference limitations with a confidence interval and margin of error. With evaluating the task complexity measure suitability, we will be focused on understanding if the task complexity measure can conceptually fit into the generated task models. A particular consideration from an evaluation perspective will be how well the model captures the interaction qualities in the task. If a particular task complexity model assumes the existence of irrelevant qualities from interface interaction perspective in the model, then it may be deemed a less suitable model.

When considering the research problem within the problem types in HCI research as discussed by Oulasvirta and Hornbæk [10] our work contributes in the conceptual-empirical study types in the HCI field. Firstly, creating a model to explain tasks algorithmically presented on interfaces constitutes a conceptual problem. The cause of the conceptual problem is implausibility [10] of manual task analysis to scale up to large scale problems, which calls for a computational interaction approach. A second conceptual problem in this work is understanding which task complexity algorithms capture the relevant interaction qualities presented on the graphs sufficiently. Lastly, the supportive research problem is an empirical problem, as we try to understand unknown effects on the correctness of the tasks presented on the interaction graphs when compared to what a human would assume the tasks to be. The unknown effects are classified to dataset limitations and inference algorithm limitations.

1.3 Research contribution

This thesis contributes to the field of computational interaction through a novel graph based method of inferring interface task structure and computing task complexity measures based on the view hierarchy of an interface. More specifically, we extract a directed graph and an adjacency matrix from the view hierarchy, which will contain a description of the tasks the user may perform on the interface. The directed graph and the adjacency matrix can then be used to compute measures of complexity for the interface. These measures can be used to automatically measure interface task complexity as they may be experienced by humans, especially once studied against measures of subjective task complexity and task performance.

Since we are inferring the interface task structure both algorithmically and evaluating them with a random sample against human understanding of the tasks on the interface, we are also able to study where an algorithm may have difficulties inferring the task structure correctly. The inference difficulties may be caused either by the underlying dataset or the inference logic of the algorithm. This will contribute towards understanding where algorithmic behavior may differ from the same task when performed by a human. Especially as it pertains to the understanding of the nature of tasks themselves. Understanding these differences will allow developing task inference algorithms more carefully with more human-like understanding of tasks when it comes to the design and evaluation of interfaces.

1.4 Research Topic Constraints

The topic area has been specifically restricted to the demonstration of applying a task inference algorithm on mobile application view hierarchies and evaluating suitable task complexity measures to be used with the interaction graphs inferred by the algorithm. This means we do not attempt to obtain ideal algorithmic accuracy for inferring task structure or complexity measures in this work. To the authors best knowledge, this is the first time task complexity has been inferred algorithmically from mobile application view hierarchies. As a consequence, demonstrating the basic principle of how to infer the task structure and identifying conceptually suitable task complexity measures for the interaction graphs shall take priority. The act of manually reviewing the algorithm performance against human understanding of tasks will contribute towards developing more accurate interface task structure inference algorithms in the future.

In this work, we focused on 1) researching what types of task inference problems can be found for an algorithm when compared to human performance on the same task, and 2) evaluating conceptually suitable task complexity measures for the interaction graphs. This means we have purposefully omitted to study how well the task complexity measures actually measure complexity as experienced by humans, or how they relate to task performance measures such as task completion time and error rate. These omissions are primarily due to this work being more focused on understanding algorithmic performance for an inference task rather than the human experience or performance, although human experience and performance form important parts of the HCI field.

1.5 Thesis Structure

In the upcoming chapters, we first discuss the required background to understand the construction of task structure based on interface view hierarchy and analysis of task complexity. The background literature discussion starts with an introduction to Rico [11] and Enrico [9] datasets, which will be used throughout this thesis. Secondly, we will discuss the basics of directed graphs, which form the basis for inferring task structure algorithmically. Then we move on to discuss task analysis, with an emphasis on task complexity analysis. Within task complexity analysis, we further place a specific emphasis on being able to analyze task complexity algorithmically, with the algorithm being suitable for the interaction graphs generated in this work.

In the third chapter, we will introduce the methodology required to infer task structure from interface view hierarchies. A particularly important concept here is how to represent interface components and the interaction modes connected to the components in graphs. We will also discuss the formulation of an adjacency matrix based on the graph and computing task complexity measures based on the graphs. In the fourth chapter, we will convert the procedure presented in the third chapter to an algorithmic inference procedure for Rico view hierarchies. In the fifth chapter, we will review the quality of the algorithmically inferred interaction graphs and also visualize the task complexity measures for the graphs. In the sixth chapter we discuss the results of the work, the limitations and potential future work directions. Lastly, we draw conclusions reached from the work in the seventh chapter.

2 Background

In this section, we focus on the required background to understand how to construct interaction graphs and compute task complexity measures based on the graphs. Firstly, we will introduce Rico [11] and Enrico [9] datasets as they will be used throughout this work. Secondly, we must understand the basics of directed graphs which can be used to model interactions and how the interactions combine into various tasks that can be performed on an interface. The basics of directed graphs are discussed in Section 2.2 with an emphasis on a perspective that will carry over to forming interaction graphs.

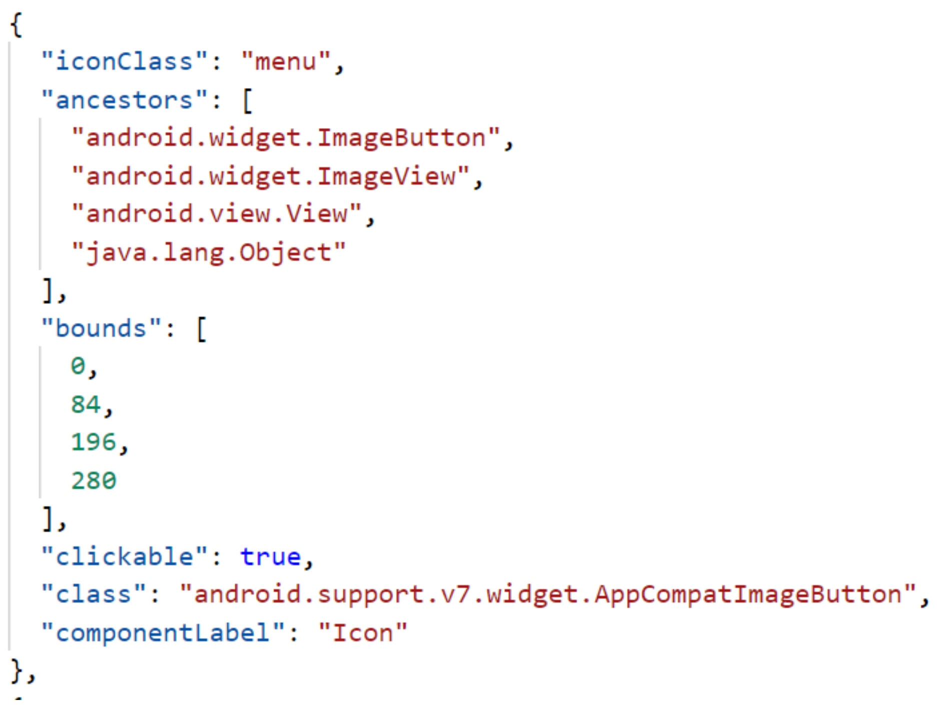
Thirdly, we discuss task analysis in Section 2.3 with a computational interaction focused perspective. We first introduce hierarchical task analysis to connect the concept of tasks in this work to previous work in the field of HCI. Then we review task complexity analysis followed by discussion on task complexity factors. Lastly, we introduce Wood's task complexity and several task complexity measures derived from software complexity measures.

2.1 Rico and Enrico

Rico [11] is a mobile UI dataset consisting of 72k UIs collected from 9.7k Android applications using a hybrid data collection strategy. The data includes screenshots, wireframes, view hierarchies, interaction traces, layout vectors, animations and metadata. The data has been collected by first employing a human to browse the application with the browsing data collected from their use sessions. The human browsing data has been used to warm up an automated crawler which performs further crawling.

Deka et al. [11] used a subset of applications to demonstrate that the hybrid crawling strategy achieves a better coverage of the applications. This extended coverage of applications with the dataset would make a useful feature for studying the total task complexities of real applications using the mathematical tools described in Section 2.3.2. However, the Rico dataset is noisy with mismatches between wireframes and screenshots and other data quality issues as described by [9, 12, 13, 14]. This means that any full application covering task complexity graph is also likely to not represent ground truth due to the noise present in the data. From the potential datasets we use Enrico [9] although the approach and dataset presented in [12] would offer a promising direction for studying task complexity as well.

Enrico [9] is a manually curated subset of Rico that contains high-quality data. Enrico was generated from a random sample of 10k UIs from the Rico dataset with 1460 UIs as the end result. The dataset curation was accomplished by having two humans review the UIs for inconsistencies between the wireframes and screenshots on a web-based graphical UI (GUI). Subsequently, the good UIs were labelled to a series of 20 UI layout topics such as login, maps and tutorial. The dataset quality was showcased by performing autoencoder classification on the dataset in order to predict the topic labels. Additionally, an UMAP algorithm based manifold projection was used to visualize the dataset. From the perspective of our work, the curation of



```
{
  "iconClass": "menu",
  "ancestors": [
    "android.widget.ImageButton",
    "android.widget.ImageView",
    "android.view.View",
    "java.lang.Object"
  ],
  "bounds": [
    0,
    84,
    196,
    280
  ],
  "clickable": true,
  "class": "android.support.v7.widget.AppCompatImageButton",
  "componentLabel": "Icon"
},
```

Figure 1: An Icon component view hierarchy description for Rico ID 124.

the data is the most critical feature, as this allows us to generate interaction graphs of the Enrico UIs in Section 4.

In Figure 1 we see an example of Rico ID 124 view hierarchy description for an Icon component. In this work, we will make use of the view hierarchy descriptions to infer task structures. The keys in the view hierarchy we will be leveraging include bounds, clickable and componentLabel. The clickable key will be particularly important as it will be used to determine which components form a part of the task structure used to measure task complexity. The componentLabel key will be used to determine which interaction type will be performed by the user, which we will discuss in the next section.

2.2 Graphs and Directed Graphs

A graph is a combinatorial structure that is formed by combining two distinct sets called vertices and edges with an incidence relation set between them. Additionally, graphs can have various attributes encoded into them with the use of colors, weights and descriptions. A directed graph is a particular type of graph where the incidence relations have arrows that point the direction of travel between the vertices. [15]

Figure 2 showcases two graphs. Both of the graphs have the same sets for vertices $\mathrm{V} = \{a, b, c\}$ but the edges differ. The left graph has edges defined by $\mathrm{E} = \{ab, ac, bc\}$ and the right graph by $\mathrm{E} = \{d, e, f\}$. The difference between these two graphs is that the one on the right is a directed graph and uses specifically assigned names for distinguishing the edges of the graph. The distinct names and directions of the edges on the right side graph form an important part of this work.

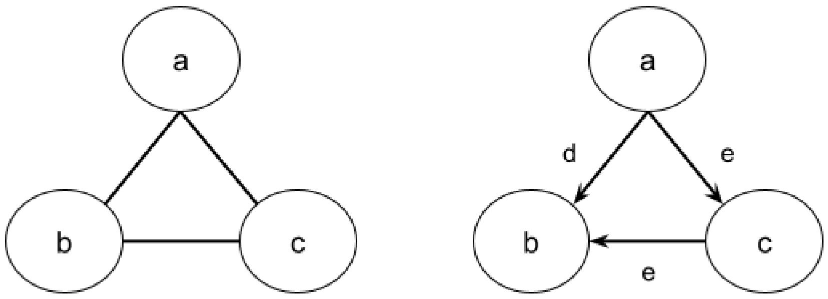
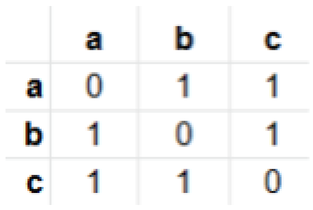
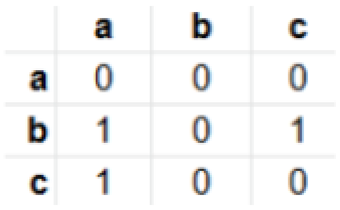


Figure 2: A graph with three nodes and vertices (left) and a graph with the same structure but specifically designated edge names in a directed graph format (right).



	a	b	c
a	0	1	1
b	1	0	1
c	1	1	0

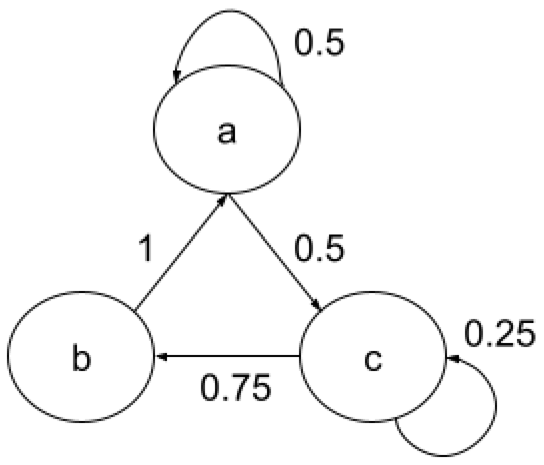


	a	b	c
a	0	0	0
b	1	0	1
c	1	0	0

Figure 3: Adjacency matrices for the graphs presented in Figure 2. The one on the left corresponds to the left graph in Figure 2 and the one on the right to the right graph. Note that the left matrix is symmetric due to two-way directions between the graphs. The right adjacency matrix is not symmetric, as the connections are one-way only in the graph the adjacency matrix represents.

From the perspective of our work, another particularly important concept is the matrix representation of graphs. A matrix representation of a graph allows us to employ linear algebra on graphs [15]. An adjacency matrix is a matrix where the rows and columns describe nodes of a graph. For a simple graph such as the one on the left of Figure 2 the adjacency matrix is a symmetric matrix with ones as values where there are connections between the nodes. The left side of Figure 3 shows an adjacency matrix for the undirected graph from Figure 2. The right side of the same figure shows the adjacency matrix for the right graph from Figure 2, where due to the directed edges the adjacency matrix is not symmetric anymore.

Graph based models can be used for various applications related to human interactions with computers, systems and human factors [16, 17, 18, 19, 20, 21, 22, 23]. For example, sequences of user interactions can be presented as paths in a graph where the user moves from one node to the next [17]. Similarly, user action choices can be modeled by transition matrices [17]. A transition matrix represents the probabilities of moving from one state to the next in a Markov chain. A Markov chain is a sequence where transitioning from one state to the next does not depend on earlier states in the sequence [15]. Figure 4 shows a transition matrix and a graph for the transition matrix for a three node scenario. Baber and Stanton [24] demonstrate how to use a transition matrix when performing task analysis for error



	a	b	c
a	0.5	0	0.5
b	1	0	0
c	0	0.75	0.25

Figure 4: A Markov diagram and a transition matrix, where moving from one state to the next depends on the probability attributes attached to the edges of the graph.

identification. In this work, we focus on non-stochastic behavior, but adjacency matrices can be replaced with transition matrices when studying stochastic user behavior with interfaces.

Hardman et al. [19] applied graph descriptions to mathematically study improving user interfaces for fighter jets such as the F-15. In the work, the display layout is modeled as a discrete Markov chain containing nodes and edges. The nodes describe a page on the display, and edges describe inputs performed by the user to move from one node to the next. In this context the directed graph is also connected as the user must be able to transition from any operational state to any other state given changing operation conditions. A graph is connected when any node is reachable from any other node in the graph [15]. The act of traversing between nodes is called a walk, and during a walk an alternating sequence of nodes and edges will be traversed through [15]. The work of Hardman et al. [19] differs on a fundamental level from our work, since the graphs generated in this work are not connected.

Ceci and Lanotte [25] applied sequential pattern mining to extract sitemaps, which help understand the design structure of websites without needing to manually maintain a sitemap. Similarly, hierarchical models can be used to model web graphs [26]. Hierarchical Markov models which are probabilistic graphs can be used to analyze human behaviors. Youngblood and Cook [27] used sequential pattern mining to automatically generate hierarchical Markov models of inhabitant behaviors in smart office and home environments. The hierarchical Markov models enabled learning how to control the smart environment using reinforcement learning.

2.3 Task Analysis

Task analysis is used to systematically obtain an understanding of human interaction with a system or between humans [28, 29]. Due to the recent developments in artificial intelligence [30, 31], and especially reinforcement learning [32], we will expand task analysis in the scope of this work to include agent interaction with a system or humans. We will call humans and agents as operators. Task analysis covers what actions and cognitive processes an operator needs to perform in order to achieve a goal [29]. A simple goal can be making the bed in the morning and moving a pillow to the end of the bed, an action that takes the operator towards the goal. Performing task analysis gives an understanding of the relations between the demands of the system, the capabilities of the operator in the system and how those demands might need to be altered to reduce errors in performing the task [29].

According to Wickens et al. [28], task analysis at a basic level consists of four steps which include

1. Defining the analysis purpose and identify the type of data required
2. Collecting task data
3. Summarizing task data
4. Analyzing task data

Performing task analysis can be accomplished by various techniques and methods, including hierarchical task analysis (HTA), [29], GOMS [33], task complexity analysis [34, 35, 29, 33] and cognitive task analysis [36]. Due to the vast number of task analysis methods and application domains [29, 33, 37] we will limit ourselves to briefly reviewing methods to those that are relevant to computational interaction and specifically to interface design and analysis. This means we are primarily concerned with task analysis where it fulfills one of the following criteria defining computational interaction [7]

- a way of updating a model with observed data from users;
- an algorithmic element that, using a model, can directly synthesize or adapt the design
- a way of automating and instrumenting the modelling and design process
- the ability to simulate or synthesize elements of the expected user-system behavior

With the above-mentioned criteria for computational interaction and steps for performing task analysis, examples of computational approaches to accomplishing task analysis or parts of it in different fields include

- Automated planning [30]
- Data mining [27, 38]
- Graph modeling [35, 39, 27]
- Machine learning [40, 41, 42]

- Reinforcement learning [43, 32, 44]
- Robotics [45, 46]
- Simulations [47, 48, 49]

Within the scope of this work we are focused on human interaction with a system and even more specifically on human interaction with a user interface. For interfaces, hierarchical task analysis is a recommended task analysis method [29] and additionally it has relevant similarities to hierarchical reinforcement learning [32]. Hierarchical task models can be formulated as matrices and tensors, connecting it to graph based task representations. As a consequence, we will describe hierarchical task analysis in Section 2.3.1 as an example. We will then move on to describing task complexity analysis in Sections 2.3.2, 2.3.4 and 2.3.5. A particular area of focus in this work is applying task complexity formulas.

Table 1 presents the task complexity equations that will be evaluated in the upcoming Sections 2.3.4 and 2.3.5. The formulas have been selected as a representative sample of formulas that have been considered for human-computer (or system) interaction previously [35, 39, 50, 51, 52, 53]. This choice is made in the light of not intending to make the primary focus of this work giving comprehensive overview of task complexity formulas. Examples of relevant formulas beyond the scope of this work include those in the finite-state machine literature [54, 55].

Equation	**Name**	**Formula**
Equation 1	Component complexity	$TC_1 = \sum_{j=0}^{p} \sum_{i=1}^{n} W_{ij}$
Equation 2	Coordinative complexity	$TC_2 = \sum_{i=1}^{n} r_i.$
Equation 3	Dynamic complexity	$TC_3 = \sum_{f=1}^{m} \lvert TC_{1(f+1)} - TC_{1(f)} \rvert$ $+ \lvert TC_{2(f+1)} - TC_{2(f)} \rvert$
Equation 4	Autocorrelation	$TC_4 = \sum_{f=1}^{m} \lvert TC_{1(f+1)} - TC_{1(f)} \rvert$ $(1 - \rho_{TC_1}) + \lvert TC_{2(f+1)} - TC_{2(f)} \rvert (1 - \rho_{TC_2})$
Equation 5	Total task complexity	$TC_t = \alpha TC_1 + \beta TC_2 + \gamma TC_3$
Equation 6	Halstead's E measure	$E = \frac{\eta_1 N_2 (N_1 + N_2)}{2 \eta_2} \log_2(\eta_1 + \eta_2)$
Equation 7	McCabe's v(G)	$v(G) = e - n + p$
Equation 8	Shannon's entropy	$H = -\sum_{i=1}^{h} p(A_i) \log_2 p(A_i)$

Table 1: Task complexity formulas evaluated in this work.

2.3.1 Hierarchical Task Analysis

HTA is a process where tasks to accomplish goals are decomposed down into subtasks until a desired level of detail is reached [56, 29]. According to Annett [56] when performing hierarchical task analysis, the unit of analysis is an operation that specified by goal. The operations are activated by actions caused by an input. Finally, the operations are terminated by feedback. Within the scope of HTA [29] goals are defined as desired system states by the human and tasks are specific methods used to obtain the goal. Operations are units of behavior that are performed to obtain a goal.

Hierarchical task analysis process begins by defining a goal the user has to accomplish [29]. The goals are decomposed down to sub-operations and plans, where the plans are used to describe in which order and under which conditions each sub-operations should be performed. The sub-operations may be broken down to even more detailed sub-operations if deemed necessary. This process is repeated until a sufficient level of detail is reached.

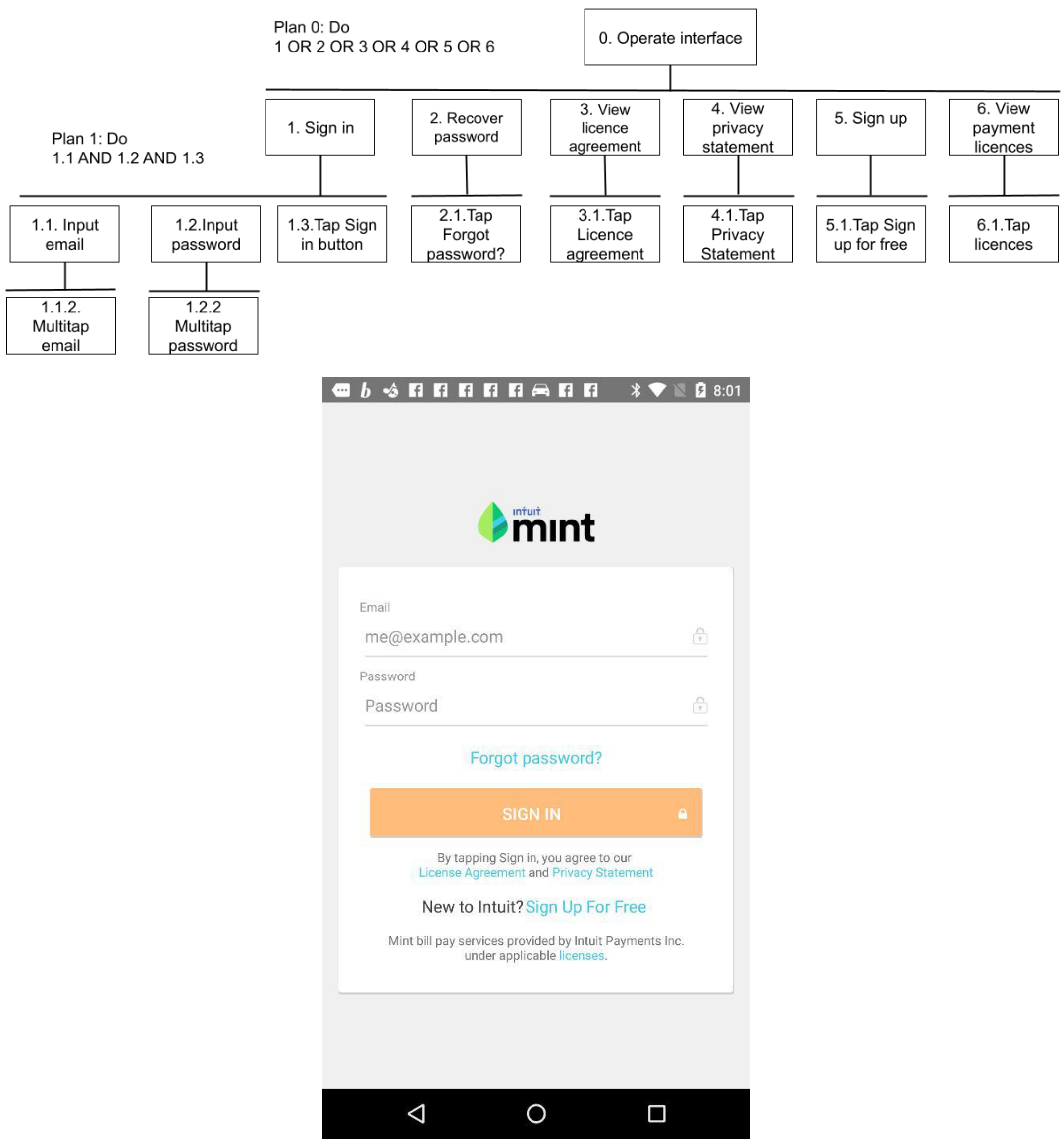
In Figure 5 contains a sample HTA for a user interface. The top level goal of the HTA is to simply operate the interface. For this goal, six possible tasks have been found. The six subgoals are on the second level of the hierarchy, such as signing in to the application. In order to accomplish the top level goal of operating the interface, the user needs to perform only one of the subgoals. This is denoted in the plan 0 where the user has to do one of the tasks as noted by the OR operators.

If the user chooses to do task 1 there will be three subtasks that need to be performed. The subtasks and their order are described in plan 1 where the AND operators define that they will all be executed to fulfill task 1. For inputting email and password (1.1.1 and 1.2.1) the user must perform subtasks for both which include multiple taps to activate the email or password fields and then fill in the correct sign in information. This is where in the context of our work we draw the line for sufficient detail of hierarchical task analysis and will not break it down any further.

2.3.2 Task Complexity Analysis

Task complexity analysis is a subtopic of task analysis [29, 33]. Task complexity simply represents the inherent difficulty with executing a particular task [33]. More specifically, task complexity refers to how complicated performing a task in terms of operations is for a human. This can include both objective and subjective task complexity [57]. Subjective task complexity is the perceived complexity observed by the task performers. On the other hand, objective task complexity is focused on measuring the intrinsic factors affecting the complexity of the task. An intrinsic attribute can be a number of steps needed to take or time taken to point to a specific element on the interface. We specifically use the definition provided by Liu and Li [57] also referenced by [58], where they defined task complexity as an aggregate of all intrinsic factors which have an effect on performing a task.

When considering different forms of task complexity, Ham, Park and Jung [59] visualized the forms of task complexity as presented in Figure 6 below. They distinguished the importance of user interface design as a mediator of task complexity. What their distinction implies is that by analyzing the representation of the work domain, we can improve human performance with the task. As both the interface and the work domain are subjects of objective complexity, we can apply optimization techniques on mathematical models of objective task complexity to study how to



By tapping Sign in, you agree to our
License Agreement and Privacy Statement

New to Intuit? Sign Up For Free

Mint bill pay services provided by Intuit Payments Inc.
under applicable licenses.

Figure 5: Hierarchical task analysis and the UI screenshot for Rico ID 2716. The purpose of this task analysis is to determine what options the user has in operating the interface.

improve human performance with the task. The improvement in human performance can be measured by reduced error rate, improved task completion time or in reduced subjective task complexity experienced by the human [57].

2.3.3 Task Complexity Factors

When considering task complexity factors suitable for objective perspective, the factors are preferentially measurable such that their measure is not dependent on the qualities of any particular user [57]. Additionally, for user interface derived measures, the factors are derivable from the visible representation of the user interface. Liu

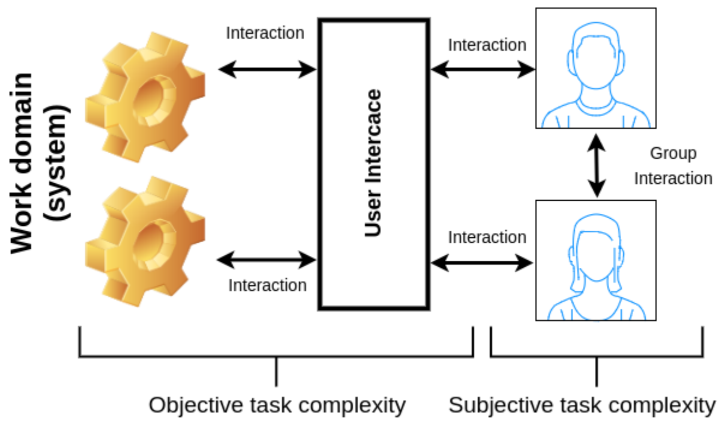


Figure 6: Task complexity types visualization based on the work of Ham, Park and Jung [59].

and Li [57] performed a review of factors contributing to task complexity, either as complexity reducing or complexity increasing factors. Table 2 compiles their review of the factors. The categories in Figure 2 serve as broader conceptual frameworks for grouping factors into related task components. The category of goal and output contains factors that are formed by the desired outcome of the task. The goal and output is necessary for task modeling in order to define when the task is successfully completed or when the user has failed the task. As a comparison of task clarity describing goal of a task as tapping on the purple button in Figure 8 leftmost screen is a task with a good task clarity. On the other hand, asking the user to identify a desirable product on the middle screenshot of Figure 8 is an example of a task that has low clarity due to ambiguity in what is considered a desirable product. A task that has little clarity is also difficult to represent objectively as a number for when it is successfully completed or not completed.

Input factors category represents the clarity and difficulty of giving input through the user interface [57]. As an example, if the text *"Henkilötunnus"* (social security number) was removed from figure 9 screenshots, then the input clarity would be reduced. A skilled user may still be able to figure out the required data for the input field through testing against the input validity validator used on the input field.

Process factors represent procedural operations that must be completed to get to the goal of the task. Quantity of steps represents an intrinsic measure of the number of procedures performed in the task. For completing a subtask represented by the three screenshots in Figure 8, the user has to complete three steps where each step is complete once the purple button has been tapped on in each of the views, for example. On the other hand, physical requirements of the process category in Table 2 are affected when the accuracy requirement of the physical action is increased, for example. This effect can be objectively measured by the index of difficulty of Fitts' law [60]. In the first screenshot of Figure 8 pressing the button with text *"Ohita edut"* places greater physical requirements as the target is smaller than the purple button above it.

Category	Factor	Effect
Goal/output	Clarity	Negative
	Quantity	Positive
	Redundancy	Negative
	Change	Positive
Input	Clarity	Negative
	Quantity	U-shape
	Diversity	Positive
	Inaccuracy	Positive
	Rate of change	Positive
	Redundancy	Negative
	Conflict	Positive
	Unstructured guidance	Positive
	Mismatch	Negative
	Non-routine events	Positive
Process	Clarity	Negative
	Quantity of paths	Positive
	Quantity of steps	Positive
	Conflict	Positive
	Repetitiveness	Negative
	Cognitive requirements	Positive
	Physical requirements	Positive
Time	Concurrency	Positive
	Pressure	Positive
Presentation	Format	Varies
	Heterogeneity	Positive
	Compatibility	Negative

Table 2: Compilation of task complexity factors by Liu and Li [57].

Time factors place temporal demands on completing and performing the task. For example, when performing emergency operating procedures on nuclear power plants, there may be limits on when the procedure must be completed in order to prevent safety hazards [35]. Similarly, when playing various computer or mobile games, there may be time limits which make performing the task presented by the game more difficult. Time factor is measurable as is, since having to perform the same task in less time adds time pressure factor to the task, measured by the difference between the original time requirement and the new time requirement.

2.3.4 Wood's Task Complexity

Wood [34] discussed three different perspectives to assess task complexity, defined as 1) component complexity, 2) coordinative complexity, and 3) dynamic complexity. Additionally, a way to assess total task complexity was presented by Wood. Component complexity refers to the number of information cues for distinct acts that

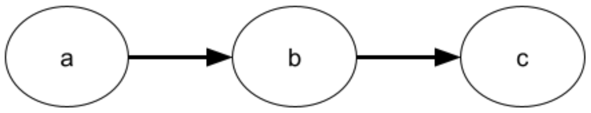


Figure 7: Precedence relations on a graph. Node b has a precedence relation to node a.

needed to be completed in order to complete a task. Component complexity can then be measured as

$$TC_1 = \sum_{j=0}^{p} \sum_{i=1}^{n} W_{ij}, \tag{1}$$

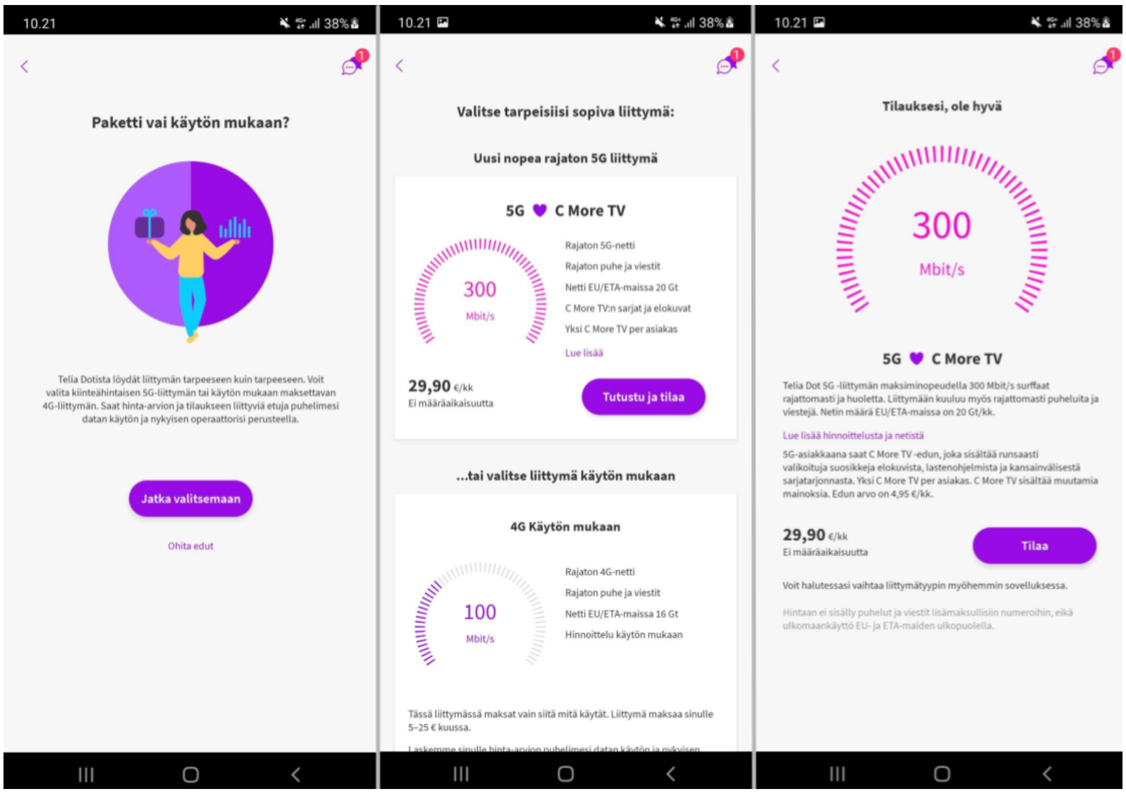
where p is the number of subtasks, n is the number of acts for each subtask and W_{ij} represents the number of information cues for each act. Wood noted redundancy as a factor that reduces component complexity, where redundancy can be analyzed from either acts or information cues perspective. An example is presented in Figure 8 of information cues and acts for that are both similar through button design across various parts of a UI. Similar design for UIs helps reduce total task complexity.

Coordinative complexity [34] refers to the demands placed on completing a task when considering the coordinative relationships between information cues, acts and products. Coordinative demands can be placed in the forms of timing, frequency, intensity and location requirements. An example of a coordiantive relationship is the requirement to move a mouse on top of a correct icon on a computer before clicking on it. A timing related coordinative relationship can be moving a game character while performing an act like collecting a moving item on the screen by pressing a collect item button. Generally, the more demands on a task are place on it in the forms of frequency, timing, intensity and location related requirements, the more coordinatively complex the task is.

Mathematically a simple case of coordinative complexity can be defined as a sequence of precedence relations in order to accomplish a particular task [34]

$$TC_2 = \sum_{i=1}^{n} r_i. \tag{2}$$

In Equation 2, TC_2 represents coordinative complexity, r_i denotes a precedence relation and n is the count of precedence relations i is the order of the precedence relation. In terms of a graph, the number of precedence relations is the number of edges when walking the start node to the current node on that specific graph path. Figure 7 showcases an example of a graph with precedence relations. Node b on the graph has one precedence relation, as the act represented by node a must be performed first. On the other hand, node c has two precedence relations, as acts on both a and b nodes on the graph must be performed first. The total TC_2 on Figure 7 is three, as both the precedence relations of nodes b and c are counted.



Paketti vai käytön mukaan?

Telia Dotista löydät liittymän tarpeeseen kuin tarpeeseen. Voit valita kiinteähintaisen 5G-liittymän tai käytön mukaan maksettavan 4G-liittymän. Saat hinta-arvion ja tilaukseen liittyviä etuja puhelimesi datan käytön ja nykyisen operaattorisi perusteella.

Valitse tarpeisiisi sopiva liittymä:

Uusi nopea rajaton 5G liittymä

5G 💜 C More TV

Rajaton 5G-netti
Rajaton puhe ja viestit
Netti EU/ETA-maissa 20 Gt
C More TV:n sarjat ja elokuvat
Yksi C More TV per asiakas

29,90 €/kk
Ei määräaikaisuutta

...tai valitse liittymä käytön mukaan

4G Käytön mukaan

Rajaton 4G-netti
Rajaton puhe ja viestit
Netti EU/ETA-maissa 16 Gt
Hinnoittelu käytön mukaan

Tässä liittymässä maksat vain siitä mitä käytät. Liittymä maksaa sinulle 5–25 € kuussa.

Laskemme sinulle hinta-arvion puhelimesi datan käytön ja nykyisen

Tilauksesi, ole hyvä

5G 💜 C More TV

Telia Dot 5G -liittymän maksiminopeudella 300 Mbit/s surffaat rajattomasti ja huoletta. Liittymään kuuluu myös rajattomasti puheluita ja viestejä. Netin määrä EU/ETA-maissa on 20 Gt/kk.

5G-asiakkaana saat C More TV -edun, joka sisältää runsaasti valikoituja suosikkeja elokuvista, lastenohjelmista ja kansainvälisestä sarjatarjonnasta. Yksi C More TV per asiakas. C More TV sisältää muutamia mainoksia. Edun arvo on 4,95 €/kk.

29,90 €/kk
Ei määräaikaisuutta

Voit halutessasi vaihtaa liittymätyypin myöhemmin sovelluksessa.

Hintaan ei sisälly puhelut ja viestit lisämaksullisiin numeroihin, eikä ulkomaankäyttö EU- ja ETA-maiden ulkopuolella.

Figure 8: Example of UI similarity (redundancy) in the form of purple buttons that acts as similar information cues and act requirements for moving to the next part across multiple views. Screen capture taken from Telia Dot application [61].

Dynamic task complexity [34] refers to changes in the complexity of a task over time, either due to predictable or stochastic processes. The changes in dynamic complexity are measured by

$$TC_3 = \sum_{f=1}^{m} |TC_{1(f+1)} - TC_{1(f)}| + |TC_{2(f+1)} - TC_{2(f)}|, \tag{3}$$

where TC_1 represents component complexity from Equation 1 and TC_2 is a measure of coordinative complexity from Equation 2. Both TC_1 and TC_2 are measured in standardized units and f represents the number of time periods when measures are taken for the task. If the task complexity is evolving unpredictably, then an autocorrelation term may be required in order to represent the predictability of the change. Mathematically this is formulated as

$$TC_4 = \sum_{f=1}^{m} |TC_{1(f+1)} - TC_{1(f)}|(1 - \rho_{TC_1}) + |TC_{2(f+1)} - TC_{2(f)}|(1 - \rho_{TC_2}), \tag{4}$$

where the terms are the same as in Equation 3 but ρ_{TC_i} represents the autocorrelation for component and coordinative complexities. As autocorrelation measures similarity between two time series, then an increase in autocorrelation means that the dynamic task changes are more similar to each other, leading to a reduction in dynamic task complexity metric.

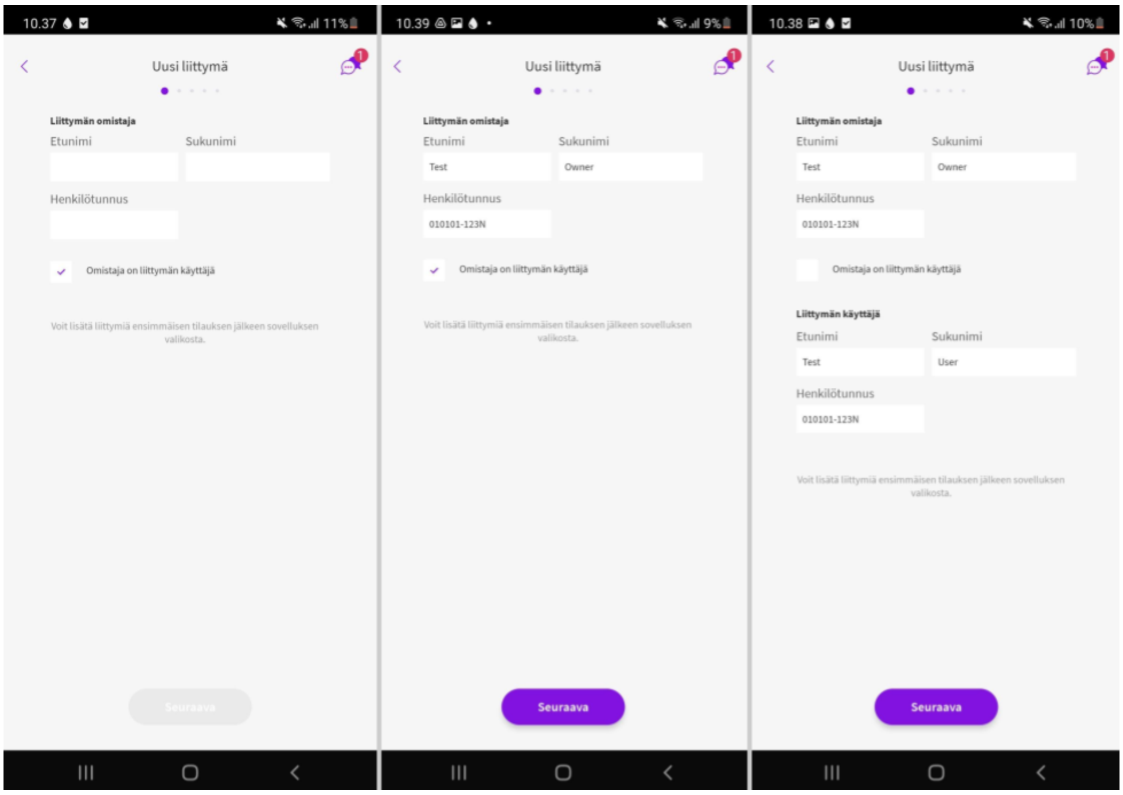
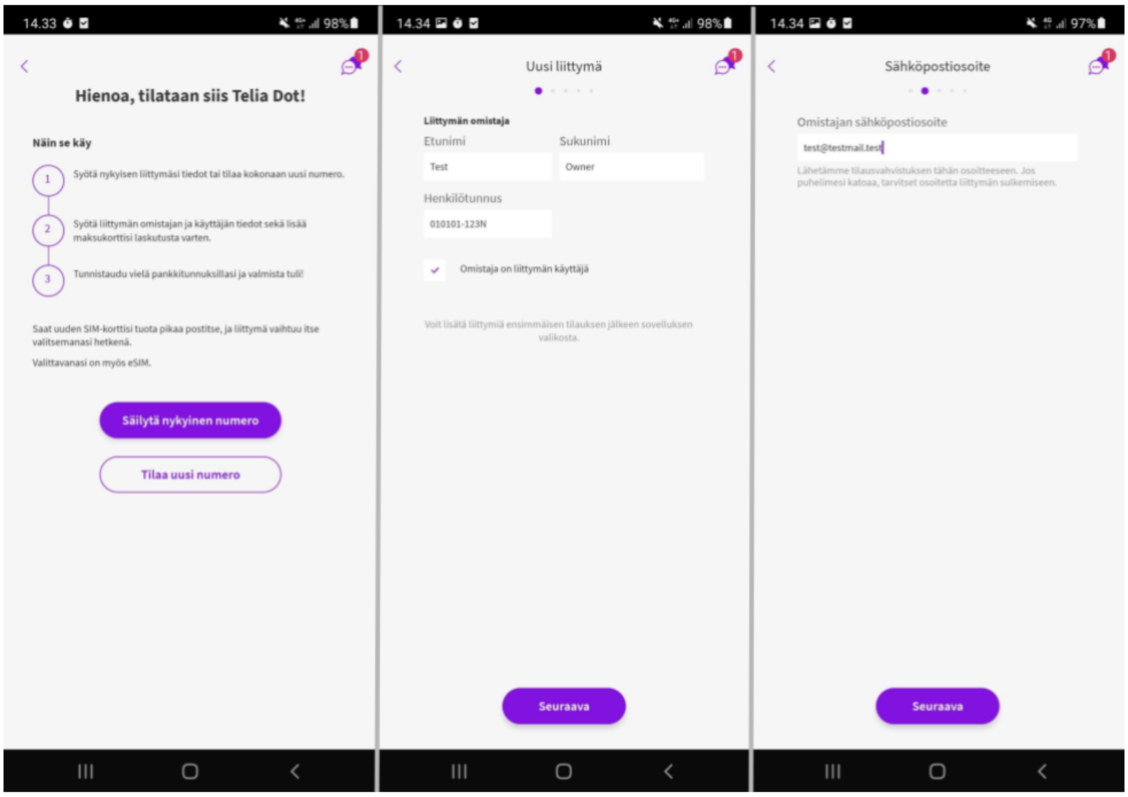

10.37
Uusi liittymä
Liittymän omistaja
Etunimi
Sukunimi
Henkilötunnus
✓ Omistaja on liittymän käyttäjä
Voit lisätä liittymiä ensimmäisen tilauksen jälkeen sovelluksen valikosta.
Seuraava
10.39
Uusi liittymä
Liittymän omistaja
Etunimi: Test
Sukunimi: Owner
Henkilötunnus: 010101-123N
✓ Omistaja on liittymän käyttäjä
Voit lisätä liittymiä ensimmäisen tilauksen jälkeen sovelluksen valikosta.
Seuraava
10.38
Uusi liittymä
Liittymän omistaja
Etunimi: Test
Sukunimi: Owner
Henkilötunnus: 010101-123N
Omistaja on liittymän käyttäjä
Liittymän käyttäjä
Etunimi: Test
Sukunimi: User
Henkilötunnus: 010101-123N
Voit lisätä liittymiä ensimmäisen tilauksen jälkeen sovelluksen valikosta.
Seuraava

Figure 9: A UI view that places coordinative complexity requirements on the user. In order to get the move to next view button active represented by purple color, the user must first correctly fill in the required information. In this view, the required coordinative complexity is variant and depends on if the owner is also the user of the mobile subscription. Screen capture taken from Telia Dot application [61].

When the change in dynamic complexity happens through a predictable process, usually there are information cues and other knowledge that help indicate what will happen when the task complexity changes. Figures 10 and 11 below represent an example of predictable dynamic task complexity change depending on what type of number order a user makes. Figure 10 represents a situation where the customer initially chooses the white button in the leftmost screenshot for a new number. Figure 11 is the same situation, but the customer wants to transfer an existing phone number and has chosen to click on the purple button. Depending on which button the customer presses, the task complexity change is dynamic and reasonably predictable. Additionally, Figure 9 showcases an example of dynamic complexity between the center and right screenshots, where selecting if the owner is the user alters the task requirements in a fairly predictable manner. If the owner of the subscription is not the owner, then personal information for the different user must be provided.


14.33

Hienoa, tilataan siis Telia Dot!

Näin se käy

1 Syötä nykyisen liittymäsi tiedot tai tilaa kokonaan uusi numero.

2 Syötä liittymän omistajan ja käyttäjän tiedot sekä lisää maksukorttisi laskutusta varten.

3 Tunnistaudu vielä pankkitunnuksillasi ja valmista tuli!

Saat uuden SIM-korttisi tuota pikaa postitse, ja liittymä vaihtuu itse valitsemanasi hetkenä.

Valittavanasi on myös eSIM.

14.34

Uusi liittymä

Liittymän omistaja

Etunimi	Sukunimi
Test	Owner

Henkilötunnus: 010101-123N

✓ Omistaja on liittymän käyttäjä

Voit lisätä liittymiä ensimmäisen tilauksen jälkeen sovelluksen valikosta.

14.34

Sähköpostiosoite

Omistajan sähköpostiosoite: test@testmail.test

Lähetämme tilausvahvistuksen tähän osoitteeseen. Jos puhelimesi katoaa, tarvitset osoitetta liittymän sulkemiseen.

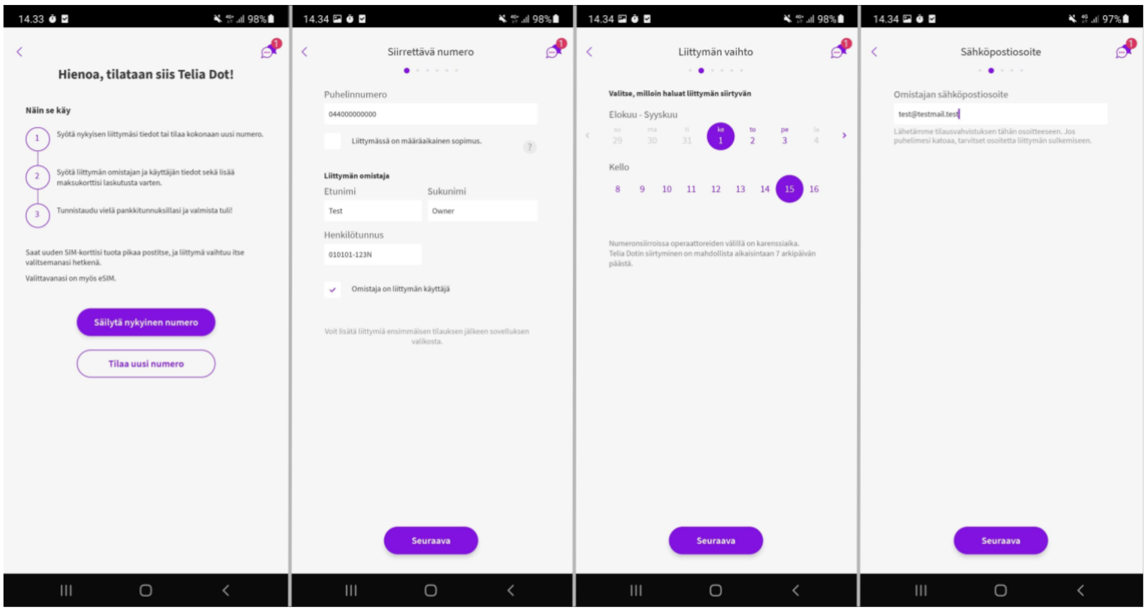
Figure 10: A scenario where the customer wants to order a new phone number, which leads to a predictable dynamic complexity change for the next view shown to the customer. The new number is represented by the white button in the left screenshot. Screen capture taken from Telia Dot application [61].

Lastly, for total task complexity, Wood [34] recommends a linear formula in the form

$$TC_t = \alpha TC_1 + \beta TC_2 + \gamma TC_3, \tag{5}$$

where each factor from equations 1 to 3 is weighted according to a scalar value with $\alpha > \beta > \gamma$. However, he notes that to get a more accurate formula derivation is obtained through derivation due to dependencies between equations 1, 2 and 3.

When considering total task complexity for a modern UI, then Figures 10 and 11 represent a case example where a measure of total task complexities could be obtained for two alternative UIs. The component and coordinative complexities are the same for the left and right figures. On the other hand, the effects of the center screens lead to different total task complexity measures for ordering a number transfer or a new number.



Hienoa, tilataan siis Telia Dot!

Näin se käy

1. Syötä nykyisen liittymäsi tiedot tai tilaa kokonaan uusi numero.
2. Syötä liittymän omistajan ja käyttäjän tiedot sekä lisää maksukorttisi laskutusta varten.
3. Tunnistaudu vielä pankkitunnuksillasi ja valmista tuli!

Saat uuden SIM-korttisi tuota pikaa postitse, ja liittymä vaihtuu itse valitsemanasi hetkenä.

Valittavanasi on myös eSIM.

Siirrettävä numero

Puhelinnumero: 044000000000

Liittymässä on määräaikainen sopimus.

Liittymän omistaja

Etunimi: Test

Sukunimi: Owner

Henkilötunnus: 010101-123N

✓ Omistaja on liittymän käyttäjä

Voit lisätä liittymiä ensimmäisen tilauksen jälkeen sovelluksen valikosta.

Liittymän vaihto

Valitse, milloin haluat liittymän siirtyvän

Elokuu - Syyskuu

Kello

Numeronsiirroissa operaattoreiden välillä on karenssiaika. Telia Dotin siirtyminen on mahdollista aikaisintaan 7 arkipäivän päästä.

Sähköpostiosoite

Omistajan sähköpostiosoite: test@testmail.test

Lähetämme tilausvahvistuksen tähän osoitteeseen. Jos puhelimesi katoaa, tarvitset osoitetta liittymän sulkemiseen.

Figure 11: The customer has chosen to press on the purple button in the left most figure as they want to transfer their existing phone number. This dynamically alters the task complexity and this time two new views must be successfully navigated through. After the two subtasks are completed, the customer returns to the same phase in the rightmost picture as in Figure 10 rightmost picture. Screen capture taken from Telia Dot application [61].

2.3.5 Software Complexity Formulas

The major difference with the equations used by Park et al. [35] when compared to the derivation of Equation 5 by Wood [34] comes from using the software complexity concept as a way to derive a step complexity measure. Software complexity can be analyzed from textual and structural perspectives. The textual perspective analyzed by Park [35] includes Halstead's E measure, while structural perspective was considered using McCabe's v(G) measure and entropy measure. Of the three formulas, Park found the entropy measure to be the most suitable for analyzing step complexity of nuclear power plant emergency operating procedures, where Xu's choice of formula [62] seems to be in agreement with this assessment.

Halstead's E measure is defined as

$$E = \frac{\eta_1 N_2 (N_1 + N_2)}{2\eta_2} \log_2(\eta_1 + \eta_2), \tag{6}$$

where η_1 is the number of unique operators, η_2 is the number of unique operands, N_1 the total frequency of operators and N_2 the total frequency of operands. Park [35] describes Halstead's E measure as an absolute measure of software complexity, where the number of mental discriminations is used to obtain a complexity measure through operands and operators. In software complexity the semicolon ; and IF clauses are traditional examples of operators while any variable A can act as an operand.

In Figure 11, the second screenshot from the left serves as an example of application of Halstead's E measure to user interface task complexity analysis. Activating the text entry fields and pressing on any of the buttons can be classified as operators, while the checkmarks for two fields and text entry fields contain operands. For the text entry field, the text value is the operand. The frequency of checkmark operands and operators is 2 for both, while the text entry field activators have 4 operators and 4 operands. Park [35] remarked that the primary challenge with applying Halstead's E measure to task complexity analysis is consistently using predefined operators and operands. In this work, the discriminating operands and operators is a suitable way to analyze task complexity as the operands and operators are predefined by Rico dataset [11].

McCabe's v(G) measure [63] uses a directed graph construction in order to define a measure of software complexity. The graph must have a unique start and end node, and each node corresponds to a block of code with sequential flow of code. The v(G) measure is also known as the cyclomatic number and is defined as

$$v(G) = e - n + p, \tag{7}$$

where e is the number of nodes, n is the number of vertices connecting the nodes and p is the number of connected components. For studying the applicability of v(G) to interface task complexity, we also need the definition of a strongly connected graph [63]. A strongly connected graph has nodes connected by edges such that each node can be reached from any node by traversing along the edges. As we will see in Section 3 with the method we have constructed interaction flow graphs McCabe's v(G) will not be a suitable task complexity metric.

Lastly, for the entropy based formulation from [64] we define entropy based on Shannon's entropy definition such that

$$H = -\sum_{i=1}^{h} p(A_i) \log_2 p(A_i). \tag{8}$$

In Equation 8, A_i defines the identified classes in the graph, h is the number of identified classes and $p(A_i)$ is the probability of encountering a class A_i within the graph. When measuring complexity using entropy, there are two types of entropy, where the first-order entropy is called chromatic information content and the second-order entropy is called structural information content. Chromatic information content measures the amount of differently connected nodes in the graph. When defining each node as a mode of interaction, such as a tap or a swipe, we in essence measure the expected paths towards the next interaction point and where the previous interaction was most likely done. Second-order entropy on the other hand considers class identification by categorizing nodes to be the same if they have the same number and types of neighbors within one arc distance [35]. Although Park et al. chose to use this measure, it is likely not ideal for our work. This is due to the majority of graph nodes being connected in the same way. Additionally, the neighbors for almost all the nodes will be the same S and E nodes.

3 Interface Task Complexity Analysis

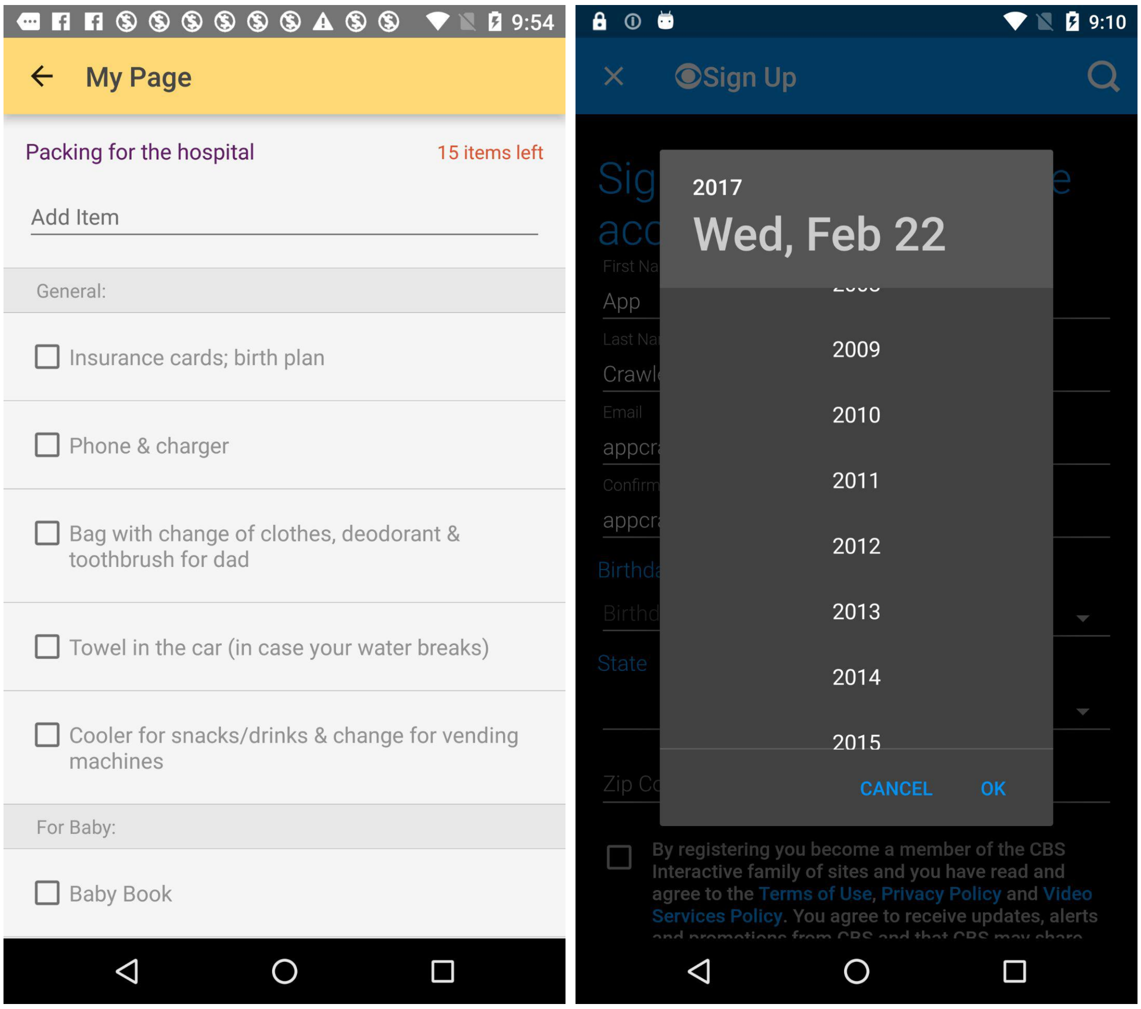
In this section, we develop an approach for extracting task structure from view hierarchies and perform computation procedure for two distinct task complexity measures for user interfaces. The two complexity measures we use are Halstead's E measure, and Wood's TC measure, with background on the measures provided in Section 2.3.4 for TC and Section 2.3.5 for Halstead's E measure.

In order to quantify task complexity for interfaces, we must first discuss the definition of available interaction modes for mobile devices and particularly their relation to different types of components occurring in Rico [11] and Enrico datasets [9]. Quantifying the interaction modes will allow us to formulate an interaction graph containing descriptions of relations between various interactions performed by the humans dependent on the interface structure. The graphs can then be reformulated as adjacency matrices, which lend themselves well for algorithmic use. Lastly, the combined information formed by the graph and adjacency matrix can be leveraged for computing task complexity measures.

3.1 Defining Interaction Modes

The two basic interaction mode definitions in this work are provided by the Rico dataset [11]. The two definitions are defined by the Android ecosystem [65]. The first interaction mode is a tap, where a user simply touches a component on the screen. The second interaction mode is a swipe, where a user will perform a horizontal swiping gesture with their finger once it comes in touch with the phone screen. With the nature of task complexity measurement, it is preferable to make two amendments to the interaction mode definitions. Some components in Rico and Enrico datasets such as Input component and Map View component can require the users to perform multiple interactions in a row.

Firstly, they may use multiple gestures to accomplish different goals on the same component. When taking the example of an Input field, a particular problem is the number of taps the user can perform to input their name or email or any other requested information. The tap count the user will perform is unknown, but we should be able to quantify the increased task difficulty caused by multiple taps. Two examples of this type of scenario are shown in Figure 12 where the user can Input a new item to the list or interact with a Date Picker component. The number of taps will be variant and depend on the task the user is intending to perform. So the task of inputting an item for "shoes" is less complicated than the task of inputting "Put insurance card in the wallet". As a solution to this problem, we define an interaction mode definition called multi-tap. The definition is valid for any component that can be tapped more than once during a task. This definition lends itself to Halstead's E measure, where a multi-tap adds to the number of unique operands as defined in Equation 6 while not modifying any other factor of the measure. The multi-tap definition is in essence a placeholder for an unknown variable that is not defined yet. If the number of interactions on the component is known, then the placeholder can be replaced with a series of taps on the component.


9:54

My Page

Packing for the hospital 15 items left

Add Item

General:

- [ ] Insurance cards; birth plan
- [ ] Phone & charger
- [ ] Bag with change of clothes, deodorant & toothbrush for dad
- [ ] Towel in the car (in case your water breaks)
- [ ] Cooler for snacks/drinks & change for vending machines

For Baby:

- [ ] Baby Book

9:10

Sign Up

Sig

acc

First Na

App

Last Na

Crawl

Email

appcr

Confirm

appcr

Birthd

Birthd

State

Zip Co

2017

Wed, Feb 22

2009

2010

2011

2012

2013

2014

2015

CANCEL OK

By registering you become a member of the CBS Interactive family of sites and you have read and agree to the Terms of Use, Privacy Policy and Video Services Policy. You agree to receive updates, alerts

Figure 12: Rico ID 1332 (right) with an Input component and, ID 19016 (left) with a Date Picker component. These two interfaces have the commonality that we don't know how many taps the user may choose to apply on the components.

In the second modification, we consider scenarios where the user may choose amongst multiple different interaction styles such as taps, swipes or pinches [65] to perform an operation on the same component. This scenario is particularly exemplified by the Map view component where the user may move the map with a swipe, tap on some part of the map or pinch to zoom in. An example interface is show in Figure 13. We call this interaction mode placeholder multimode and the purpose is to be replaced once the exact interaction choice is known, just like with multi-tap interaction placeholder.

As a final review, we now have four distinct interaction modes we are using, as opposed to the two modes originally used in Rico [11]. The modes are tap, swipe, multi-tap, and multimode. The multi-tap and multimode exist to serve as 1) placeholders for unknown interaction quantities, and 2) as distinguishers for Halstead's E measure. If the tasks the users perform were specified explicitly beforehand, we could omit the placeholder interaction modes. An alternative solution is to use the

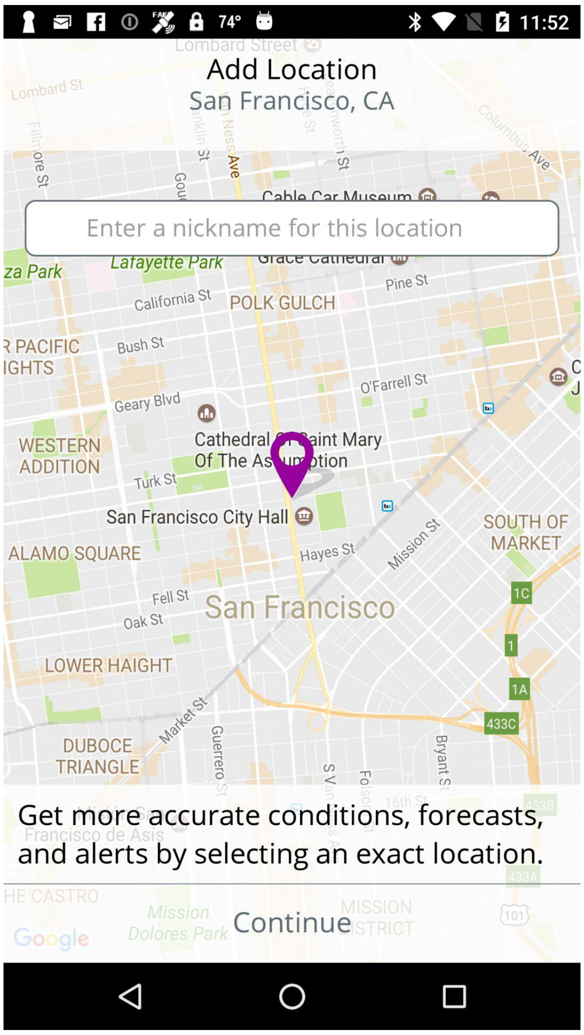

74°
11:52
Add Location
San Francisco, CA
Enter a nickname for this location
Get more accurate conditions, forecasts, and alerts by selecting an exact location.
Continue

Figure 13: Rico ID 4008 with a Map View component enabling the user to choose among several interaction modes such as a swipe or a pinch.

most common interaction mode for these components, which is a tap.

3.2 Mappings of Interaction Modes, Components and Task Endings

Enrico [9] dataset has 17 distinct components that are interactable as shown in Table 3. For each component we have designated an abbreviation purely for making vertex plotting more feasible for graphs with a large vertex count as performed in Section 4. The interaction modes for each component are defined under the interaction column in Table 3. For the vast majority of components, a tap [65] is the most suitable interaction mode. For three of the components we have the multi-tap and multimode interaction modes as described in Section 3.1 We cannot guarantee perfect accuracy for some components with respect to the correct interaction mode such as Image or Web View components. Ensuring correct interaction mode interpretation all the

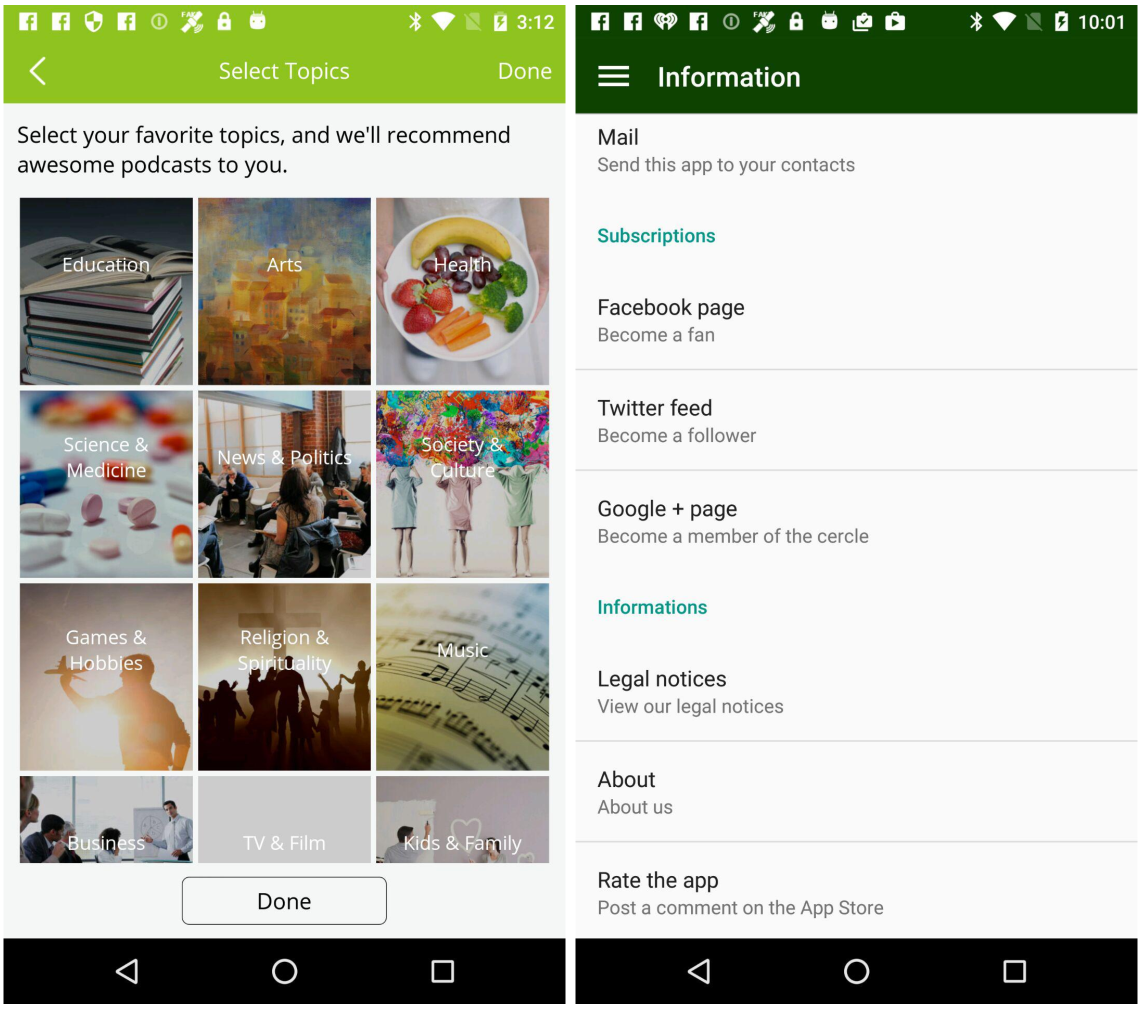
Component	**Abbreviation**	**Interaction**	**Ends task**
Advertisement	AD	Tap	yes
Background Image	BI	Tap	yes
Button Bar	BB	Tap	yes
Card	CC	Tap	yes
Checkbox	C	Tap	no
Date Picker	DP	Multi-tap	no
Icon	I	Tap	yes
Image	IM	Tap	yes
Input	IP	Multi-tap	no
List Item	L	Tap	yes
Map View	M	Multi-mode	yes
On/Off Switch	S	Tap	no
Pager Indicator	P	Swipe	yes
Radio Button	RB	Tap	yes
Text	TC	Tap	yes
Text Button	TB	Tap	yes
Web View	WV	Tap	yes

Table 3: Enrico component labels and their corresponding abbreviations, HCI interaction types and if the interaction will end the current task the user is performing. The definition of which interactions can end a task is dependent on the way task and goals are defined. In this work, the goal is to simply operate an interface until a new view is shown, so any action that likely moves the user to the next view is deemed as task ending interaction.

time would require manual human review or an advanced algorithm that is able to infer by itself the interaction mode for components based on the nature of the task to be performed.

The ends task column is used to define if interacting with the specific component is likely to end a task. If the task will not be ended it means we will connect two components to each other rather than to an end node which will be described in detail in 3.3. When considering Rico ID 505 in Figure 14 the UI is asking the user to select topics of interests and then click Done which would complete the task the user should perform on that interface. In this UI the Done button is in two places which makes algorithmic inference more difficult as we will see in Section 4.

On the contrary to Rico ID 505 in ID 545 in Figure 14 we see List Items that do not end the task. The task description for ID 545 is to select a particular item from the menu the user wishes to further interact with, such as opening About section. An alternative task is to open the top left menu icon component if they wish to navigate elsewhere on the application. This is another challenge we will face when performing algorithmic task inference in Section 4. Generally a designer may choose to use a component in a variety of ways and encoding all of these ways into an algorithm is challenging. This also forms one of the more prominent limitations of this work


3:12

Select Topics

Done

Select your favorite topics, and we'll recommend awesome podcasts to you.

10:01

Information

Mail
Send this app to your contacts

Subscriptions

Facebook page
Become a fan

Twitter feed
Become a follower

Google + page
Become a member of the cercle

Informations

Legal notices
View our legal notices

About
About us

Rate the app
Post a comment on the App Store

Figure 14: Rico IDs 505 (left) and 545 (right) showcasing UIs with List Item components. The UIs differ notably in the sense that the left one has not task ending List Items, while ID 545 has task ending List Items. This conflict is one of the fundamental challenges in developing algorithms to automatically infer task structures from preexisting UI view hierarchies.

in the algorithmic section. When a human does manual task analysis and graph constructions, inferring these differences is almost trivial on the other hand, as shown in Section 3.3.

3.3 Adjacency Matrix and Interaction Graphs

In order to measure task complexity on interfaces, we need a systematic way to extract the structure and order of interactions from the interface structure based on the description in the view hierarchy of the interface. Interaction graphs and adjacency matrix are based on directed graphs as described in Section 2.2. Even more specifically, the interaction graphs we construct are based on control graphs. Some relevant contexts where control graphs have been introduced include software complexity studies [63, 66], nuclear power plant EOPs [35], and spaceflight operation complexity [39]. As a consequence, our concern is on applying control graphs within the novel context of studying human interaction with mobile user interfaces. For brevity, we call the graphs in this work interaction graphs rather than interaction control graphs.

3.3.1 Graph Attributes

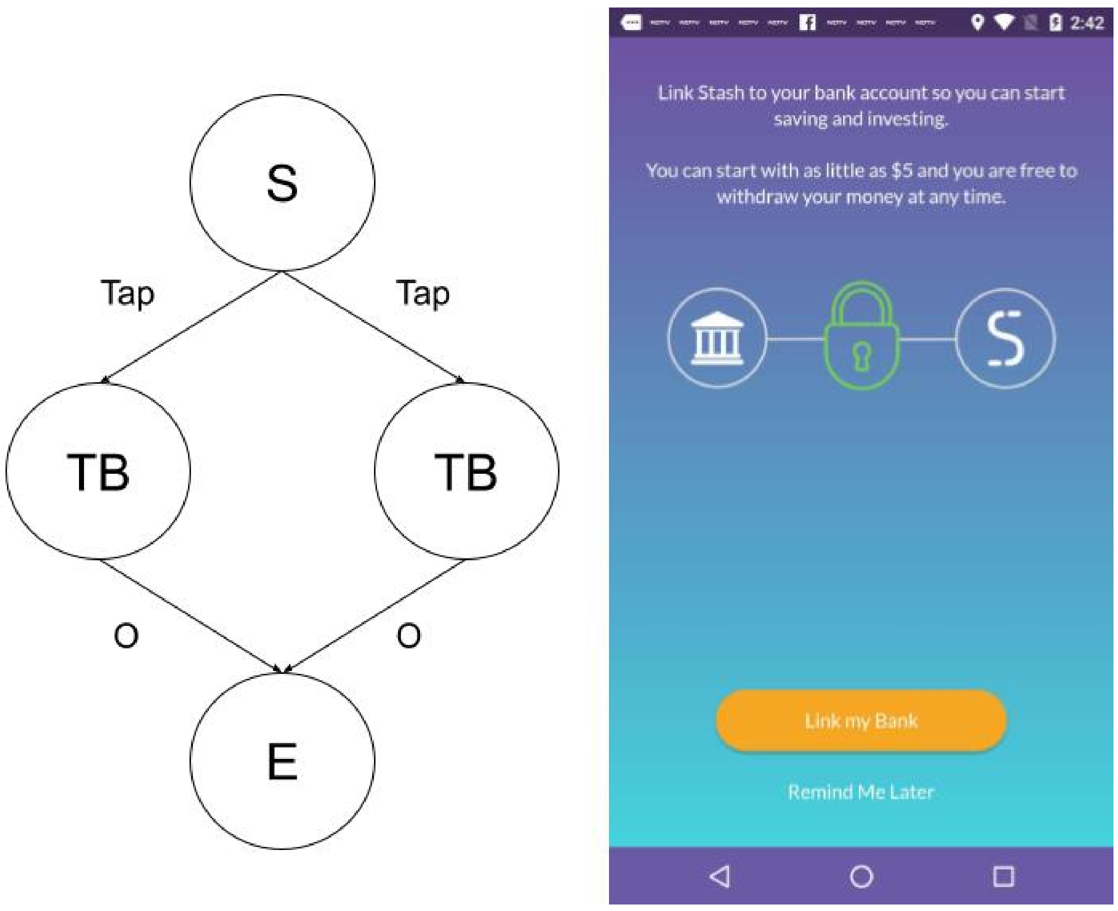
The process of building interaction graphs starts from the types of interface components. The component will determine the mode of interaction, so we first embed the component on the graph. The components are placed as the nodes of the graphs according to the encoding provided in Table 3. An example graph for Rico ID 2789 is shown in Figure 15. A component being defined as a node can be interpreted as something the arc of the graph will perform an operation on, such as a tap. So the nodes form the operands for Halstead's E measure. The arc of the graph has an interaction mode as an attribute, which is defined by the node the arc will attach to. The arc represents the operation a human must perform in the form of an interaction gesture.

When we consider the situation in Figure 15 the user is able to perform two different tasks that both require interaction. The form of interaction is a tap for both of the tasks, and they are both performed on text button (TB) components. When considering Figure 15 further, we can see that a graph is a suitable mathematical structure to store information about the fundamental aspects of multiple tasks at the same time. Thus, each of the arcs starting from the node marked with S then constitutes a distinct subtask within a collection of possible tasks making up the whole UI view.

The purpose of the S node in Figure 15 is to represent the start of the task, which also can be interpreted as the user observing the interface when such a distinction is beneficial. It also serves as a placeholder for constructing larger interaction graphs made up of multiple views with distinct components. Similarly, the E node represents the end of a task once a task ending interaction as defined in Table 3 has been performed.

When we have an interface with components that do not end a task within the particular scope of task analysis, we get interaction paths longer than one node for some of the paths that start from the S node. An example is presented in Figure 16 where the user may perform three distinct tasks starting from the S node. The first task on the left side is for tapping the top left corner icon (I) to move backwards to the previous screen. The second task is getting another confirmation code on the phone by tapping the *Resend Code* text button (TB). The last task on the right is the most complicated one, as the user must first input the received confirmation code on the input (IP) component with a series of taps. Once the user has inputted the code, they must tap the *Continue* text button (TB) connected to the IP component to finish the task located on the right side of the Figure 16.

The UI in Figure 16 I also show us an example of how we have limited the scope of this work. A generic task for the user logging in could require the user to resend code



Link Stash to your bank account so you can start saving and investing.

You can start with as little as $5 and you are free to withdraw your money at any time.

Figure 15: Rico ID 2789 interaction graph and the UI. In the graph, the S node refers to a start position or initial observation by the user. It is best thought of as an attachment point for all the tasks visible on the interface. The TB nodes represent the two text buttons on the interface, and on both of them the user can perform a tap gesture. The E node refers to the end point for all the tasks that can be performed on the interface. The O attributes of the arcs can be thought of as observations caused by the interface refreshing the view after interacting with a task ending component.

and after that perform input entry and tapping the *Continue button*. Constructing a graph for this particular task could be accomplished by simply attaching the right most path on Figure 16 directly after the TB node. We however will keep the scope of tasks as simple as possible during this work and focus on demonstrating application of graphs to user interface complexity computation.

3.3.2 Adjacency Matrix

Although we have already constructed interaction graphs by hand for user interfaces, it is not enough to algorithmically infer interaction graphs. Rather, we need to have a way to represent the component order and relations between them in a numeric format suitable for a computer. This is where the adjacency matrix comes in, as we

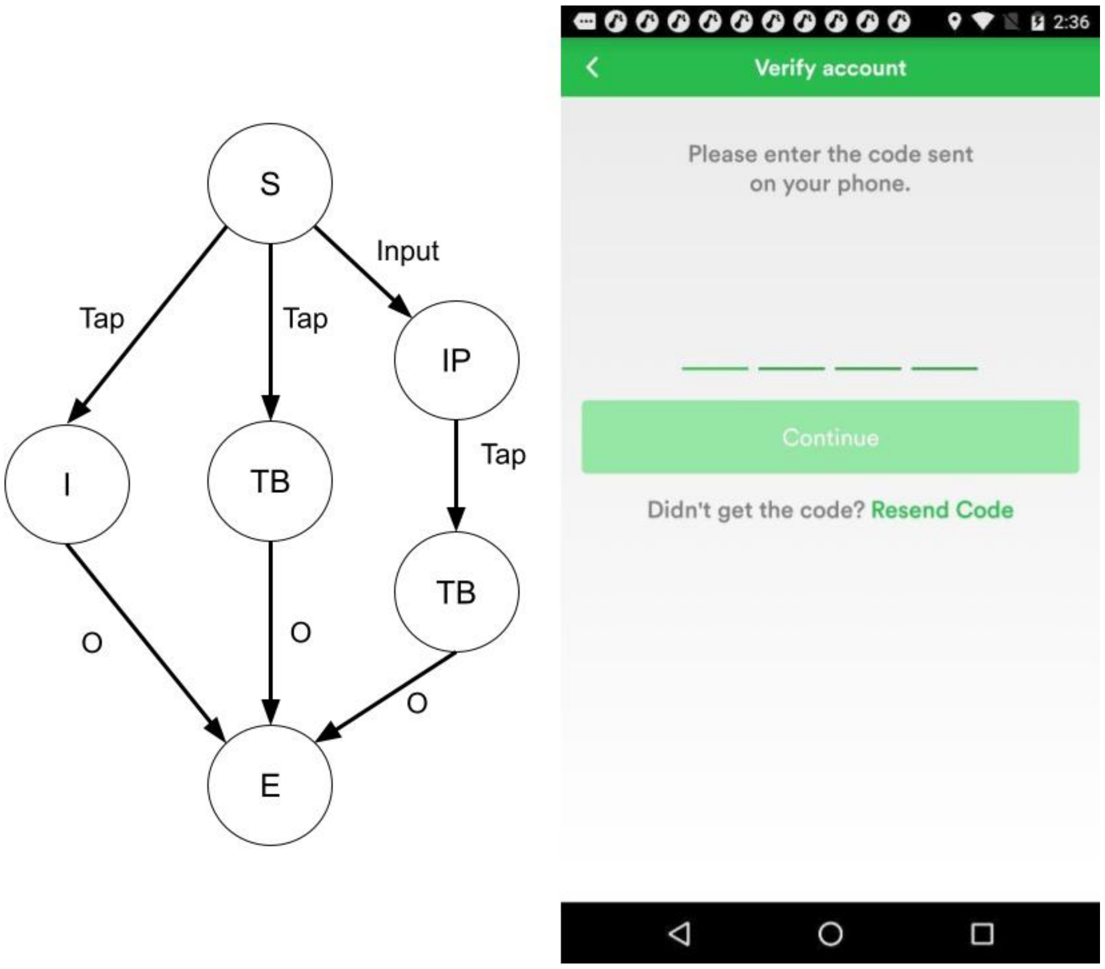

S
Tap
Tap
Input
IP
Tap
I
TB
TB
O
O
O
E
2:36
Verify account
Please enter the code sent on your phone.
Continue
Didn't get the code? Resend Code

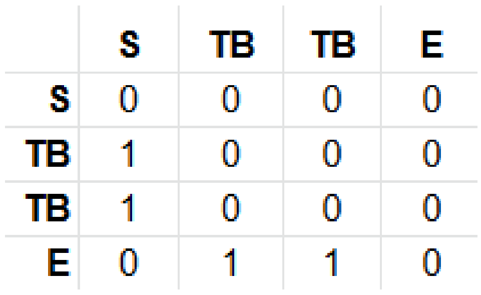
Figure 16: Rico ID 1074 with two types of traverse paths starting from the S node. The graph has two paths with traverse length one and one path with traverse length two. Each of the paths starting from the S node represents a distinct task.

will later see in Section 4. Here, we focus on describing a few simple cases that help understand how to read an adjacency matrix used to describe an interaction graph.

Based on what we saw in the previous Section 3.3.1 and based on the theory of directed graphs discussed in Section 2.2 we still need to fully describe the relations between nodes in a manner that is suitable for algorithmic inference of task structure from user interface hierarchies. In order to do this, we use adjacency matrices, where the columns and rows of the matrix describe the user interface components. In other words, each column and row position refers to a node on the interaction graph.

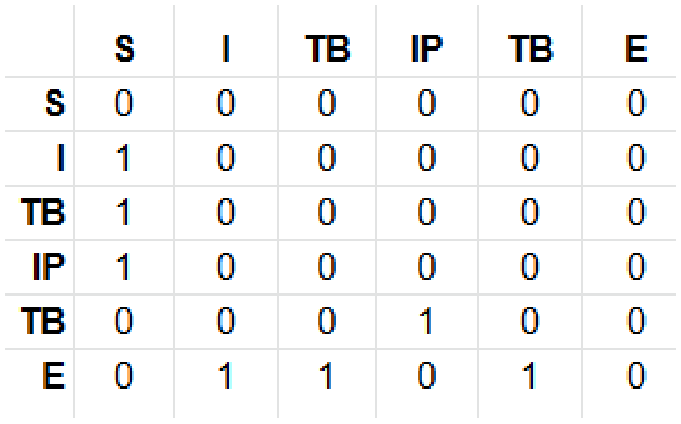
When we consider the graph and screenshot of Rico ID 2789 in Figure 15 we get an adjacency matrix as show in Figure 17. The matrix is interpreted such that the column nodes have arcs pointing towards the row nodes, where the values are set to one. Thus, the column with S has arcs pointing to both of the text button (TB) components. Similarly, both of the text buttons (TB) have arcs towards the end node E represented by the ones on the last row of the matrix.

A slightly more complicated case is presented by the interface shown in Figure 16



	S	TB	TB	E
S	0	0	0	0
TB	1	0	0	0
TB	1	0	0	0
E	0	1	1	0

Figure 17: Rico ID 2789 adjacency matrix.



	S	I	TB	IP	TB	E
S	0	0	0	0	0	0
I	1	0	0	0	0	0
TB	1	0	0	0	0	0
IP	1	0	0	0	0	0
TB	0	0	0	1	0	0
E	0	1	1	0	1	0

Figure 18: Rico ID 1074 adjacency matrix with an arc between input and text button interface component as seen in Figure 16.

for Rico ID 1074. The corresponding adjacency matrix is show in Figure 18. In the adjacency matrix, we have an arc connection between input (IP) and text button (TB) components. The connection between them is represented by the value 1 show on the second last row on the matrix. It follows that any number of connected subtasks for related interface components may be represented in the matrix.

3.4 Applying Task Complexity Measures

This section discusses the selection of a particular task complexity formula for measuring task complexity and the necessary definitions for interaction types that are required to obtain consistent numerical measures across different views shown on the same application. Additionally, we define how to combine generating designs and contrasting the impact through task complexity formulas. First we discuss Halstead's E measure which is the easiest to compute, followed by Wood's task complexity measure. In this work, we will not use McCabe's measure, as the graphs do not work well for obtaining varying measures for connected components in Equation 7. We also will not employ the entropy measure used by [35, 52, 62] despite the existence of empirical support for their relation to subjective task complexity and task performance measurement. This is due to the monotonicity of the neighboring nodes, which are primarily S and E nodes.

3.4.1 Halstead's E Measure

Halstead's E measure is the simplest task complexity measure to compute. Based on the formula presented in Equation 6 we simply need the unique and total counts for both operators and operands. We define the operators conceptually equivalent to the types of interactions performed to interact with user interface components within the scope of our work. Similarly, we define operands to be conceptually equivalent to interface components which the user will be operating on.

We use a somewhat arbitrary decision to not count the start (S) and end (E) nodes when computing task complexity. This will also apply to Wood's task complexity later. What this means is we will count only the user interface components and their related interaction modes. Depending on the scope where task complexity is studied for user interfaces including the start and end nodes may be beneficial. This may be if they for example represent the complexity experienced by a user when performing an observation of the interface. For our initial demonstration purpose, these are not a key consideration, albeit they affect the numerical values.

For the user interface with Rico ID 2789 presented in Figure 15 we can easily use either the graph or the adjacency matrix from Figure 17 to compute the E measure. Firstly, the total number of operands $N_2 = 2$ is simply the number of interface components on the graph or the adjacency matrix. Next we obtain total operators $N_1 = 2$ from the total number of interaction options which are determined by the interface components. In our context, it is the same value as the total operands count. For unique operands $\eta_2 = 1$ we simply take the set of all interface components in the interaction graph. Similarly, the number of unique operators $\eta_1 = 1$ is the set of edges of the graph excluding the O-value. By plugging the values into Equation 6 we obtain E = 4 as the task complexity measure. Rico ID 1074 offers a slightly more complicated example, since it has three different types of components and two types of interaction modes. Using the same principles we obtain $N_1 = 4$, $N_2 = 4$, $\eta_1 = 2$ and $\eta_2 = 3$. The corresponding task complexity measure is E $\approx$ 24.8.

3.4.2 Wood's Task Complexity

When considering Wood's task complexity as discussed in Section 2.3.4 our work will not make use of dynamic task complexity from Equation 3 as stochasticity is beyond the scope of this work. Equations 1 and 2 for component complexity and coordinative complexity are within the scope of this work. As discussed in Section 3.4.1, we will not count S and E nodes. This will affect the measures obtained from both of the Wood equations.

For Rico ID 2789 in Figure 15 we first compute component complexity TC1 using Equation 1. The number of subtasks on the graph is 2 with each subtask containing exactly one act of tapping on a text button. The buttons themselves represent one information cue for completing the acts. The text on top of the interface screenshot represents one more potential information cue. We however cannot reliably quantify such cases correctly within the scope of this work. As a consequence, only the components to be interacted with will be used as information cues. A potential solution to this limitation will be described in Section 6.2. With these criteria, we obtain $TC_1 = 2$ for Rico ID 2789. Using Equation 2 we obtain $TC2 = 0$ for coordinative complexity (precedence relations) as we are not counting the S node as parts of walks on the graph. Using Equation 5 with, $\alpha, \beta = 1$ we obtain $TC_t = TC_1 + TC_2 = 1 + 0 = 1$.

Rico ID 1074 in Figure 16 we again compute component complexity using Equation 1. In this case, we obtain $TC_1 = 4$ for the whole graph. We see that the interface contains text that likely behaves as information cues for performing some acts, such as what to do if you did not receive a verification code. Given algorithmic limitations in Section 4 we will not count these specific information cues again. Using Equation 2 we obtain $TC_2 = 1$. On the graph for Rico ID 1074 the right-most text button (TB) has input (IP) as the precedence relation. Based on Equation 5 we obtain $TC_t = TC_1 + TC_2 = 4 + 1 = 5$.

4 Algorithmic Computation Procedure

Turning a manual computational procedure into an automated procedure can be a non-trivial task. Add to that using a collection of UIs created by others rather than self-created sandbox UIs, and the difficulty to automate becomes even greater. As a consequence, in this chapter, we describe how the computation of task complexity for 1460 interfaces from Enrico [9] has been implemented using adjacency matrices and directed graphs. The graph and task complexity computation implementation can be divided to five sub-phases.

1. Definition of interface component types to human-computer interaction modes for mobile devices. We demonstrated this procedure in Sections 3.1 and 3.2.
2. Extraction, sorting and determination of human interactable control components from Enrico [9].
3. Computing the adjacency matrix structure based on the sorted components.
4. Defining the vertices and edges based on the adjacency matrix.
5. Computation of task complexity measures for Halstead's E measure and Wood's task complexity measure.

Each of the sub-phase is discussed in a subsection in the same order as listed above.

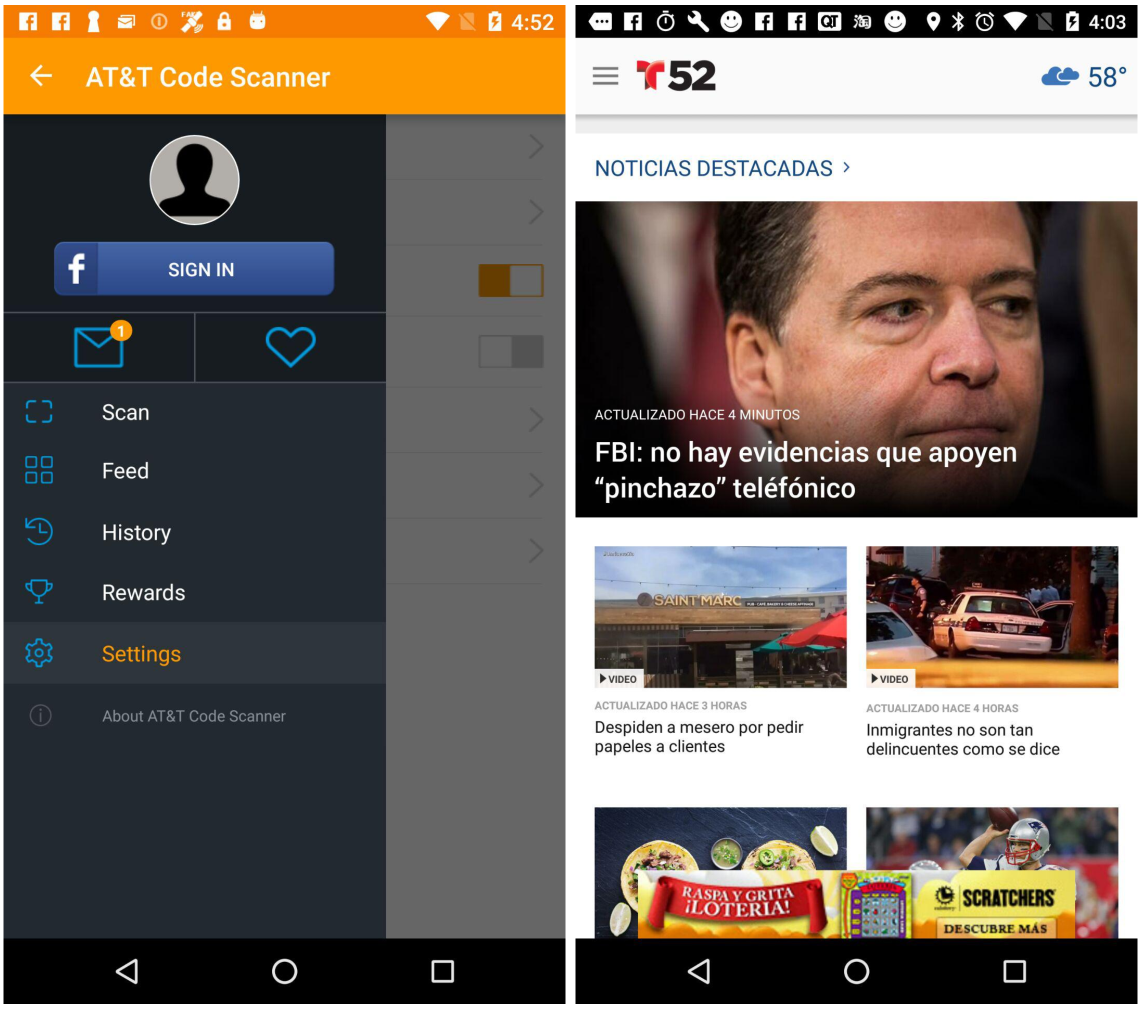
4.1 Interactable Components Extraction

We extract each interactable component by searching for the *clickable* key in the JSON file (for an example, refer to Figure 1). The interactable component extraction is done such that each top level component is checked for clickability first. If it is not clickable and the component has child components placed inside it, then we check if those are interactable. This process is repeated until an interactable component is found or there are no child components to check anymore.

After extracting the interactable components, we perform sorting on the components. We sort the components in order to determine in what order a person might interact with interactable components when there are components such as an input field or checkbox (see Table 3) on the interface that do not lead to the end of a task. The sorting is done to determine which interactable components need to be connected to another interactable component rather than the task end node. The sorting algorithm is such that the component with the highest vertical axis values are placed first compared to lower components. If there is overlap between the components within vertical axis bounds, we perform comparison between the horizontal axes to determine which component is higher and place it first.

4.2 Computing Adjacency Matrix

We follow the principles discussed in 3.3.2 for formulating the adjacency matrix, but focus on briefly discussing the automated formulation of the adjacency matrix. The



ACTUALIZADO HACE 4 MINUTOS
FBI: no hay evidencias que apoyen "pinchazo" telefónico

ACTUALIZADO HACE 3 HORAS
Despiden a mesero por pedir papeles a clientes

ACTUALIZADO HACE 4 HORAS
Inmigrantes no son tan delincuentes como se dice

Figure 19: Rico IDs 30348 (left) and 36999 (right). On the left we see an example where the list items are supposed to be clickable, such as the one containing the *Scan* text and the corresponding icon on the left, but instead the whole Drawer component was marked as clickable. On the right the list items were not marked as clickable containing news items and additionally the top Toolbar was marked as clickable, although the child icons are the clickable components in reality.

adjacency matrix is formulated from a collection of vectors, where the first row vector is the initial observation performed by the user. In our setting, where we consider only one task at a time, all the values in the first row vector are zeroes. This also means it does not add to task complexity by itself. The interactable component row vectors will be formulated with a for-loop one-by-one in the same order they were sorted earlier in Section 4.1.

When running the for-loop, there are three required central concepts to place the interactions and their relations in the correct order automatically in the adjacency matrix. The first concept is the concept of an end point counter, which is initialized with the same value as the length of interactable components. Each end point accounted for in the counter means, there shall be columns with one as the value corresponding to the counter count on the last row of the matrix. The ones are placed starting from the second column of the matrix towards the end but never to the first or last columns which represent start and end nodes. The number of end points can be reduced while in the for-loop if components are found that do not end the task, and the number of zeros on the last row is increased correspondingly.

The second central concept is the concept of an anchor column. The purpose of the anchor column is to determine how to connect interactable components that do not end the task chosen by the user. As detailed in Section 3.3.1 when completion of a task requires an action such as an input the input is usually connected to something like a button that will bring the task such as user registration to completion.

The for-loop applies three different logical rules to each interactable component row vector, where each rule is checked for with an if-else clause. The standard action performed by the else-clause has a purpose to place a task ending user interaction to the adjacency matrix. A task ending interaction here means that the last row of the adjacency matrix will have a connection to this particular row. In practice, the connection is defined by placing value one on column two of the last row if the first sorted component is a task ending interaction.

In the second if-else condition, we process components that do not end the task, such as a text input field. The particular significance of this clause is that it modifies the currently active anchor column we described above, in addition to adding a new row describing the possible tasks the user may complete. The currently active anchor column is set to match the column position of the matrix representing this particular component. The position of the anchor component on the columns is known due to the sorting procedure described in Section 4.1 (the first component has column position 1 when counting starts from 0). The next component in the for-loop will be attached to this anchor component by setting the anchor column value to one for the currently active component row. This is visualized in figures 20 and 21, where the Rico ID 492 interface has an input field followed by a button that must be clicked to finish the task of sending a comment. In this case, the input component (IP) with "Send a comment" text sets the current anchor column value to 5, and the following text button (TB) component located on the second last row will have a value 1 set to column 5. The task for sending a comment has the task finishing exit node attached to the text button, as shown on Figure 21.

In the first if-else condition, we process a special case of the second condition. This is for cases where there may be an interface with an input field but no subsequent button. Rather, these types of interfaces are such that the application will automatically bring the task to end when the user has for example inputted a number code for a verification procedure. Figure 22 showcases an example scenario for Rico ID 10482. The trigger for this condition is the existence of a not task ending component as presented in Table 3 and the component being the last in the sorted component array. We acknowledge that this may not be a foolproof way to identify such components, but have not come across other scenarios like this so far. With the adjacency matrix constructed, we are ready to define the vertices and edges of the directed graph.

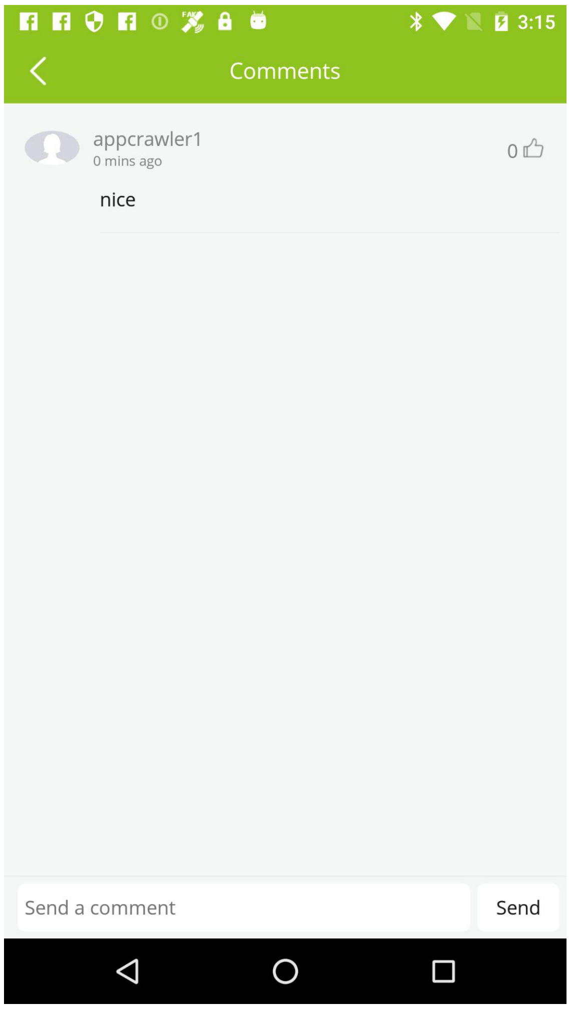
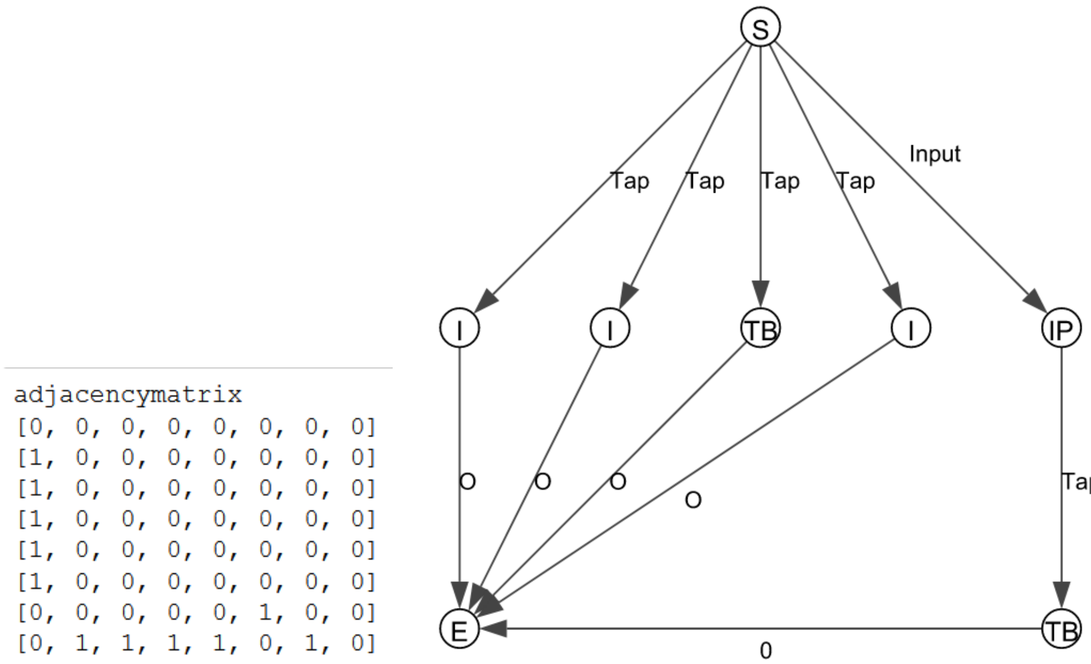


Figure 20: Rico ID 492.

4.3 Computing Vertices and Edges

The vertices and edges are generated in a format suitable for Python implementation of the Igraph library [67]. Between the vertices and edges, the vertices are defined first, as generating the edges depends on the vertices. The vertices for each component are named according to the abbreviation-column presented in Table 3 while the associated interaction mode defined as an edge is based on the interaction-column of the same table.

In order to algorithmically determine the correct vertex positions for each component, the adjacency matrix per column non-zero value counts need to be extracted from the adjacency matrix. We should take note that the first and last row of the adjacency matrix are not included in determining the vertex locations. The start and end vertex locations are known beforehand, so they can be placed at the beginning and end of the vertex location vector automatically.



```
adjacencymatrix
[0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 1, 1, 1, 1, 0, 1, 0]
```

Figure 21: Rico ID 492 adjacency matrix and directed graph construction based on the JSON file.

4.4 Task Complexity Computations

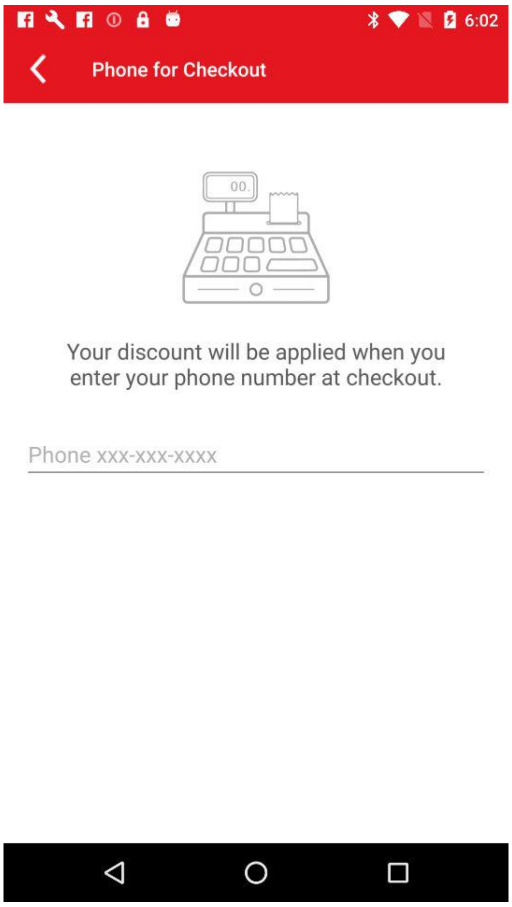
Here we discuss how to compute two task complexity measures algorithmically with general background of the algorithms described in Sections 2.3.4 and 2.3.5, and manual computation procedure described in Section 4.4.

For computing Halstead's E measure, we take the length of the set of edges for unique operators count, n_1 as described in Section 2.3.5. For the number of unique operands, n_2 we take the length of the set of vertices but subtract two so the start and end nodes are not counted. For total operator frequency N_1 and operand frequency N_2 we take the length of the arrays, but for N_2 we subtract two again. With this, we can simply plug in the variables to Equation 6. Compared to Halstead's measure computing, Wood's total task complexity based measures is considerably more challenging for interfaces, which we will discuss next.

4.4.1 Wood's Task Complexity

As described in Section 3.4.2 we compute Wood's task complexity measure for component complexities and coordinative complexities. Based on these two measures, we compute the total task complexity measure based on Equation 5. It is worth noting that in our work the α, β, and γ for Wood's task complexity are all 1 and can be omitted as we have not done empirical determination for them. For reference on empirically determining the values or using different reference values, see [35, 50].

Computing Wood's component complexity is trivial in this work, as we are not able to infer all the information cues for performing each act from the interfaces


Phone for Checkout

Your discount will be applied when you enter your phone number at checkout.

Phone xxx-xxx-xxxx

Figure 22: Rico ID 10482 showcasing a special case for an input field where there is no subsequent task ending component. Rather, the UI should automatically move forward to a new state and end the current task when the user has inputted a phone number.

correctly. This issue is treated more in Section 3.4.2. As a consequence, TC_1 from Equation 1 is simply computed as the length of the vertices array minus two to not count the S and E nodes.

TC_2 measure from Equation 2 is computed by employing a cursor inside the adjacency matrix. The algorithm loops through each of the sorted control components. Each of the components on the adjacency matrix has one as a value representing a connection to a preceding component, such as in Figure 21. On the rightmost text button component, the algorithm finds the position of the current component and rolls the cursor position (row number) on the adjacency matrix backwards inside a second loop until a component attached to the first column is found. Each time the cursor is moved adds one to the precedence relation count for TC

5 Results

In this section, we describe the results obtained from algorithmic inference of task structure for Enrico interfaces based on the algorithm described in Section 4. First, we will review the quality of a random sample of N = 30 graphs. The review will help identify previously unrecognized issues with Rico dataset and analyze limitations with the task structure inference algorithm. Secondly, we will visualize the task complexity measures obtained for the algorithmically generated graphs.

5.1 Graph Issues Identification

In this section, we take a random sample of N = 30 from the algorithmically generated task graphs and compare them to human understanding of the correct task structure. A sample size of 30 will offer a 90% confidence interval and a 15% margin of error for 1413 graphs. This review process will help us identify two types of errors in the generated graphs. The first type of error includes errors caused by incorrect data in the view hierarchy file. The second type of error includes incorrect assumptions made in the inference algorithm. The graph quality review process is performed as follows:

1. Compare algorithmically generated graph to UI screenshot for consistency.
2. If any inconsistencies are identified, review the graph and screenshot against the view hierarchy file to identify the cause of the inconsistency.

The results of the manual review are listed in Table 4 and the related UI screenshots and graphs are presented in Figure 24. Out of the 30 algorithmically generated interaction graphs, we have 11 type 1 issues and 4 type 2 issues. Finally, we did not find issues between the interface and the interaction graph for 15 of the random samples.

As an example of the review procedure, when we consider the graph for Rico ID 21070 located top left corner of Figure 24 we see that the task structure is correct except that there is one node that we cannot verify with certainty to be correct. The leftmost I node in the graph represents the menu icon located in the top left corner of the interface. The L nodes represent list item icons containing descriptions of the games and their results. Each list item seems to be clickable on the interface for the purpose of starring the game or game series as a favorite. Next, we have a TB node at the second rightmost position in the graph. The node represents the *Live* text in the toolbar. We cannot verify with certainty that this text should be clickable, as based on human intuition it looks more likely to be just a header text. Lastly, we have a WV node that describes a Web View component located at the very bottom of the interface that displays an advertisement to the user. The tasks represented on the graph include the following:

- Click on the top left menu icon in order to navigate to another part of the application.

ID	Type	Explanation
21070	0	Cannot verify if the Live text button in toolbar is clickable.
52188	0	
46877	1	Top left corner back arrow Icon not marked as clickable
36664	1	Stats, Diary and Growth not marked as clickable
10798	1	Clickable components listed from underneath side menu
43093	1	Contains multiple views and Pager component is not clickable
9438	2	The algorithm assumes Input connected to another component
61683	0	
12804	0	
58529	1	Components under side panel marked as clickable
64005	0	
39118	0	
68924	0	
53268	0	
64417	0	
28646	0	
28021	0	
36999	2	The algorithm failed to find all 11 clickables
64461	1	Hierarchy describes a hidden card component
41976	2	Misclassified toolbar components as not clickable
33868	0	
46677	0	
1115	1	Passbook text is probably not clickable
69913	0	
17169	1	Components under side panel marked as clickable
21103	2	Failed to represent multiple Inputs correctly
21397	1	Toolbar Text components not marked as clickable
8592	1	List items marked as clickable, not children
60061	1	Text link not marked as clickable
1028	0	

Table 4: Manual review results for a random sample of 30 graphs. Type 0 means the graph matches the UI screenshot based on manual review. Type 1 means an inconsistency caused by incorrect data in the view hierarchy. Type 2 means inconsistency between the screenshot and graph due to limitations with the algorithm.

- Select a game or a game series as a favorite by clicking on the list item representing that. particular game or series. There are 12 options to choose from in total.
- Click on the advertisement at the bottom of the interface in order to access an audiobook.

The second interface for from the second row in Figure 24 for Rico ID 9438 on the other hand has 3 paths starting from the S node at the top. This does not match expected behavior. We see one path for opening a menu in the top right corner. The menu is represented by the I-node for an icon. At the bottom of the interface we see two text buttons represented by TB nodes which allow the user to log in or create an account. Lastly, the input node represented by IP in the graph seems to be such that once the input has been entered correctly, the application will attempt to join a webinar automatically. Based on this, we expect 4 nodes to start from the S node. In other words, the algorithm incorrectly assumes that after the input is complete, a button can be clicked to complete the task of joining a webinar. The tasks represented by the graph we have generated are as follows:

- Press in the top right corner Icon to access a menu.
- Enter a webinar ID in the input field and press *Try GoToWebinar for free* button to join the webinar.
- Press *Sign in* text button to log in to your user account.

In addition to the issues identified in the manual review procedure, the algorithm failed to generate graphs for 70 Enrico IDs due to errors during runtime. The specific interfaces where we were not able to generate a graph have been listed in Appendix A. The primary reasons for generating a graph included division by zero, unexpected component in the view hierarchy marked as clickable and math domain error. Division by zero is caused by the view hierarchy not having any components marked as clickable. An example of such a UI is provided in Figure 23 on the right, where the interface clearly has clickable components, but the view hierarchy does not describe them correctly. In the same figure, Rico ID 50400 on the left has a Drawer component incorrectly marked as clickable.

5.2 Task Complexity Measure Results

The task complexity measures presented here are measuring the total task complexity of the graphs we generated based on the algorithmic computation procedure in Section 4. This means each measurement looks at the whole interface and does not measure the effects of individual subtasks, such as "click on a menu icon to access another part of the application". More specifically, we have obtained task complexity measure results for Halstead's E measure and Wood's task complexity. The results for Halstead's E measure are depicted in Figure 25. Halstead's E measure has a relatively tight spread when measured in total control components for each interface.

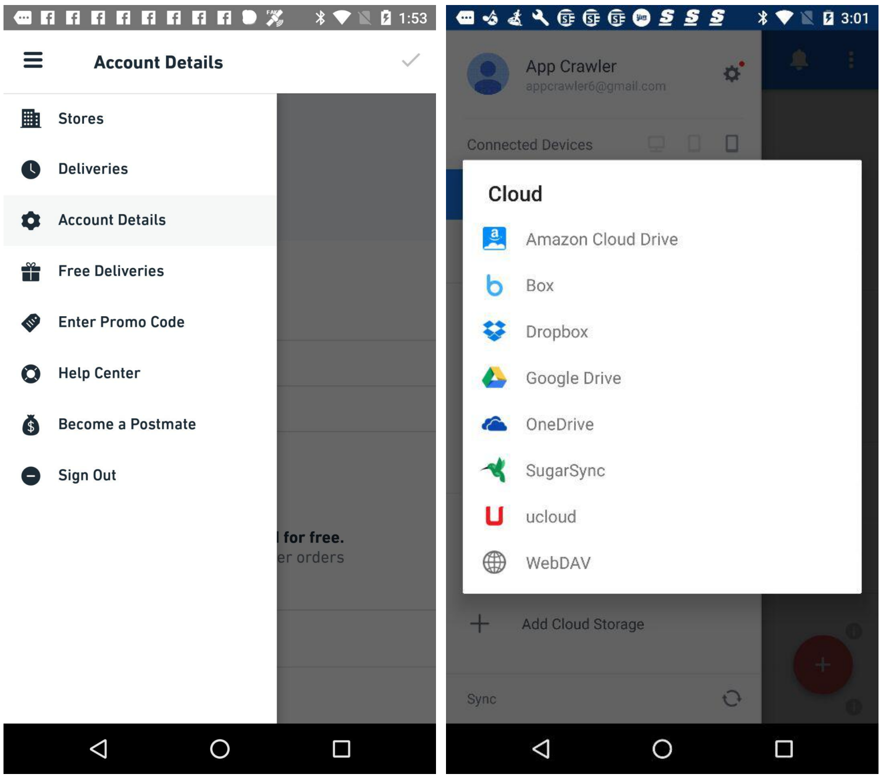
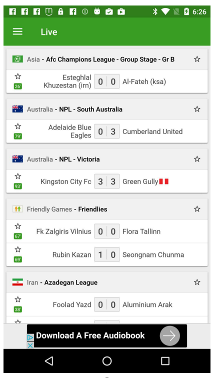
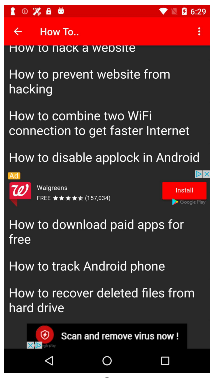


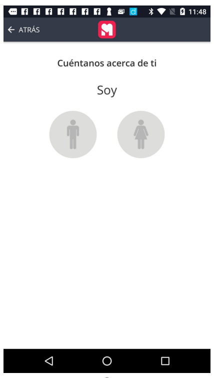
Figure 23: Rico ID 50400 (left) with Drawer component incorrectly marked as clickable when the List Item child components should be marked as clickable. Rico ID 54399 (right) has a view hierarchy where no components are marked as clickable, although there should be clickable components.

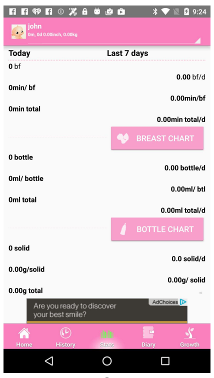
The spread has more variability when measured in the number of paths, on the other hand. The thing worth noticing with Halstead's E measure is that the complexity measure is exponentially upwards sloping. At the same time, [52] had to match task completion time with an exponential function to an entropy based (see Equation 8) TACOM formula [52]. The TACOM formula yielded complexity values above 0 but less than 4 for nuclear power plant emergency operating procedures. However, no definitive conclusions can be drawn from this as the context of our work is less complex.

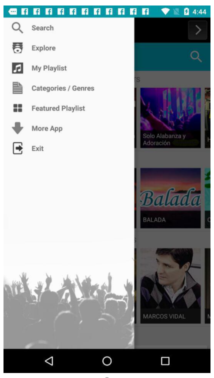
For Wood's task complexity, we have visualized the component complexity results in Figure 26 and coordinative complexity results in Figure 27. The component complexity measure behaves perfectly linearly with respect to total number of control components, as we have not made any modifications to Equation 1. With number of paths we see some spread upwards as the number of paths can be less than the total number of components. Wood's coordinative complexity measure has more variability in the measure behavior, with the variability explained by a reduction in path counts when compared to control component counts. A ratio of less than 1:1 between paths and control components means the interface has components that do not end the task. Each level of depth adds $n!$ to coordinative complexity measure, where n is the count of precedence relations the last node on that walk has. This

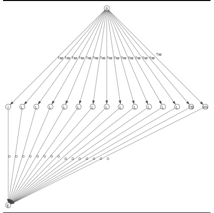
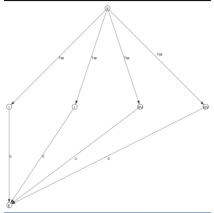
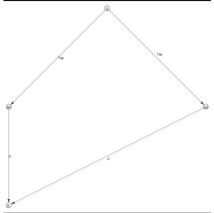
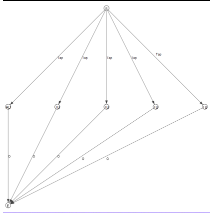
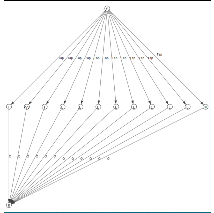

Live
Asia - Afc Champions League - Group Stage - Gr B
Esteghlal Khuzestan (irn) 0 0 Al-Fateh (ksa)
Australia - NPL - South Australia
Adelaide Blue Eagles 0 3 Cumberland United
Australia - NPL - Victoria
Kingston City Fc 3 3 Green Gully
Friendly Games - Friendlies
Fk Zalgiris Vilnius 0 0 Flora Tallinn
Rubin Kazan 1 0 Seongnam Chunma
Iran - Azadegan League
Foolad Yazd 0 0 Aluminium Arak
Download A Free Audiobook

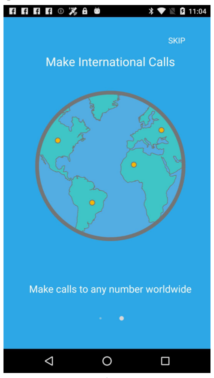

How To..
How to hack a website
How to prevent website from hacking
How to combine two WiFi connection to get faster Internet
How to disable applock in Android
Walgreens FREE ★★★★★ (157,034) Install
How to download paid apps for free
How to track Android phone
How to recover deleted files from hard drive
Scan and remove virus now !

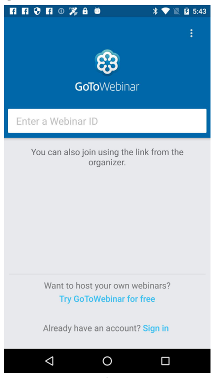

ATRÁS
Cuéntanos acerca de ti
Soy

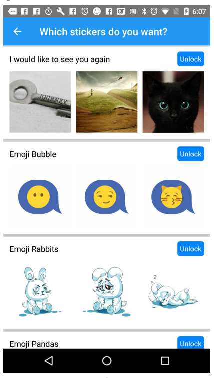

john
Today Last 7 days
0 bf
0.00 bf/d
0min/ bf
0.00min/bf
0min total
0.00min total/d
BREAST CHART
0 bottle
0.00 bottle/d
0ml/ bottle
0.00ml/ btl
0ml total
0.00ml total/d
BOTTLE CHART
0 solid
0.0 solid/d
0.00g/solid
0.00g/ solid
0.00g total
Are you ready to discover your best smile?
Home History Diary Growth

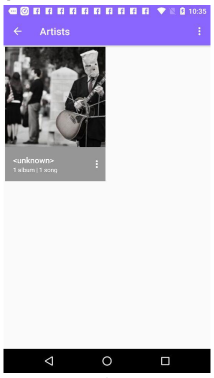

Search
Explore
My Playlist
Categories / Genres
Featured Playlist
More App
Exit

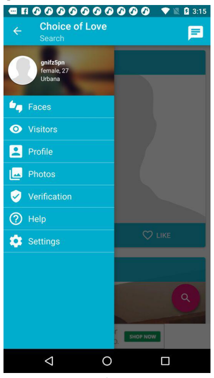






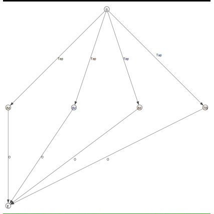
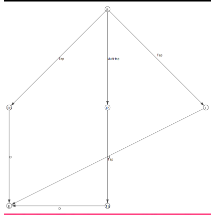
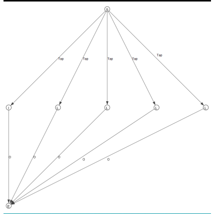
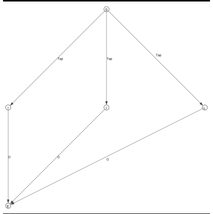
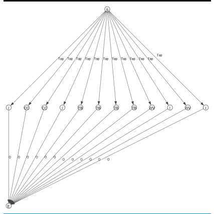

SKIP
Make International Calls
Make calls to any number worldwide

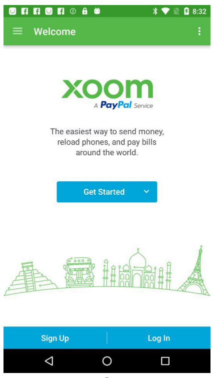

GoToWebinar
Enter a Webinar ID
You can also join using the link from the organizer.
Want to host your own webinars?
Try GoToWebinar for free
Already have an account? Sign in

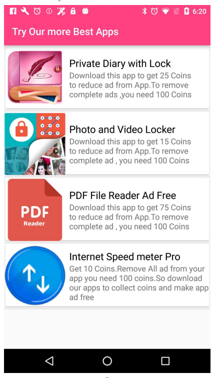

Which stickers do you want?
I would like to see you again Unlock
Emoji Bubble Unlock
Emoji Rabbits Unlock
Emoji Pandas Unlock

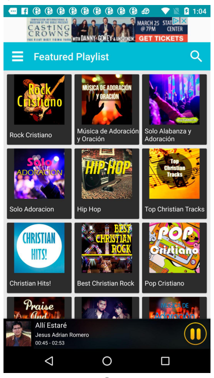

Artists
<unknown>
1 album | 1 song

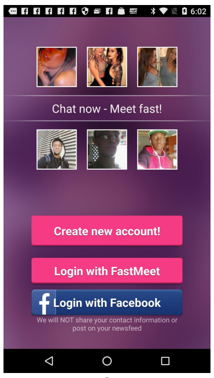

Choice of Love
Search
Faces
Visitors
Profile
Photos
Verification
Help
Settings
LIKE

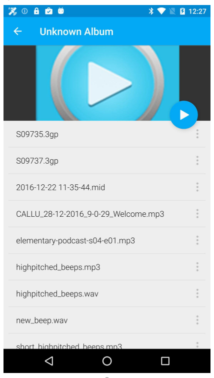






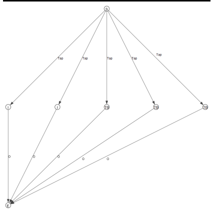
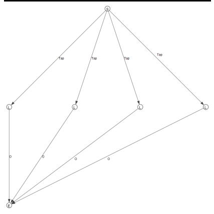
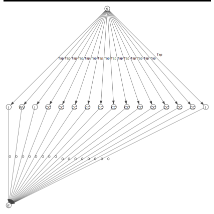
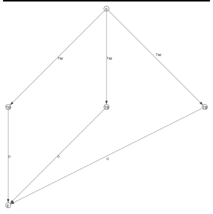
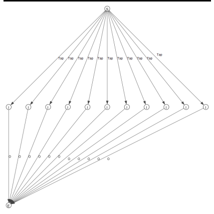

Welcome
xoom
A PayPal Service
The easiest way to send money, reload phones, and pay bills around the world.
Get Started
Sign Up Log In


Try Our more Best Apps
Private Diary with Lock
Download this app to get 25 Coins to reduce ad from App.To remove complete ads ,you need 100 Coins
Photo and Video Locker
Download this app to get 15 Coins to reduce ad from App.To remove complete ad , you need 100 Coins
PDF File Reader Ad Free
Download this app to get 75 Coins to reduce ad from App.To remove complete ad , you need 100 Coins
Internet Speed meter Pro
Get 10 Coins.Remove All ad from your app you need 100 coins.So download our apps to collect coins and make app ad free


Featured Playlist
Rock Cristiano
Música de Adoración y Oración
Solo Alabanza y Adoración
Solo Adoracion
Hip Hop
Top Christian Tracks
Christian Hits!
Best Christian Rock
Pop Cristiano
Allí Estaré
Jesus Adrian Romero


Chat now - Meet fast!
Create new account!
Login with FastMeet
Login with Facebook
We will NOT share your contact information or post on your newsfeed


Unknown Album
S09735.3gp
S09737.3gp
2016-12-22 11-35-44.mid
CALLU_28-12-2016_9-0-29_Welcome.mp3
elementary-podcast-s04-e01.mp3
highpitched_beeps.mp3
highpitched_beeps.wav
new_beep.wav
short_highpitched_beeps.mp3

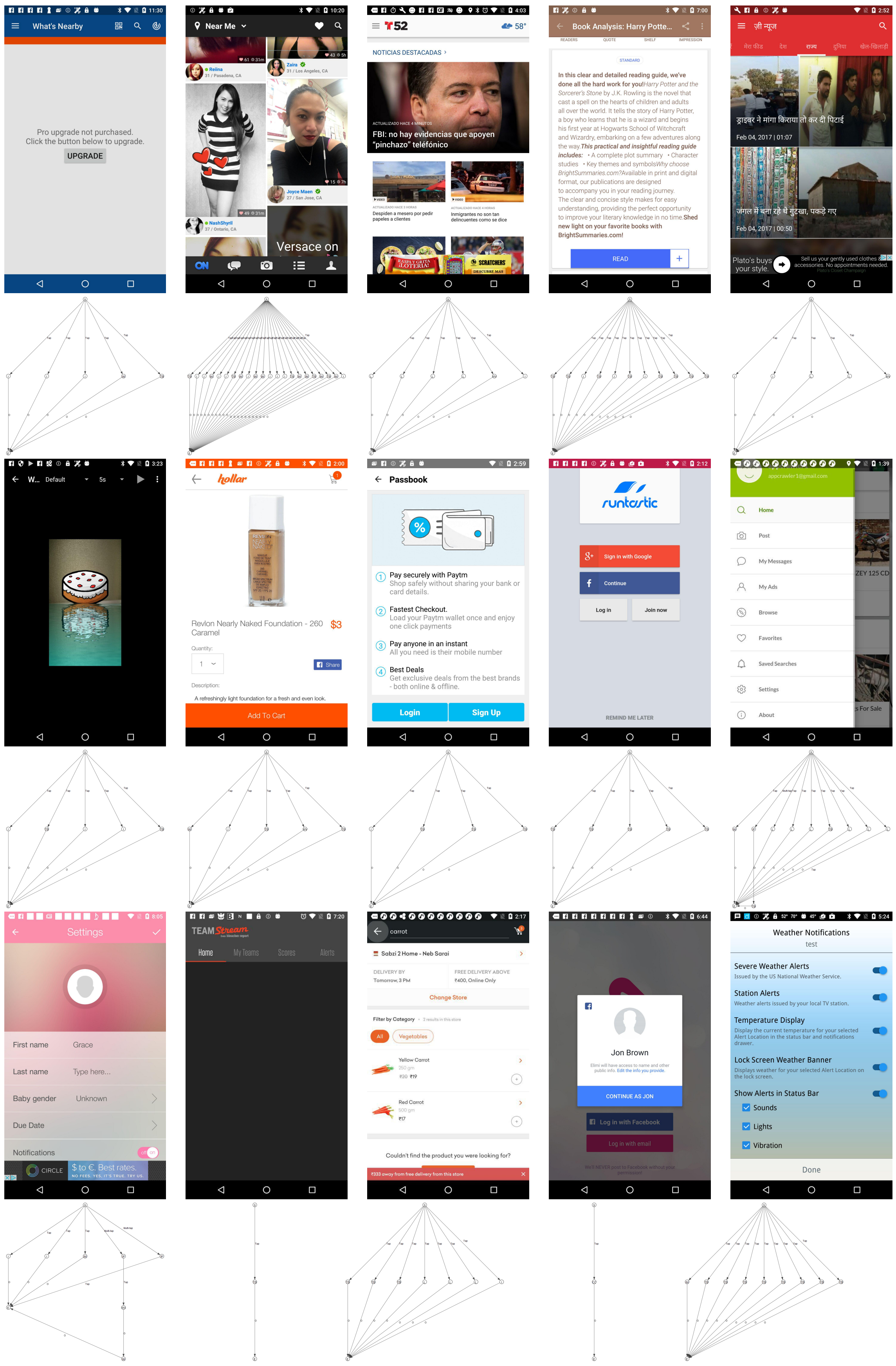


Figure 24: Manually reviewed graphs with the IDs corresponding to Table 4 so that the first table ID is in the top left corner and the last ID is in the bottom right corner. Each UI screenshot has the corresponding graph directly underneath it.

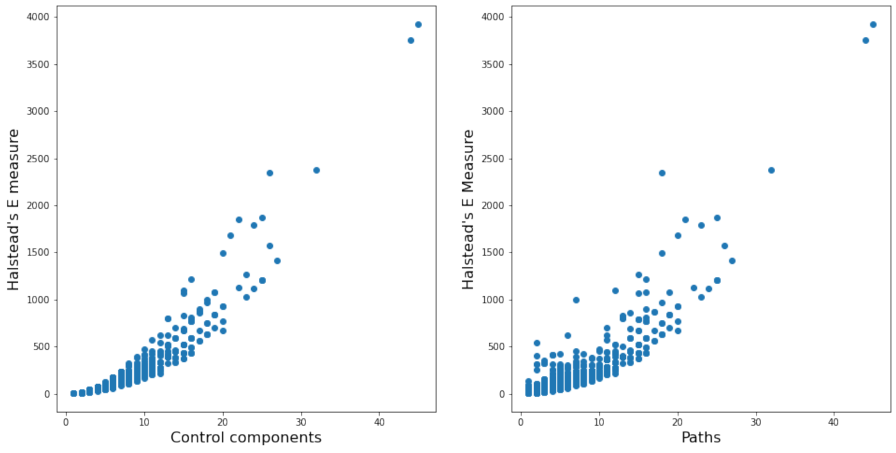


Figure 25: Halstead's E measure results. On the left we have results with respect to total number of control components and on the right with respect to number of paths starting from the S node.

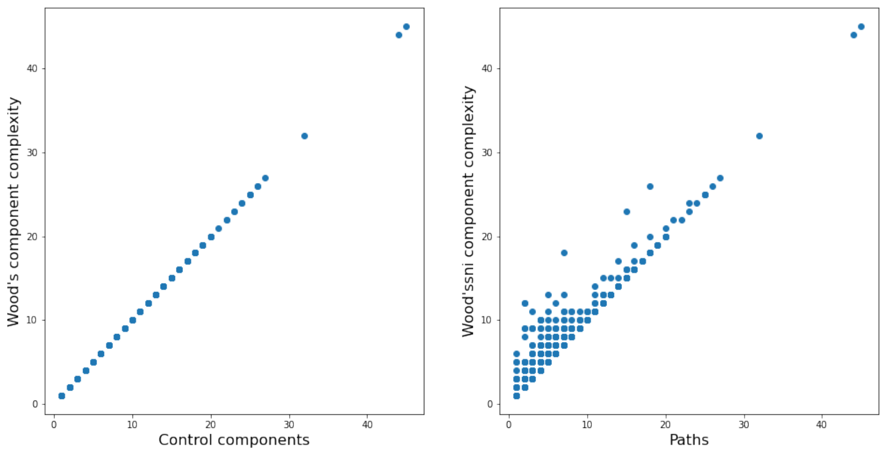


Figure 26: Wood component complexity results.

behavior is explained by Equation 2.

The results for Wood's total task complexity are visualized in Figure 28. The results are the sum of coordinative complexity and component complexity measures. The measure simply has the joint behavior of Figures 26 and 27. We can expect the behavior of the measure to be different if the algorithm measuring component complexity has the ability to infer all the information cues related to performing an act on each component. However, the algorithm in this work does not have the capability.

In Figure 29 representing Rico ID 27152 UI and interaction graph, we see an

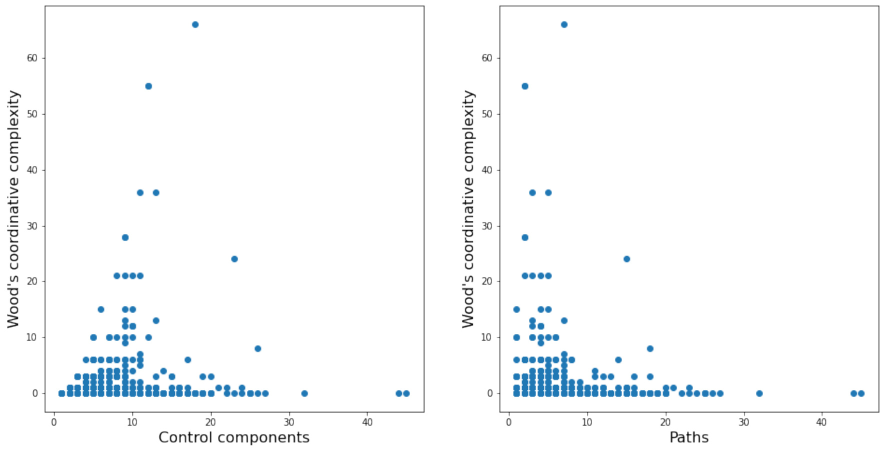


Figure 27: Wood coordinative complexity results.

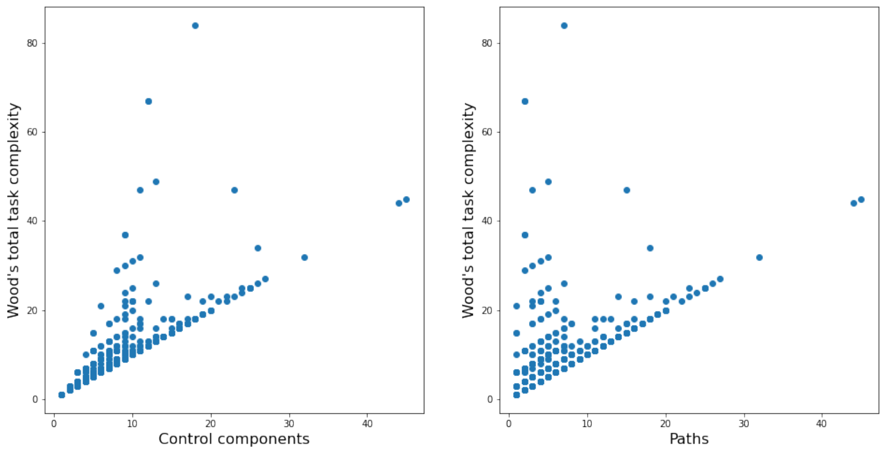
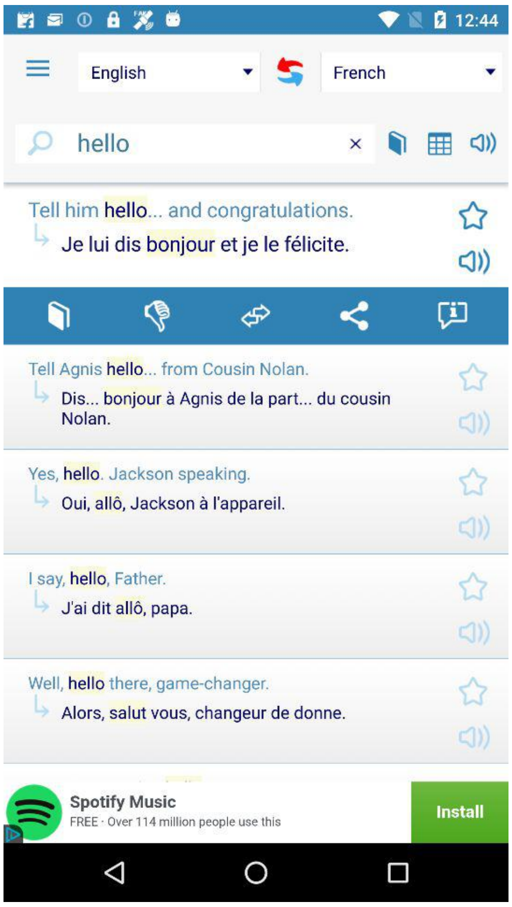


Figure 28: Wood's total task complexity results.

example of a relatively complicated UI according to the Halstead E and Wood's total task complexity measures. Similarly, the UI has 25 control components and 25 paths, making it a relatively complicated interface. The figure also highlights one of the key limitations with the results. In Table 2 Liu and Li [57] compiled a list of factors that affect task complexity. The interface has a notable level of redundancy, which reduces the level of task complexity. As an example of redundancy, the interface uses the same star and speaker icons on the right side. However, the results for Halstead E measure increase exponentially. As a consequence, the measure only captures task complexity increasing factors in the form of operators and operands, while not accounting for task complexity reducing factor or other task complexity increasing factors.


12:44

English

French

hello

Tell him hello... and congratulations.

Je lui dis bonjour et je le félicite.

Tell Agnis hello... from Cousin Nolan.

Dis... bonjour à Agnis de la part... du cousin Nolan.

Yes, hello. Jackson speaking.

Oui, allô, Jackson à l'appareil.

I say, hello, Father.

J'ai dit allô, papa.

Well, hello there, game-changer.

Alors, salut vous, changeur de donne.

Spotify Music

FREE · Over 114 million people use this

Install

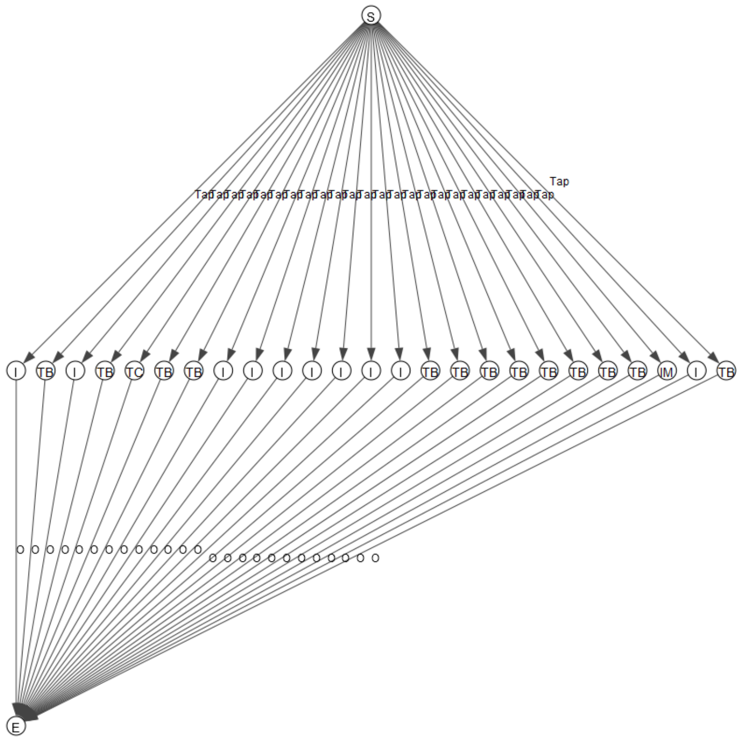


Figure 29: Interaction graph results for Rico ID 27152. The graph and interface are among the more complicated in the dataset, with Halstead E measure at 1211.7 and Wood's total task complexity at 25.

6 Discussion

In this work, we aimed to research what types of challenges exist for inferring task structure and what types of task complexity measures would be suitable for the graphs used in this work. Understanding the challenges will allow us to design better algorithms for task structure inference in the future. Additionally, we are better able to understand what types of factors need to be accounted for in the dataset being used for the inference task in order to prevent graph quality issues.

In Section 3 we first described how a human would infer task structure from interfaces with manual computation and analysis. This was followed by computation of task complexity measures. Manual computation allowed us to understand what types of steps are required in inferring task structure and complexity measures from view hierarchies. With this understanding we moved on to Section 4 where we focused on turning the manual computation procedure into an algorithmic version. The benefit of the algorithmic inference procedure is that we are able to analyze a large amount of interfaces in a shorter amount of time. Lastly, in Section 5 we reviewed a random sample of 30 graphs in order to identify potential quality issues with the underlying dataset and the graph generating algorithm. Then we presented the results obtained from computing task complexity measures for the graphs and discussed their behavior in light of component counts and number of paths on a graph.

In the upcoming two sections we will first discuss the limitations of this work including with using a pre-existing dataset, the way we have formulated the graphs, limitations of the algorithm, and the graph quality review we performed. Secondly, we discuss future work that can be performed. The areas we will discuss for future work include requirements on datasets for inference, what directions inference algorithm improvement may take, and what types of studies can be performed to verify the applicability of different task complexity measures for task complexity as experienced by real users.

6.1 Limitations

When considering the results obtained during quality assurance in Section 5.1 we see that problems with the dataset were quite frequent. This is despite Leiva, Hota and Oulasvirta [9] generating a high quality dataset suitable for layout design categorization with manual review from the original Rico dataset [11]. However, looking at clickability in the view hierarchy was beyond the scope of their work. We note that this seems to be one of the few works where the correctness of clickability for Rico view hierarchies has been reviewed and can be considered to augment the work of [9] in understanding what types of quality issues exist for large-scale algorithmically generated datasets. Schoop et al. [12] performed research that has looked into view Rico view hierarchy tappability. One of the primary causes for the issues seems to be components not being correctly marked as clickable. From the perspective of our work, the primary limitation for the graphs presenting the task structure correctly is formed by quality issues in the Rico view hierarchy.

As a consequence, we are able to estimate that approximately 35 to 65% of the graphs are correct. However, the correctness of the generated graphs does not affect the validity of the results obtained in Section 5.2 as they demonstrate how the measures behave with respect to different graph shapes rather than human performance or experience. However, understanding on the applicability of the task complexity measures used in this work to human perception of task complexity is still limited, as we did not investigate subjective task complexity. Rather, we were focused on the implementation of the algorithm as a proof of concept. Generally, understanding of task complexity measures relationship to human task performance on interfaces [68] is limited.

The graphs we generated based on Section 3 have a limited range of application. The graphs don't account for situations where stochasticity is involved, such as when a user will perform an unknown number of interactions, such as with an Input or a Map View component. However, the stochasticity limitation can be resolved by using a transition matrix as described in Section 2.2. Additionally, our graphs do not consider situations where the user may need to perform multiple actions across components before finishing a task, like activating three switches to on state and then pressing a component that leads to a new view.

The algorithm has limited inference capabilities when a component that does not lead to the end of a task immediately, such as an Input component. When we try to infer which component should be interacted with next, the algorithm may find it difficult to find the correct component. This is especially true if completing the Input operation leads to automatic view refresh, such as in Figure 22. This becomes even more complicated if there are other components directly below the Input component, as the algorithm may assume it to be a task finishing component related to the Input. We found Halstead's E measure (Equation 6) to be particularly suitable for analyzing interactions in the graph format employed in this work. The suitability is due to the E measure having direct correspondence to performing interactions on components through the operators and operands concepts. The primary limitation with Halstead's E measure is that it does not account for task complexity increasing and reducing factors outside of operations and operators, as described in Table 2.

Wood's task complexity measure (Equation 5) shows some promise, but it also requires further development. In Figure 26 the behavior of the metric is trivial since it increases linearly with the number of components. This is caused by the algorithms' inability to determine which non-clickable components act as information cues for the clickable components. We will discuss a potential solution to this limitation in Section 6.2. Lastly, in Section 2.3.5 we evaluated McCabe's v(G) (Equation 7) to not be ideal for the types of graphs presented in this work. Shannon's entropy formula (Equation 8) has been employed on graphs by [35, 50, 51, 52, 53]. There is potential to employ the formula, but it requires further consideration with numerical behavior caused by the S and E nodes. The S and E nodes are suitable for KLM on the other hand, where S node can represent mental preparation time and E node the reaction to system delay after an interaction.

6.2 Future work

The most important future work direction is performing validation for the Wood and Halstead E measure metrics on interfaces. As discussed by Liu and Li [68] there has not been much research validating the (objective) task complexity measures to subjective task complexity and task performance metrics. Especially for graph based measure validation. A few notable works that have done validation for graph based task complexity metrics include [35, 52, 39, 62]. However, none of these have employed Wood's task complexity or Halstead's E measures as metrics. Validating the measures can be accomplished with laboratory testing by obtaining measures of task performance metrics and subjective task complexity ratings. An alternative validation approach is to study the behavior of task complexity metrics against keystroke-level model task completion time estimates [69, 70, 43]. Subjective task complexity can be measured with NASA TLX [71, 35, 51]. Recently, an approach for validating a graph based complexity measure for human errors has also been demonstrated [53]

Secondly, In our work, we faced limitations with inferring the information cues for each tappable component from the interface. This could be amended by using XRAI heatmaps [12] and using components above a certain saliency threshold value as information cues for Equation 1. This would improve the quality of results obtained from Equation 1. Another notable limitation was that not all the components were correctly marked as clickable. Taperception can potentially offer a solution to this limitation.

The third direction for future work can involve applying task complexity measures and as objective functions for interface design [7, 72]. This is the opposite direction of what we performed in this work, as new interfaces are generated from scratch by formulating interaction graphs with desirable interaction sequences and converting them to interfaces where one or multiple objective functions are optimized for. Additionally, when generating interfaces algorithmically based on task complexity modeling, it is possible to use tappability prediction [12] as a measure of design goodness besides total task complexity.

A fourth research direction involves performing more comprehensive computational task analysis on interfaces that span across multiple views or pages on an interface. Analyzing such scenarios requires using tensors to represent the tasks to be performed on the interface. Such task analysis can be used to open new investigation directions into how to optimize navigation structure across multiple views or pages such that a human is able to perform their desired goals easily. In stochastic scenarios, we can apply transition probability tensors and model tasks across multiple views or pages as Markov decision processes. In this line of research, concepts such as small world graphs, strongly connected graphs and the diameter and radius of the graph are of interest as well [17]. As an example, the diameter (eccentricity) of a graph can act as an objective function [7] for multi-view or page navigation structure. The optimization of the graph diameter (maximal distance to reach from interface views or pages the furthest away from each other) becomes particularly interesting when we consider limited space for navigation on each view or page as an interface optimization constraint [7]. A more comprehensive tensor based computational task analysis can also be used to perform a full HTA evaluation.

Lastly, further basic research into generating interaction graphs and numerical task complexity measures could increase the scope of applicability of task complexity measures. In particular, previous work has studied factors affecting task complexity [57]. By studying how to represent these factors as mathematical functions and combining said functions in a manner similar to that performed by Wood [34] it is possible to formulate comprehensive numerical measures of tasks and the complexity associated with performing the task. Mathematical representation of tasks and their nature as graphs can offer new approaches in the field of computational interaction to study the quality of an interface and on the other hand for constructing an interface algorithmically from a task based perspective [7].

7 Conclusions

This work has shown that it is possible to algorithmically infer interface task structure from user interface view hierarchies and computationally perform task analysis based on the inferred structures. The inferred task structure represented as an interaction graph represents the task options a human has when interacting with an interface. Additionally, graphs and matrices seem to be a promising mathematical tool for representing interactions and lend themselves for solving large scale interaction problems. Transition matrices can offer a promising tool to study stochastic task and goal scenarios on interfaces.

Under simple task structure scenarios such as those based on Enrico [9], the largest concern with the inferred task structures are formed by potential quality issues in the view hierarchy correctness. A secondary concern is caused by scenarios where it may be difficult to determine how a component that does not end a task relates to other components on the interface. A particularly challenging aspect is caused by creativity of designers, which an algorithm may have difficulties adapting to creative use of interface elements and interaction modes.

Given the ability to represent tasks and goals as matrices such as adjacency matrices and transition matrices, we are able to employ mathematical techniques on them and compute various mathematical measures depending on the information that is mapped on the matrix. Such mathematical measures offer an avenue for studying algorithmically human performance metrics and experiences on interfaces. This ability can be particularly beneficial for 1) large scale problems where manual task analysis methods do not scale well, and 2) for estimating interface quality early in the development process. This is accomplished by capturing relevant interface qualities affecting interface complexity on the matrix as attributes. This can provide savings in interface development time and cost, as the matrix does not necessarily require a fully developed interface prototype.

References

[1] A. Fabijan, P. Dmitriev, H. Holmstrom Olsson, and J. Bosch, “Online controlled experimentation at scale: An empirical survey on the current state of a/b testing,” in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2018, pp. 68–72.

[2] R. Kohavi and S. Thomke, “The surprising power of online experiments: Getting the most out of a/b and other controlled tests.” *Harvard Business Review*, vol. 95, no. 5, pp. 74 – 82, 2017.

[3] T. Tullis and W. Albert, *Measuring the User Experience, Second Edition: Collecting, Analyzing, and Presenting Usability Metrics*, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013.

[4] F. Auer, R. Ros, L. Kaltenbrunner, P. Runeson, and M. Felderer, “Controlled experimentation in continuous experimentation: Knowledge and challenges,” *Information and Software Technology*, vol. 134, p. 106551, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584921000367

[5] B. Shneiderman, C. Plaisant, M. Cohen, S. Jacobs, N. Elmqvist, and N. Diakopoulos, *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, 6th ed. Pearson, 2016.

[6] W. Hwang and G. Salvendy, “Number of people required for usability evaluation: The 10±2 rule,” *Commun. ACM*, vol. 53, no. 5, p. 130–133, May 2010. [Online]. Available: https://doi.org/10.1145/1735223.1735255

[7] A. Oulasvirta, P. O. Kristensson, X. Bi, and A. Howes, *Computational Interaction*. Oxford University Press, 2018. [Online]. Available: http://www.oup.com/academic/product/9780198799610

[8] B. Fitzgerald and K.-J. Stol, “Continuous software engineering and beyond: Trends and challenges,” in *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*, ser. RCoSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 1–9. [Online]. Available: https://doi.org/10.1145/2593812.2593813

[9] L. A. Leiva, A. Hota, and A. Oulasvirta, “Enrico: A high-quality dataset for topic modeling of mobile UI designs,” in *Proc. MobileHCI Adjunct*, 2020.

[10] A. Oulasvirta and K. Hornbæk, “Hci research as problem-solving,” in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 2016, pp. 4956–4967. [Online]. Available: https://dl.acm.org/citation.cfm?id=2858283

[11] B. Deka, Z. Huang, C. Franzen, J. Hibschman, D. Afergan, Y. Li, J. Nichols, and R. Kumar, “Rico: A mobile app dataset for building data-driven design applications,” in *Proceedings of the 30th Annual Symposium on User Interface Software and Technology*, ser. UIST ’17, 2017.

[12] E. Schoop, X. Zhou, G. Li, Z. Chen, B. Hartmann, , and Y. Li, “Predicting and explaining mobile ui tappability with vision modeling and saliency analysis,” in *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. New Orleans, LA, USA: Association for Computing Machinery, May 2022, pp. 1–13. [Online]. Available: https://doi.org/10.1145/3491102.3502042

[13] Y. Li, J. He, X. Zhou, Y. Zhang, and J. Baldridge, “Mapping natural language instructions to mobile ui action sequences,” *arXiv preprint arXiv:2005.03776*, 2020.

[14] X. Zang, Y. Xu, and J. Chen, “Multimodal icon annotation for mobile applications,” in *Proceedings of the 23rd International Conference on Mobile Human-Computer Interaction*, 2021, pp. 1–11.

[15] J. L. Gross, J. Yellen, and M. Anderson, *Graph theory and its applications*. Chapman and Hall/CRC, 2018.

[16] M. GAO and H. DO, “Computational methods for socio-computer interaction,” *Computational Interaction*, p. 399, 2018.

[17] H. Thimbleby and J. Gow, “Applying graph theory to interaction design,” in *IFIP International Conference on Engineering for Human-Computer Interaction*. Springer, 2007, pp. 501–519.

[18] N. S. Hardman, *An empirical methodology for engineering human systems integration*. Air Force Institute of Technology, 2009.

[19] N. Hardman, J. Colombi, D. Jacques, R. Hill, and J. Miller, “Application of a seeded hybrid genetic algorithm for user interface design,” in *2009 IEEE International Conference on Systems, Man and Cybernetics*, 2009, pp. 462–467.

[20] S. Combéfis, D. Giannakopoulou, C. Pecheur, and M. Feary, “A formal framework for design and analysis of human-machine interaction,” in *2011 IEEE International Conference on Systems, Man, and Cybernetics*, 2011, pp. 1801–1808.

[21] H. Thimbleby, “Combining systems and manuals,” *People And Computers*, pp. 479–479, 1993.

[22] J. C. Campos and M. D. Harrison, “Interaction engineering using the ivy tool,” in *Proceedings of the 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, ser. EICS ’09. New York, NY, USA: Association for Computing Machinery, 2009, p. 35–44. [Online]. Available: https://doi.org/10.1145/1570433.1570442

[23] H. Thimbleby, "Action graphs and user performance analysis," *International journal of human-computer studies*, vol. 71, no. 3, pp. 276–302, 2013.

[24] C. Baber and N. A. Stanton, "Task analysis for error identification," *Handbook of human factors and ergonomics methods*, pp. 381–389, 2005.

[25] M. Ceci and P. F. Lanotte, "Closed sequential pattern mining for sitemap generation," *World Wide Web*, vol. 24, no. 1, pp. 175–203, 2021.

[26] J. Han, Y. Yu, C. Lin, D. Han, and G.-R. Xue, "A hierarchical model of web graph," in *ADMA*, 2006.

[27] G. M. Youngblood and D. J. Cook, "Data mining for hierarchical model creation," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 37, no. 4, pp. 561–572, 2007.

[28] C. D. Wickens, S. E. Gordon, Y. Liu, and J. Lee, *An introduction to human factors engineering.* Pearson Prentice Hall Upper Saddle River, NJ, 2004, vol. 2.

[29] B. Kirwan and L. K. Ainsworth, *A guide to task analysis: the task analysis working group.* CRC press, 1992.

[30] S. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Pearson, 2022.

[31] Y. Li and O. Hilliges, *Artificial Intelligence for Human Computer Interaction: A Modern Approach.* Springer, 2021.

[32] S. Pateria, B. Subagdja, A.-h. Tan, and C. Quek, "Hierarchical reinforcement learning: A comprehensive survey," *ACM Comput. Surv.*, vol. 54, no. 5, jun 2021. [Online]. Available: https://doi.org/10.1145/3453160

[33] D. Diaper and N. Stanton, "The handbook of task analysis for human-computer interaction," 2003.

[34] R. E. Wood, "Task complexity: Definition of the construct," *Organizational Behavior and Human Decision Processes*, vol. 37, no. 1, pp. 60–82, 1986. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0749597886900440

[35] J. Park, W. Jung, and J. Ha, "Development of the step complexity measure for emergency operating procedures using entropy concepts," *Reliability Engineering System Safety*, vol. 71, no. 2, pp. 115–130, 2001.

[36] B. Crandall, G. A. Klein, and R. R. Hoffman, *Working minds: A practitioner's guide to cognitive task analysis.* Mit Press, 2006.

[37] N. A. Stanton, "Hierarchical task analysis: Developments, applications, and extensions," *Applied ergonomics*, vol. 37, no. 1, pp. 55–79, 2006.

[38] B. Schmidt, J. Kastl, T. Stoitsev, and M. Mühlhäuser, “Hierarchical task instance mining in interaction histories,” in *Proceedings of the 29th ACM International Conference on Design of Communication*, ser. SIGDOC ’11. New York, NY, USA: Association for Computing Machinery, 2011, p. 99–106. [Online]. Available: https://doi.org/10.1145/2038476.2038495

[39] Y. Zhang, Z. Li, B. Wu, and S. Wu, “A spaceflight operation complexity measure and its experimental validation,” *International Journal of Industrial Ergonomics*, vol. 39, no. 5, pp. 756–765, 2009.

[40] A. Garland, K. Ryall, and C. Rich, “Learning hierarchical task models by defining and refining examples,” in *Proceedings of the 1st International Conference on Knowledge Capture*, ser. K-CAP ’01. New York, NY, USA: Association for Computing Machinery, 2001, p. 44–51. [Online]. Available: https://doi.org/10.1145/500737.500748

[41] A. Garland and N. Lesh, “Learning hierarchical task models by demonstration,” *Mitsubishi Electric Research Laboratory (MERL), USA–(January 2002)*, pp. 51–79, 2003.

[42] M. G. Armentano and A. A. Amandi, “Modeling sequences of user actions for statistical goal recognition,” *User Modeling and User-Adapted Interaction*, vol. 22, no. 3, pp. 281–311, 2012.

[43] K. Leino, A. Oulasvirta, and M. Kurimo, “Rl-klm: automating keystroke-level modeling with reinforcement learning,” in *24rd International Conference on Intelligent User Interfaces*, 2019. [Online]. Available: http://users.comnet.aalto.fi/oulasvir/pubs/RL-KLM-IUI2019.pdf

[44] N. Usunier, G. Synnaeve, Z. Lin, and S. Chintala, “Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks,” *arXiv preprint arXiv:1609.02993*, 2016.

[45] C. R. Kube, “Task modelling in collective robotics,” *Autonomous robots*, vol. 4, no. 1, pp. 53–72, 1997.

[46] N. Koenig and M. J. Matarić, “Robot life-long task learning from human demonstrations: a bayesian approach,” *Autonomous Robots*, vol. 41, no. 5, pp. 1173–1188, 2017.

[47] A. Angelopoulou, “A simulation-based task analysis using agent-based, discrete event and system dynamics simulation,” 2015.

[48] A. Angelopoulou and K. Mykoniatis, “Utasimo: a simulation-based tool for task analysis,” *SIMULATION*, vol. 94, no. 1, pp. 43–54, 2018. [Online]. Available: https://doi.org/10.1177/0037549717711270

[49] K. Mykoniatis and A. Angelopoulou, “A modeling framework for the application of multi-paradigm simulation methods,” *SIMULATION*, vol. 96, no. 1, pp. 55–73, 2020. [Online]. Available: https://doi.org/10.1177/0037549719843339

[50] J. Park, W. Jung, J. Kim, and J. Ha, “Step complexity measure for emergency operating procedures - determining weighting factors,” *Nuclear Technology*, vol. 143, no. 3, pp. 290–308, 2003. [Online]. Available: https://doi.org/10.13182/NT03-A3418

[51] J. Park and W. Jung, “A study on the validity of a task complexity measure for emergency operating procedures of nuclear power plants—comparing with a subjective workload,” *IEEE Transactions on Nuclear Science*, vol. 53, no. 5, pp. 2962–2970, 2006.

[52] ——, “A study on the validity of a task complexity measure for emergency operating procedures of nuclear power plants—comparing task complexity scores with two sets of operator response time data obtained under a simulated sgtr,” *Reliability Engineering System Safety*, vol. 93, no. 4, pp. 557–566, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0951832007000543

[53] I. Jang, Y. Kim, and J. Park, “Investigating the effect of task complexity on the occurrence of human errors observed in a nuclear power plant full-scope simulator,” *Reliability Engineering & System Safety*, vol. 214, p. 107704, 2021.

[54] T. S. Chow, “Testing software design modeled by finite-state machines,” *IEEE Transactions on Software Engineering*, vol. SE-4, pp. 178–187, 1978.

[55] N. Walkinshaw, R. G. Taylor, and J. Derrick, “Inferring extended finite state machine models from software executions,” *Empirical Software Engineering*, vol. 21, pp. 811–853, 2013.

[56] J. Annett, “Hierarchical task analysis,” *Handbook of cognitive task design*, vol. 2, pp. 17–35, 2003.

[57] P. Liu and Z. Li, “Task complexity: A review and conceptualization framework,” *International Journal of Industrial Ergonomics*, vol. 42, no. 6, pp. 553–568, 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169814112000868

[58] A. Calero Valdez, P. Brauner, A. K. Schaar, A. Holzinger, and M. Ziefle, “Reducing complexity with simplicity-usability methods for industry 4.0,” in *Proceedings 19th Triennial Congress of the IEA*, TBC, Ed., vol. 9, 2015, p. 14.

[59] D.-H. Ham, J. Park, and W. Jung, “Model-based identification and use of task complexity factors of human integrated systems,” *Reliability Engineering System Safety*, vol. 100, pp. 33–47, 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0951832011002821

[60] P. M. Fitts, "The information capacity of the human motor system in controlling the amplitude of movement." *Journal of Experimental Psychology: General*, vol. 121, no. 3, pp. 262 – 269, 1992. [Online]. Available: http://search.ebscohost.com.libproxy.aalto.fi/login.aspx?direct=true&db=pdh&AN=1993-00286-001&site=ehost-live&authtype=sso&custid=ns192260

[61] Telia. (2021) Telia dot. [Online]. Available: https://www.telia.fi/dot

[62] S. Xu, Z. Li, F. Song, W. Luo, Q. Zhao, and G. Salvendy, "Influence of step complexity and presentation style on step performance of computerized emergency operating procedures," *Reliability Engineering System Safety*, vol. 94, no. 2, pp. 670–674, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0951832008001932

[63] T. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, 1976.

[64] A. Mowshowitz, "Entropy and the complexity of graphs: I. an index of the relative complexity of a graph," *The Bulletin of mathematical biophysics*, vol. 30, pp. 175–204, 04 1968.

[65] M. Design. (2021) Types of gestures. [Online]. Available: https://material.io/design/interaction/gestures.html#types-of-gestures

[66] J. S. Davis and R. J. LeBlanc, "A study of the applicability of complexity measures," *IEEE transactions on Software Engineering*, vol. 14, no. 9, pp. 1366–1372, 1988.

[67] Igraph. (2020) Igraph. [Online]. Available: https://igraph.org/

[68] P. Liu and Z. Li, "Toward understanding the relationship between task complexity and task performance," in *International Conference on Internationalization, Design and Global Development*. Springer, 2011, pp. 192–200.

[69] S. C. Seow, "Information theoretic models of hci: A comparison of the hick-hyman law and fitts' law," *Human–Computer Interaction*, vol. 20, no. 3, pp. 315–352, 2005. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1207/s15327051hci2003_3

[70] K. El Batran and M. D. Dunlop, "Enhancing klm (keystroke-level model) to fit touch screen mobile devices," in *Proceedings of the 16th International Conference on Human-Computer Interaction with Mobile Devices Services*, ser. MobileHCI '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 283–286. [Online]. Available: https://doi.org/10.1145/2628363.2628385

[71] S. G. Hart and L. E. Staveland, "Development of nasa-tlx (task load index): Results of empirical and theoretical research," in *Advances in psychology*. Elsevier, 1988, vol. 52, pp. 139–183.

[72] A. Oulasvirta, N. Ramesh Dayama, M. Shiripour, M. John, and A. Karrenbauer, "Combinatorial optimization of graphical user interface designs," *IEEE Proceedings*, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9000519

8 A Rico Ids With Errors

File	Reason	File	Reason
10498.json	'Toolbar'	44719.json	'Toolbar'
11148.json	'Drawer'	46019.json	division by zero
13721.json	'Drawer'	46146.json	division by zero
14310.json	division by zero	46528.json	division by zero
18262.json	'Drawer'	46609.json	division by zero
1996.json	'Drawer'	47883.json	division by zero
20891.json	division by zero	49228.json	'Drawer'
21186.json	'Toolbar'	50078.json	division by zero
21545.json	division by zero	50105.json	'bounds'
23689.json	division by zero	50109.json	'bounds'
24673.json	division by zero	50400.json	'Drawer'
24701.json	'Drawer'	52304.json	'Drawer'
26150.json	division by zero	53596.json	'Drawer'
26442.json	'Toolbar'	54046.json	math domain error
27046.json	'Drawer'	54392.json	'Toolbar'
29058.json	division by zero	54399.json	division by zero
30348.json	'Drawer'	54536.json	division by zero
3140.json	'Drawer'	57819.json	'Drawer'
33395.json	division by zero	5782.json	division by zero
34595.json	'Drawer'	5942.json	division by zero
35472.json	division by zero	59577.json	division by zero
35597.json	math domain error	62012.json	'Drawer'
36100.json	division by zero	64454.json	division by zero
36918.json	division by zero	6469.json	'Toolbar'
36999.json	'Toolbar'	65183.json	division by zero
37376.json	division by zero	68041.json	division by zero
39015.json	'Toolbar'	68580.json	division by zero
39118.json	division by zero	68871.json	'Toolbar'
39334.json	division by zero	7000.json	division by zero
3993.json	'Drawer'	7013.json	'Drawer'
41458.json	division by zero	70943.json	math domain error
41679.json	'Toolbar'	72071.json	division by zero
43387.json	division by zero	7953.json	division by zero
44706.json	'Toolbar'	8150.json	division by zero
44718.json	'Toolbar'	8522.json	'Toolbar'

Table 5: Rico view hierarchy files that caused errors when running the algorithm for generating graphs. A total of 70 view hierarchy files caused errors.

9 B Task Complexity Sourcecode

```
import pandas as pd
import json
from pandas.io.json import json_normalize
import igraph as igraph
import math
import numpy as np
import os as os
import csv
import random
import matplotlib.pyplot as plt

verticesMap = {"Advertisement": "AD",
               "Background␣Image": "BI",
               "Button␣Bar": "BB",
               "Checkbox": "C",
               "Card": "CC",
               "Date␣Picker": "DP",
               "Icon": "I",
               "Image": "IM",
               "Input": "IP",
               "List␣Item": "L",
               "Map␣View": "M",
               "On/Off␣Switch": "S",
               "Pager␣Indicator": "P",
               "Radio␣Button": "RB",
               "Text": "TC",
               "Text␣Button": "TB",
               "Web␣View": "WV"}

edgesMap = {"AD": "Tap",
            "BB": "Tap",
            "BI": "Tap",
            "C": "Tap",
            "CC": "Tap",
            "DP": "Multi",
            "I": "Tap",
            "IM": "Tap",
            "IP": "Input",
            "L": "Tap",
            "M": "Multi",
            "P": "Swipe",
            "RB": "Tap",
            "S": "Tap",
          "TB": "Tap",
          "TC": "Tap",
          "WV": "Tap"}

doNotEndTask = ["Checkbox", "Date␣Picker", "Input",
    "On/Off␣Switch"]

def getNormalizedBounds(dimensions, bounds):
    x1 = bounds[0] / dimensions[2]
    y1 = bounds[1] / dimensions[3]
    x2 = bounds[2] / dimensions[2]
    y2 = bounds[3] / dimensions[3]
    return [x1, y1, x2, y2]

def sortControlComponents(components):
    sortedComponents = []
    for component in components:
        bounds = component[1]
        componentPosition = 0
        for i in range(len(sortedComponents)):
            compareTo = sortedComponents[i]
            compareToBounds = compareTo[1]
            #If the new component is below the previous one
            #then we move
            #it to towards the end of component ordering
            if bounds[1] > compareToBounds[3]:
                componentPosition += 1
            #if we have overlap between the
            #components we choose by
            #the component ordering in horizontal plane
            elif bounds[0] > compareToBounds[2]:
                componentPosition += 1
            #otherwise we assume that the current
            #component is before the other component
        sortedComponents.insert(componentPosition, component)
    return sortedComponents


def getControlComponents(children):
    controlComponents = []
    def loopForControlComponents(children):
        for c in children:
            clickable = c['clickable']
            if(clickable):
                controlComponents.append((clickable, c['bounds'],
                                        c['componentLabel']))
            elif("children" in c):
                loopForControlComponents(c["children"])

    loopForControlComponents(children)
    ccCount = len(controlComponents)
    return controlComponents, ccCount

def defineAdjacencyMatrix(controlComponents, ccCount):
    adjacencyMatrix =  [[0] * (ccCount+2)]
    endVector = [0] * (ccCount+2)
    endpointsCounter = ccCount
    #this defines where the next nodes shall form connections
    currentAnchorColumn = 0
    componentLimit = ccCount - 1
    for i in range(ccCount):
            component = controlComponents[i]
            intermediateVector = []

            #special case for filling a login code
            #with automatic UI refresh
            #when Input field meets required length
            if(component[2] in doNotEndTask and i == componentLimit):
                intermediateVector = [0] * (ccCount+1)
                intermediateVector.insert(currentAnchorColumn, 1)
                currentAnchorColumn = 0
                endVector[i+1] = 1
            elif(component[2] in doNotEndTask):
                intermediateVector = [0] * (ccCount+1)
                intermediateVector.insert(currentAnchorColumn, 1)
                currentAnchorColumn = i + 1
                endVector[i+1] = 0
                endpointsCounter -= 1
            else:
                intermediateVector = [0] * (ccCount+1)
                intermediateVector.insert(currentAnchorColumn, 1)
                currentAnchorColumn = 0
                endVector[i+1] = 1
            adjacencyMatrix = adjacencyMatrix + [intermediateVector]
            #endVector[id+1] = 1
    adjacencyMatrix = adjacencyMatrix + [endVector]
    return np.array(adjacencyMatrix), endpointsCounter


def defineVertices(adjacencyMatrix, controlComponents):
    verticeLabels = ["E"]
    noEndpoints = pd.DataFrame(adjacencyMatrix)
    noEndpoints.drop(0,inplace=True)
    noEndpoints.drop(noEndpoints.tail(1).index,inplace=True)
    layersWithVertices = noEndpoints.sum(axis=0)
    layersWithVertices = layersWithVertices[layersWithVertices != 0]
    columnsWithVerticesIndexes = layersWithVertices.index
    currentComponentItem = 0
    for i in columnsWithVerticesIndexes:
        currentColumn = adjacencyMatrix[:, i]
        for row, c in enumerate(currentColumn):
            if c == 1:
                currentItem =
                    controlComponents[currentComponentItem][2]
                #These form the base level of the graph
                #and other components are inserted around these
                #to match the order required by igraph
                if(i == 0):
                    verticeLabels.append(verticesMap[currentItem])
                else:
                    verticeLabels.insert(row-1,
                        verticesMap[currentItem])
                currentComponentItem += 1

    verticeLabels = verticeLabels + ["S"]
    return verticeLabels

def defineEdges(adjacencyMatrix, verticeLabels, endpointCount):
    edgeLabels = []
    workedThroughGroupItems = 0
    for v in range(len(verticeLabels)-1):
        if(verticeLabels[v] in edgesMap):
            edgeLabels = edgeLabels + [edgesMap[verticeLabels[v]]]
        elif(verticeLabels[v] == "E"):
            edgeLabels = (["O"]*endpointCount) + edgeLabels

    return edgeLabels

def computeHalstead(edgeLabels, verticeLabels):
    #number of unique operators
    n1 = len(set(edgeLabels))
    #number of unique operands. We remove start and end
    n2 = len(set(verticeLabels))-2
    #total frequency of operators
    N1 = len(edgeLabels)
    #total frequency of operands. We remote start and end
    N2 = len(verticeLabels)-2

    firstPart = (n1*N2*(N1+N2))/(2*n2)
    logPart = math.log(n1+n2, 2)
    #rounded to S significant numbers
    return round(firstPart*logPart,3)

#we have omitted McCabe from this work due
#to difficulties with scaling
#McCabe might be a suitable measure in other contexts
def computeMcCabeVG(adjacencyMatrix):
    nodes = len(adjacencyMatrix)
    edges = adjacencyMatrix.sum().sum()
    return nodes - edges

#called from computeWoodEntropy()
def woodCoordinativeComplexity(adjacencyMatrix,
sortedControlComponents, screenDimensions):
    precedenceRelations = 0
    for i in range(len(sortedControlComponents)):
        cursor = i + 1
        currentAdjacencyRow = adjacencyMatrix[cursor]
        currentComponentIndex =
        np.where(currentAdjacencyRow==1)[0][0]
        if(currentComponentIndex == 0):
            continue
        while(currentComponentIndex > 0):
            precedenceRelations += 1
            cursor -= 1
            currentAdjacencyRow = adjacencyMatrix[cursor]
            currentComponentIndex =
                np.where(currentAdjacencyRow==1)[0][0]


def woodComponentComplexity(verticeLabels):
verticesCount = len(verticeLabels)
    return verticesCount

def computeWoodEntropy(adjacencyMatrix, sortedControlComponents,
screenDimensions, verticeLabels):
    coordinativeComplexity = woodCoordinativeComplexity(
        adjacencyMatrix, sortedControlComponents, screenDimensions)
    componentComplexity = woodComponentComplexity(verticeLabels)
    return coordinativeComplexity, componentComplexity
```

```
def getFile(fileName):
    with open("hierarchies/"+fileName, encoding="utf-8")
    as f:
        data = json.load(f)
        dimensions = data['bounds']
        children = data['children']
        return dimensions, children

def visualizeMatrix(adjacencyMatrix, verticeLabels,
    edgeLabels, fileName, manualTest=False):
    directed = ig.Graph.Adjacency(adjacencyMatrix, mode="directed")
    directed.vs["label"] = verticeLabels
    directed.vs["color"] = "white"
    directed.es["label"] = edgeLabels
    #directed.es["curved"] =
        seq(-0.5, 0.5, length = ecount(directed))
    directed.es["curved"] = False
    layout = directed.layout("rt")
    #save to file
    #return as a plot for a jupyter cell
    if(manualTest == True):
        return ig.plot(directed,  layout=layout,
            bbox=(0, 0, 350, 350), margin=20)

    ig.plot(directed, "graphs/"+fileName[:-5]+".png",
        layout=layout, bbox=(0, 0, 800, 800), margin=20)
    return


def determineControlGraphForFile(file):
    screenDimensions, children = getFile(file)
    controlComponents, ccCount = getControlComponents(children)
    #we need to sort control components as they are not always
    #in a sensible order from the perspective of how the actions
    #would be performed
    sortedControlComponents =
        sortControlComponents(controlComponents)
    adjacencyMatrix, endpointCount =
        defineAdjacencyMatrix(sortedControlComponents, ccCount)
    verticeLabels =
        defineVertices(adjacencyMatrix, sortedControlComponents)
    edgeLabels =
        defineEdges(adjacencyMatrix, verticeLabels, endpointCount)

    #complexity computation
    halstead = computeHalstead(edgeLabels, verticeLabels)
    mcCabe = computeMcCabeVG(adjacencyMatrix)
    woodCoordinative, woodComponent = computeWoodEntropy(
        adjacencyMatrix, sortedControlComponents,
        screenDimensions, verticeLabels)

    print("Halstead", halstead, "McCabe", mcCabe,
            "Wood␣coordinative", woodCoordinative,
          "Wood␣component", woodComponent, "Wood␣total",
          round(woodCoordinative+woodComponent,2))
    return [file, adjacencyMatrix, verticeLabels, edgeLabels,
        halstead,woodCoordinative, woodComponent, mcCabe]

def listAllFiles(folderPath):
    return os.listdir(folderPath)

def writeAnalyticsDataToFile(graphArrays):
    header = ['id', 'halstead', 'coordinative', 'component',
            'wood', 'wood_sqrt', 'paths', 'components']
    with open('./analytics.csv', 'w',
            encoding='UTF8', newline="") as f:
        writer = csv.writer(f)
        writer.writerow(header)
        for graphArray in graphArrays:
            paths = 0
            for row in graphArray[1]:
                if row[0] == 1:
                    paths +=1
            #file = graphArray[0]
            wood = graphArray[5]+graphArray[6]
            rowToWrite = [graphArray[0], graphArray[4],
                graphArray[5], graphArray[6],
                        round(wood,2)  , round(math.sqrt(wood),2),
                        paths, len(graphArray[2])-2]
            writer.writerow(rowToWrite)

def writeErrorsDataToFile(errors):
    with open('.errors.csv', 'w', encoding='UTF8', newline="")
    as f:
        writer = csv.writer(f)
        writer.writerow(['file', 'error'])
        writer.writerows(errors)


def loopAllJsonFiles(folderPath):
    allFiles = listAllFiles(folderPath)
    csvAnalytics = []
    errors = []
    for f in allFiles:
        try:
            controlGraphArray = determineControlGraphForFile(f)
            csvAnalytics.append(controlGraphArray)
            visualizeMatrix(controlGraphArray[1],
                controlGraphArray[2], controlGraphArray[3], f)
        except Exception as e:
            errors.append([f, e])
            continue
    writeAnalyticsDataToFile(csvAnalytics)
    writeErrorsDataToFile(errors)

loopAllJsonFiles("hierarchies")

results = pd.read_csv('analytics.csv')
print(results)

plt.figure(1, figsize=(16, 8))
plt.subplot(1,2,1)
plt.scatter(results['components'], results['halstead'])
plt.xlabel("Control␣components")
plt.ylabel("Halstead")
plt.subplot(1,2,2)
plt.scatter(results['paths'], results['halstead'])
plt.xlabel("Paths")
plt.ylabel("Halstead")

plt.figure(2, figsize=(16, 8))
plt.subplot(1,2,1)
plt.scatter(results['components'], results['wood'])
plt.xlabel("Control␣components")
plt.ylabel("Wood␣total")
plt.subplot(1,2,2)
plt.scatter(results['paths'], results['wood'])
plt.xlabel("Paths")
plt.ylabel("Wood␣total")

plt.figure(2, figsize=(16, 8))
plt.subplot(1,2,1)
plt.scatter(results['components'], results['coordinative'])
plt.xlabel("Control␣components")
plt.ylabel("Wood␣coordinative")
plt.subplot(1,2,2)
plt.scatter(results['paths'], results['coordinative'])
plt.xlabel("Paths")
plt.ylabel("Wood␣coordinative")

plt.figure(2, figsize=(16, 8))
plt.subplot(1,2,1)
plt.scatter(results['components'], results['component'])
plt.xlabel("Control␣components")
plt.ylabel("Wood␣component")
plt.subplot(1,2,2)
plt.scatter(results['paths'], results['component'])
plt.xlabel("Paths")
plt.ylabel("Wood␣component")

reRun = False
if reRun == False:
    raise Exception("Don't␣rerun␣the␣code␣block␣below␣by␣accident")

 #this is set to false so we don't accidentally reset the sample
def randomSampleForQA(checkFolder):
    if reRun == True:
        allFiles = listAllFiles(checkFolder)
        sample = random.sample(allFiles, k=30)
        for s in sample:
            print(s)

randomSampleForQA('screenshots')
```