Automated Estimation of Web Interaction Complexity based on UI Tests

Sebastian Heil[1][0000−0003−2761−9009], Julián Grigera[2,3][0000−0002−7962−4312], Ekaterina Pavlova[4][0009−0004−2464−6129], and Martin Gaedke[1][0000−0002−6729−2912]

[1] Technische Universität Chemnitz, 09111 Chemnitz, Germany
{sebastian.heil,martin.gaedke}@informatik.tu-chemnitz.de
[2] LIFIA, Fac. de Informatica, Univ. Nac. de La Plata, La Plata, Argentina
julian.grigera@lifia.info.unlp.edu.ar
[3] CONICET, Argentina
[4] Siemens Energy Global GmbH & Co. KG, Munich, Germany
pavekwa@gmail.com

Abstract. Evaluating the complexity of user interaction is crucial for engineering usable web application and to make informed design decisions. Automated approaches are increasingly being investigated due to the time, cost and expertise requirements of conducting empirical user studies. However, these approaches require creating extra artifacts otherwise not useful for software production, focus on static aspects of the user interface, or require specially instrumented running interface versions. Thus, we explore the feasibility of automated interaction complexity estimation through the analysis of existing web UI tests. We propose a novel software architecture for the analysis of UI tests and study the suitability of 11 different candidate UI test metrics from the domains of UI analysis, program understanding and cognitive modeling. By conducting an experiment with 38 participants, we compared human-delivered assessments to the UI test metrics. From this experiment, we obtained 2 linear and multiple linear regression models that can be used to predict interaction complexity. Additionally, we obtained a second set of multiple linear regression models based on a wider range of metrics, using Principal Component Analysis to narrow the input down to 3 components. These results suggest that automatically extracted UI test metrics can help determine interaction complexity, using existing artifacts. Our method is well-suited for integration into CI/CD pipelines, supporting the incremental and iterative nature of modern agile practices.

Keywords: Web User Interfaces · User Interaction · Complexity · IXD · UI · Automated Testing · UI Testing · Human-centric AI

1 Introduction

Measurement of interaction complexity is an important activity for engineering usable web application frontends and guiding design decisions. It serves as a

foundation for a systematic approach of iteratively building, evaluating, and improving user interfaces to control user cognitive effort [3]. Complexity is usually measured through user studies, but these require a high level of resources and methodological expertise. Therefore, research on automating complexity measurements has received increasing attention in academia [23, 8, 30, 21, 33]. Automation, further accelerated by advances in AI, allows for a high measurement frequency that leads to smaller increments being created more often, which is a key factor for integrating with agile software development methods [16].

However, current automation approaches exhibit several limitations. Approaches like KLM, FLM and GOMS are domain-specific languages (DSLs) that require creating artifacts to model the interaction based on which the automatic analysis is performed [9, 13, 11], but these artifacts have no intrinsic use in the software production. Other approaches only require existing artifacts, e.g. the UI source code [24, 23, 29, 31], screenshots of the UI [8, 30, 21], or a combination [33]. These approaches, however, only consider static properties of the user interface, mostly ignoring dynamic micro-interactions of users. Such interaction aspects can be automatically analyzed based on log analysis [1, 27] but, to create the logs, real users running special versions of the user interface are required. Not only is this semi-automatic approach therefore limited in applicability but also comes with legal privacy-related challenges of user tracking.

Ideally, an automation method for measuring web interaction complexity should not require involvement of users, make use of existing artifacts of web development only, and capture the dynamics of user interaction. Reduced costs spent on user evaluations, increased consistency, and a higher evaluation frequency that facilitates iterative frontend development cycles are the potential outcomes of lowering the barrier of automatic web UI assessments. UI tests are an artifact type that represents "human-like" [14] user interactions and serves its own purpose in professional software development. Additionally, UI tests are maintained alongside the UIs, so evaluations based on them also evolve, enabling the continuous monitoring of complexity within CI/CD cycles. Recent research has even demonstrated their automatic creation [14]. These characteristics make UI tests a valuable resource for iterative assessments of web interaction complexity without the need for additional artifacts.

In this paper we propose a novel automated approach leveraging UI tests to measure the interaction complexity of web user interfaces. We address the challenges of devising a suitable software architecture and the selection of suitable complexity-related UI test metrics. Our contributions are: a novel approach to repurpose UI tests for measuring web UI complexity, an empirical study with a diverse group of participants comparing human and automated measures, and a set of regression models for predicting human assessments of complexity.

2 Approach

Our proposal consists in automating the assessment of interaction complexity to simplify iterative web engineering processes. Thus, we propose a software

architecture – UIT-IC– that leverages source code and execution logs of UI tests (UITs) to estimate interaction complexity (IC) of the web UIs they cover.

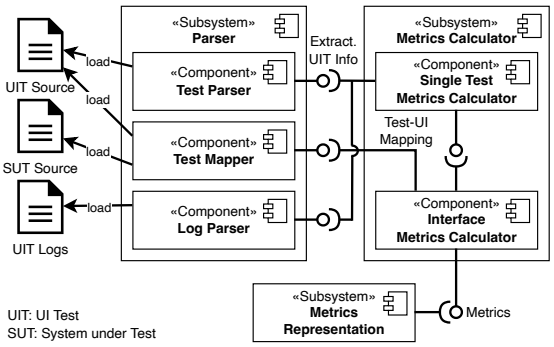

Fig. 1. UIT-IC Architecture.

From a software engineering perspective, our approach applies automatic software measurement techniques on UI tests to produce software metrics related to interaction complexity. From the perspective of HCI, these metrics are instrumentations that operationalize the latent interaction complexity construct.

2.1 Analysis Architecture and Artifacts

Figure 1 provides an overview of the main subsystems and components of the UIT-IC architecture and their interactions with each other: The `Parser` reads information from the UI tests and extracts data, which is then used by the `Metrics Calculator` to produce the output metrics per interface. Lastly, the `Metrics Representation` prepares a suitable representation for the integration of the analysis results into the calling system (e.g. a CI/CD pipeline). The UIT-IC architecture requires three input artifacts:

1. UI Tests source code, which represents the programming language-specific files containing the implementation of the UI tests using a UI testing framework like Selenium, Playwright or Cypress, used for static analysis,
2. System-under-Test (SUT) source code, required to match the information about the UI tests to the user interfaces which they are testing,
3. UI Tests execution logs, which are the textual representations of the framework-specific outputs produced when running the tests on the user interface and allow for dynamic analysis of timings etc.

The `Parser` subsystem extracts relevant information from these three input artifacts. Using the `Test Parser`, it identifies UI Tests, their location and metadata. The `Test mapper` is responsible for determining the corresponding user interface for each UI test analyzed. Execution logs information is extracted by the `Log Parser`, e.g. the time required to run a test.

The UI test information extracted by the `Parser` is the input for the computation of UI test metrics by the `Metrics Calculator`. This computation runs in two stages: first, metrics for individual UITs are computed by the `Single Test Metrics Calculator`. Usually, a user interface has more than one UIT representing various features and corner cases to be tested on it. Thus, in the second stage these results are then aggregated to metrics at the UI level by the `Interface Metrics Calculator`, using the Test-UI mapping previously derived by the `Test Mapper`. We detail the actual metrics computed at test and UI level in section 2.2. The final computation results are then passed to the `Metrics Representation` which transforms them into suitable format for the integration with the initiating system calling the analysis infrastructure. In the case of a CI/CD-pipeline, this is a formatted console output. For research purposes, the final representation uses a CSV-based tabular format.

2.2 Candidate Metrics

The `Metrics Calculator` features a plugin architecture that supports the definition of test and UI level metrics. This approach represents the realisation of a algorithmic/metric-model based method[5], in which stimuli affecting the human perception are mapped onto objectively computable quantities [2]. These *metrics* can be derived automatically through the analysis of software artifacts and need to be empirically assessed to demonstrate their correspondence with the actual subjective constructs – in our case interaction complexity. To the best of our knowledge, the idea to use UI tests for complexity analysis has not been investigated before. Thus, we had to identify potentially suitable metrics based on established theories and studies from three related research areas: UI analysis, program understanding and cognitive modeling.

UI analysis metrics related to complexity focus on measurable characteristics of the UI [2, 6, 26, 7]. As some of these UI characteristics are implicitly reflected in the UI tests, they served as inspiration for the first group of metrics in this study. UI analysis metrics studied comprise of UI elements count [2, 25, 7], navigation graph complexity [28], scrolls count [10], and user actions count [31, 5].

Program understanding metrics are traditional software metrics measured through static analysis of the source code structure or dynamically through code execution and monitoring runtime behavior which measure complexity with regard to readability, maintainability or development effort estimations. As UI tests represent software artifacts themselves, such metrics can be applied to indirectly measure complexity based on the assumption that complex interaction is reflected in increased software complexity of the UI tests. Program understanding metrics studied here are UIT SLOC [4, 32, 17], the counts of UIT statements, assert statements, wait statements [32, 22], and the test execution time [17].

Cognitive modeling metrics are measured using dedicated artifacts (cf. GOMS/ KLM [10]) that represent interactions and consider cognitive characteristics/ differences in the cognitive load caused by different types of actions. UI tests

[5] cf. Taxonomy proposed by Akca et al. [2]

are software artifacts that exhibit similarities to these models as they are also representations of interactions with user interfaces, so that cognitive modeling metrics can be applied in a similar way to UI tests. The two metrics studied are the weighted sum of user actions and the homing actions count [10]

2.3 Proof-of-Concept Implementation

While the UIT-IC architecture presented in Figure 1 is agnostic to specific technologies, we implemented a proof-of-concept as CLI application for easy integration with CI/CD pipelines using Kotlin and the Spring Boot framework. The implementation supports UI tests written in Selenium[6], due to its high popularity and the availability of more than 225 thousand repositories on GitHub using it. The implementation is part of our replication package[7].

3 Experimentation

To evaluate our approach, we conducted an experiment in which we compared the candidate complexity metrics calculated automatically against real user complexity assessments. We formulated this as a correlation study. The overall procedure consisted of analyzing the complexity of several WUIs using 2 sets of complexity measurements: those calculated by UIT-IC, and user-provided assessments. The overall design of the experiment is outlined in Figure 2.

3.1 Materials

The study was conducted on 4 different web applications, for a total 16 WUIs. Our selection was based on domain diversity, and UI tests availability. The 4 websites were the following:

- **Alf.IO:** a booking and scheduling application.
- **Lime Survey:** a popular platform for creating polls and events.
- **Zimbra:** a collaborative software suite.
- **OpenOLAT:** an online education platform.

For each of the 16 WUIs a task was defined. The selected tasks were typical interactions, like creating appointments, adjusting settings, or creating a test for a given course.

Since we aimed for a group of WUIs with diverse levels of complexity, we assessed each interface's complexity using a 3-level scale, as other existing studies [18, 15]. This ensured the experiment data has various complexity levels, so the potential differences can be noted in the resulting metrics. However, as this evaluation was made by hand, it can differ from the results of experiment participants and objective metrics. The list of applications and interfaces, along with extra information like the complexity estimation can be found in Table 1.

[6] `https://www.selenium.dev/`
[7] Replication package available at: `https://doi.org/10.5281/zenodo.18481281`

Table 1. Description of interfaces chosen for the experiment

Application	Category	Test Language	Interface	Number of Tests	Complexity
Alf.IO	Booking and scheduling	Java	Ticket purchase	1	low
Lime Survey	Polls and Events	PHP	Survey	2	medium
			Questions group	2	medium
			Labels	3	medium
Zimbra	Groupware	Java	Briefcase	75	low
			Task	97	low
			Email	194	low
			Calendar	371	medium
			Contact	167	low
			Preferences	84	medium
OpenOLAT	Learning and Courses	Java	Group	8	high
			Test	1	high
			Portfolio	7	high
			Resource folder	3	medium
			Course	12	high
			Question bank	3	high

Additionally, we created a survey for collecting the participants' assessments, as well as their demographic information. The survey also collected the time the participants took for answering each question. It was hosted in the LimeSurvey[8] platform (coincidentally, one of the assessed web applications).

3.2 Participants

A total of 38 participants took part of the study: 24 male, 14 female (average age 30.1, 7.8 SD). We applied the convenience sampling method, most of them students or recent graduates - 92.1% completed their Bachelor or higher. The self-denoted countries of residence are Germany (73.7%), Russia (7.9%), Argentina (7.9%), Romania (5.3%), India (2.6%) and Nepal (2.6%). The input devices used along the keyboard were mouse (52.6%) and touchpad (39.4%).

3.3 Experiment Design and Procedure

The study was conducted online, and participants self-moderated their assigned tasks. This design helped us obtain more participants in a shorter time period. Each participant had to provide 40 assessments on two of the four applications available (20 assessments each). This design ensured each interface received multiple independent assessments. The order in which both the applications and the tasks were provided was randomized, to avoid a potential learning effect.

Regarding the procedure, each participant first received some introductory information explaining the overall process and the collected data. At this point, participants were able to contact the researcher to ask any question about the study. This part was important since they were also informed that once started,

[8] https://www.limesurvey.org/

the study could not be paused. Once the participants were ready, they were provided with the link to start the evaluation, which started with the demographic survey. Then the main part of the study began, and they were presented with the tasks for the first application. After completing each task, they answered the assessment questions. This process was repeated for the second application. Finally, they were allowed to leave feedback on the study and the session was over. The expected completion time was expected to be between 15 and 20 minutes.

We designed the list of questions to gather the complexity assessments from the existing literature. We based our questionnaire on the 13-topic survey but Ling et al.[20], which showed good reliability and validity and was specifically adapted for the evaluation of interface complexity. Out of the original 13 categories (each with 2/4 statements), we selected 10 categories with 2 statements each. The criteria for this selection were minimizing repetition and simplifying formulations. The final survey includes statements related to locating information (e.g. "1.1 I know where to look to see the information I need"), interpreting visual structure (e.g. "2.2 The website uses too many different sizes, colors, fonts, and icons"), and handling task-related demands (e.g. "10.2 I feel overwhelmed by the amount of interaction required by the website"). Each of the 10 categories addresses a specific dimension of complexity, from visual and informational load to cognitive and action-related challenges.

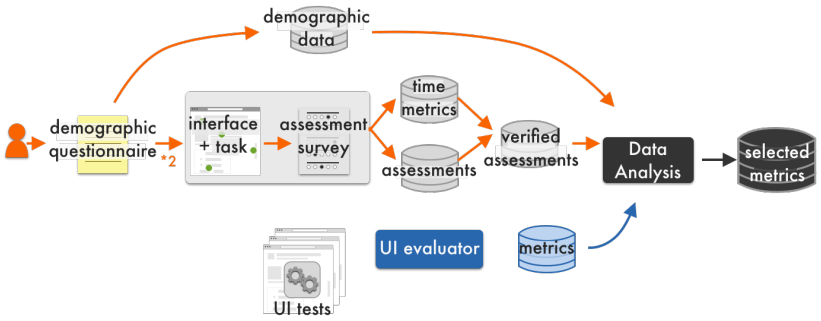

Fig. 2. Experiment design

3.4 Results

We gathered the results for all 4 applications, both **metrics** (using our tool) and participants' **assessments**. Regarding the metrics, we calculated 1 **visual complexity** metric, 2 **navigation** metrics, and 8 **task complexity** metrics, for a total of 11 candidate metrics.

Regarding the participants' results, we gathered the assessments for the 38 participants. We obtained 55 valid assessments out of 76. We discarded 17 incomplete ones, and 2 for taking too long according to our outlier analysis.

To compare the metrics with the assessments, we used the median across all participants, obtaining one value for each interface. We assessed the inter-rater reliability using the Kripendorff's alpha test, which is well suited for the Likert scale[9]. The results ultimately showed poor levels of agreement (discussed in Section 3.6), but we still proceeded with the medians for the analysis. We used the Spearman correlation coefficient to make the comparison, since the data was not normal. We calculated the correlation to examine the degree of the relationship between all measurements, namely 11 automated metrics and 20 user-provided assessments. Out of the 220 pairs of measurement correlations, 7 showed significant, strong negative correlations ($p < 0$, $p > 0.5$). The values for these 7 pairs are shown in Table 2. Others showed weak or no significant correlation ($p < 0.3$ or $p >= 0.05$). Despite the lack of significance in some cases, the limited dataset size ($n = 16$) suggests these metrics may still be valuable.

Table 2. Correlation coefficients (p-values) for user experience factors

	2.2 Many elements	**3.1** Busy site	**3.2** Adequate spacing	**4.2** Too much info	**8.2** Hard to find
Nav. graph complexity	**-0.58 (0.02)**	-0.38 (0.15)	-0.33 (0.21)	-0.02 (0.93)	0.05 (0.86)
Homing actions	-0.23 (0.39)	-0.34 (0.20)	0.04 (0.89)	**-0.51 (0.04)**	**-0.52 (0.04)**
Wait statements	**0.53 (0.03)**	**0.52 (0.04)**	**0.52 (0.04)**	0.36 (0.18)	0.23 (0.38)
Weighted actions sum	-0.48 (0.06)	**-0.51 (0.04)**	-0.22 (0.42)	-0.42 (0.10)	-0.46 (0.08)

Following the correlation study, we continued with regression analysis, since we are interested in predicting user-assessed complexity from the automated metrics. To this end, the 7 most correlated pairs presented earlier were analyzed. The two models showed the best results were (1) **"Average number of wait statements"** to **2.2 There are many elements** and (2) **"Average navigation complexity & Average number of wait statements"** to **"2.2 There are many elements"**. Both models are shown in the following equations:

$$y = 2.318 + 0.299 * x \tag{1}$$

$$y = 3.819 + 0.136 * x1 + -0.002 * x2 \tag{2}$$

The first model (equation (1)) showed significance $F = 16.473$, p-value $= 0.01$ and predictive power $R^2 = 0.541$; the second model (equation (2)) showed significance $F = 8.606$, p-value $= 0.04$ and $R^2 = 0.570$. The former is plotted on the chart in Figure 3, to show the difference between real and predicted values (we do not provide a scatterplot for the latter, since it is a multiple linear regression with two input variables).

[9] https://www.statisticshowto.com/krippendorffs-alpha/

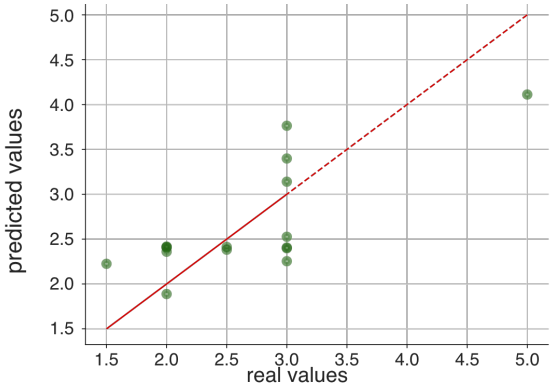

Fig. 3. Scatterplot illustrating the regression model for the metric "Average number of wait statements" and the assessment" 2.2 There are many elements".

Since the obtained regressions models are able to predict a limited number of subjective metrics, we intended to create a more comprehensive model using other objective metrics. However, the number of objective metrics is considerably large (11), so we tried to reduce this number by analyzing the relationships between them to select the most relevant ones. Again, we used the Spearman rank correlation to calculate the cross-correlations of the objective metrics. Table 3 shows the significant results of cross-correlation coefficient calculation, where strong correlation is emphasized with bold.

Table 3. Significant Spearman cross-correlation coefficients, p (p-value)

	Average number of UI elements	Average number of homing	Average number of scrolls	Average number of user actions	Average weighted user actions sum
Average number of UI elements	-	0.674 (0.004)	**0.843 (0.000)**	0.729 (0.001)	**0.924 (0.000)**
Average number of homing actions	0.674 (0.004)	-	0.774 (0.000)	**0.968 (0.000)**	0.639 (0.008)
Average number of scrolls	**0.843 (0.000)**	0.774 (0.000)	-	**0.849 (0.000)**	**0.888 (0.000)**
Average number of user actions	0.729 (0.001)	**0.968 (0.000)**	**0.849 (0.000)**	-	0.715 (0.002)
Average weighted user actions sum	**0.924 (0.000)**	0.639 (0.008)	**0.888 (0.000)**	0.715 (0.002)	-

Results allowed removing 3 metrics strongly correlated with others: "Average number of user actions", "Average number of scrolls" and "Average number of UI elements". We removed these instead of their correlated counterparts, since the latter built significant regression models with some of the user assessments.

After this analysis we narrowed the list down to 8 metrics, but we reduced it further using *Principal Component Analysis (PCA)*. To determine the number of components we used the broken stick method, which leverages the amount of variance each components explains, compared to what would be expected by chance. The plot (Figure 4) showed that three components sufficiently capture the variance in the data.

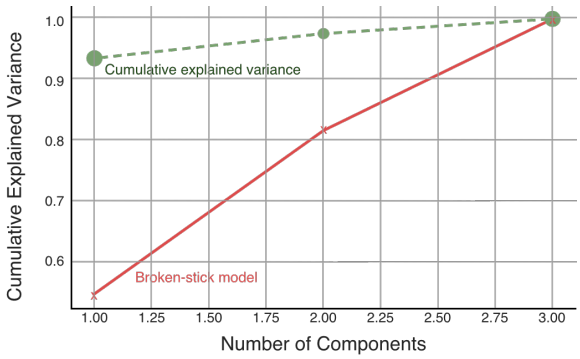

Fig. 4. PCA, finding the number of components with broken-stick method

The explained variance ratios for the components are 0.935419 for PC1, 0.038880 for PC2, and 0.024404 for PC3. Using these 3 components, we repeated the process of generating regression models with the subjective metrics. Some of the obtained models are displayed in Table 4.

Table 4. A sample of the regression models generated after the Principal Component Analysis results.

Metric	F-statistic	p-value	R^2	Model
2.2 There are many elements	2.500	0.109	0.385	$y = 2.656 + -0.002 * x1 + 0.007 * x2 + -0.021 * x3$
3.1 Website is busy	2.875	0.080	0.418	$y = 3.656 + -0.003 * x1 + 0.002 * x2 + -0.018 * x3$
4.1 Difficult information management	2.679	0.094	0.401	$y = 3.469 + -0.00 * 5x1 + 0.003 * x2 + -0.023 * x3$

3.5 Discussion

The results allowed to draw several insights. In the first place, some of the objective metrics (4) showed significant correlations with other subjective metrics. Two of the metrics, "Average number of wait statements" and "Average number of homing actions" also had more than one correlation to the subjective metrics. Regarding their interpretation, the metric "Average number of wait statements" captures waiting times in the UI tests, which are typically used to wait for elements to appear or react. This could explain correlations with metrics such as "2.2 There are many elements" and "3.1. Website is busy". The metric "Average number of homing actions" negatively correlated with subjective metrics about finding information, which is less expected, considering homing action is an additional effort and would be assumed to cause the interface to appear more confusing for the information-finding process and not vice-versa.

The cross-correlation of the objective metrics, part of which is shown in Table 3, produced some high coefficients. This can be explained by the way the metrics were captured. In the case of the most correlated ones, some similar markers of

UI interactions were used e.g. "Average number of scrolls per test" and "Average number of user actions per test".

Overall, the use of UI test analysis to assess interface complexity has demonstrated moderate effectiveness in predicting users' perceived complexity. Since UI tests primarily capture user interactions, the metrics related to interaction complexity showed stronger correlations with subjective user evaluations.

3.6 Threats to Validity

Our experiment was subject to some validity threats. Regarding external validity, demographic data mostly described a consistent group of young well-educated and computer-advanced participants. Therefore, their assessments might differ from those of the more general public, due to the more intensive use of complex websites, higher exposure to the Web in general, and specifically desktop UIs. Even if we could have aimed at more diverse backgrounds, we opted for the convenience method, since at this stage we prioritized the number of volunteers rather than their personal traits.

Additionally, the selected materials, such as web applications with their UI tests and corresponding tasks, may introduce threats to external validity due to under representation, and also a potential bias in the selection process (although that threat is more related to construct validity). To mitigate the first threat, the experiment conditions mirrored real-world scenarios: participants were unsupervised, used existing tools, and were not in a controlled environment. To decrease bias in material selection, applications were chosen from commonly used application categories, and the tasks represented typical interactions.

Regarding construct validity, the inter-rater agreement showed low values, which could be due to the low number of evaluations per interface and to the complexity of the questions. As some questions were more specific and technical in the sphere of interface design, participants may not have interpreted the concepts correctly, leading to more diverse scores. Even if we did not anticipate this effect, given the homogeneous group in terms of demographics, typical mitigation measures involve interacting with participants in additional sessions to improve consistency. This design would have compromised the number of participants in our evaluation, but should be considered for further experiments. Another mitigation measure for follow-up evaluations could consist in increasing the number of assessments *per interface*.

4 Related Work

In this section, we provide a brief overview on related work discussing background and recent research in the analysis of Interaction complexity and its automation.

4.1 Interaction Complexity and Metrics

There are many approaches that analyze UI complexity. Two well-known methods to predict user performance without requiring participants are KLM and

GOMS, even if they measure aspects related to complexity rather than complexity itself. The Keystroke-Level Model (KLM) [9] assigns an estimated time to each interactive component, providing a time estimate for the entire UI. Estimates are based on measured average execution times for the specific target user group and operators using the target application. GOMS (Goals, Operators, Methods, and Selection rules) [11] helps identifying the steps involved in achieving a goal and includes the mental processes, decision-making, and strategies used to perform a task (while KLM is purely focused on physical actions).

Another approach for predicting and comparing interaction complexity is the *Big I* notation[12], which is based on the Big O notation for algorithmic complexity. Big I expresses interaction complexity as a mathematical function, and it was designed for scenarios in which efficiency of use is critical. It is applied to interaction concepts, which can be useful even for early lo-fi prototypes.

4.2 Automated Analysis of Interaction Complexity

Visual clutter is analyzed to reduce the cognitive load of users using metrics such as feature congestion, subband entropy [30], or edge density [21]. Similarly, assessment of visual complexity [23, 33] aims at facilitating users' cognition of a WUI. The analysis result can be either a scalar value [33], or 3D representation of spatial complexity distribution similar to a heat map [23]. A comprehensive survey on methods for visual complexity analysis was conducted by Akça and Tanriöver [2]. In their proposed taxonomy [2], UIT-IC would belong to Metric-Model Based methods, however neither in the child group of GUI Component Oriented nor GUI Image Oriented. Instead of using UI components or image features of GUI screenshots, UIT-IC proposes the novel idea of using UI tests as the main input for complexity analysis.

In a work focused on mobile applications, Riegler and Holzmann [29] devised a tool to analyze screenshots and interaction data to quantify UI complexity using metrics of their own. They consider aspects like object separation or color consistency, while the user interaction data used is dwelling time. Even if dwelling time is a simple metric, this is one of the few works in the field to consider user interaction at all in UI complexity measurement.

5 Conclusion and Future Work

In this article, we proposed UIT-IC, a novel procedure for evaluating the complexity of web user interaction in an automated way. Our method uses existing UI tests, providing a key advantage over other methods since there is no need for additional artifacts. Leveraging existing UI tests also enables continuous evaluation of UI complexity as the tests evolve, making UIT-IC a practical addition to CI/CD pipelines. To the best of our knowledge, UI tests were not used before for the purpose of complexity assessment.

To develop a solution, we derived a set of candidate metrics extracted from different components of the UI tests cycle: source code, web application source

code and UI test execution logs. This allowed us to extract 11 metrics that can be used to predict complexity in aspects such as visual design, navigation, and interaction. These metrics were gathered and systematized based on existing research in the field of UI complexity assessment.

To explore the feasibility of our proposal, we conducted a study with 38 participants, in which we gathered their complexity assessments in 4 different web applications. These applications were also assessed by an implementation of UIT-IC that captured our automated metrics, and we analyzed correlations between them, in order to propose prediction models. Of all investigated regression models, 2 are shown to successfully predict human-rated complexity in terms of number of visual elements.

Finally, by applying different analysis techniques like cross-correlation and Principal Component Analysis (PCA) on the metrics, we were able to reduce the metrics down to 3 components while still keeping the variability in the data. We used these results to generate another set of simpler regression models still able to predict the human assessments of complexity.

In future work, we plan to investigate the applicability of UIT-IC for predicting task load, e.g. compared to NASA-TLX [19], or cognitive load, e.g. compared to tapping [3]. Another promising research direction is the combination of UIT-IC with methods for automated creation of UI tests [14] to investigate the impact on the reliability of metrics. Our current study was focused on Selenium-based UI tests. Future experiments can replicate the approach for other testing frameworks that are increasingly popular such as Playwright[10] or Cypress[11], which would provide insights on the testing platform-dependence of UIT-IC and also provide a wider basis for experimentation with public repositories. We are also currently experimenting with a variation of our approach that includes an intermediate step of mapping UI tests onto GOMS/KLM models. This direction, if successful, would allow to connect UIT-IC with established and validated cognitive modeling techniques.

Acknowledgments. This work is supported by the EU's HORIZON Research and Innovation Programme under grant agreement No 101120657, project ENFIELD.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Abb, L., Rehse, J.R.: Process-related user interaction logs: State of the art, reference model, and object-centric implementation. Information Systems **124**, 102386 (sep 2024). https://doi.org/10.1016/j.is.2024.102386
2. Akça, E., Ömer Özgür Tanriöver: A comprehensive appraisal of perceptual visual complexity analysis methods in gui design. Displays **69**, 102031 (9 2021)

[10] `https://playwright.dev`
[11] `https://www.cypress.io`

3. Albers, M.J.: Tapping as a measure of cognitive load and website usability. SIGDOC'11 - Proceedings of the 29th ACM International Conference on Design of Communication pp. 25–32 (2011). https://doi.org/10.1145/2038476.2038481
4. Albrecht, A., Gaffney, J.: Software function, source lines of code, and development effort prediction: A software science validation. IEEE Transactions on Software Engineering **SE-9**(6), 639–648 (1983). https://doi.org/10.1109/TSE.1983.235271
5. Alsmadi, I., Al-Kabi, M.: Gui structural metrics. The International Arab Journal of Information Technology **8**, 124–129 (4 2011)
6. Bakaev, M., Heil, S., Khvorostov, V., Gaedke, M.: In: Web Engineering: 18th International Conference, ICWE 2018. Springer. https://doi.org/10.1007/978-3-319-91662-0_10
7. Bakaev, M., Heil, S., Khvorostov, V., Gaedke, M.: Auto-extraction and integration of metrics for web user interfaces. Journal of Web Engineering **17**, 561–590 (2019). https://doi.org/10.13052/jwe1540-9589.17676
8. Bakaev, M., Speicher, M., Heil, S., Gaedke, M.: I don't have that much data! reusing user behavior models for websites from different domains. In: Bielikova, M., Mikkonen, T., Pautasso, C. (eds.) Web Engineering, Lecture Notes in Computer Science, vol. 12128, pp. 146–162. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-50578-3_11
9. Card, S.K., Moran, T.P., Newell, A.: The keystroke-level model for user performance time with interactive systems. Commun. ACM **23**(7), 396–410 (Jul 1980). https://doi.org/10.1145/358886.358895
10. Card, S.K., Moran, T.P., Newell, A.: The keystroke-level model for user performance time with interactive systems. Communications of the ACM **23**, 396–410 (7 1980). https://doi.org/10.1145/358886.358895
11. Card, S., Moran, T., Newell, A.: The model human processor- an engineering model of human performance. Handbook of perception and human performance. **2**(45–1), 1–35 (1986)
12. Degen, H.: Big i notation to estimate the interaction complexity of interaction concepts. International Journal of Human–Computer Interaction **38**(16), 1504–1528 (2022)
13. El Batran, K., Dunlop, M.D.: 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. pp. 283–286 (2014)
14. Fan, Y., Wang, S., Fei, Z., Qin, Y., Li, H., Liu, Y.: Can cooperative multi-agent reinforcement learning boost automatic web testing? an exploratory study. In: Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering. pp. 14–26. ACM, New York, NY, USA (oct 2024). https://doi.org/10.1145/3691620.3694983
15. Georges, V., Courtemanche, F., Sénécal, S., Baccino, T., Léger, P.M., Frédette, M.: Measuring visual complexity using neurophysiological data. In: Information Systems and Neuroscience: Gmunden Retreat on NeuroIS 2015. pp. 207–212. Springer (2015)
16. Grigera, J., Espada, J.P., Rossi, G.: Ai in user interface design and evaluation. IT Professional **25**(2), 20–22 (2023). https://doi.org/10.1109/MITP.2023.3267139
17. Hada, B.: Software testing metrics guide; definition, types & example (7 2023), `https://www.lambdatest.com/learning-hub/software-testing-metrics`
18. Harper, S., Michailidou, E., Stevens, R.: Toward a definition of visual complexity as an implicit measure of cognitive load. ACM Transactions on Applied Perception (TAP) **6**(2), 1–18 (2009)

19. Hart, S.G.: Nasa-Task Load Index (NASA-TLX); 20 Years Later. Proceedings of the Human Factors and Ergonomics Society Annual Meeting **50**(9), 904–908 (oct 2006). https://doi.org/10.1177/154193120605000909
20. Ling, C., Lopez, M., Shehab, R.: Complexity questionnaires of visual displays: A validation study of two information complexity questionnaires of visual displays. Human Factors and Ergonomics in Manufacturing & Service Industries **23**(5), 391–411 (2013)
21. Mack, M.L., Oliva, A.: Computational estimation of visual complexity. In: The 12th annual object, perception, attention, and memory conference (2004)
22. Magni, S., Ozcan, M.: Ui testing best practices (2024), `https://github.com/NoriSte/ui-testing-best-practices`
23. Michailidou, E., Eraslan, S., Yesilada, Y., Harper, S.: Automated prediction of visual complexity of web pages: Tools and evaluations. International Journal of Human-Computer Studies **145**, 102523 (jan 2021). https://doi.org/10.1016/j.ijhcs.2020.102523
24. Miniukovich, A., De Angeli, A.: Computation of interface aesthetics. In: Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems. pp. 1163–1172. ACM, New York, NY, USA (apr 2015). https://doi.org/10.1145/2702123.2702575
25. Miniukovich, A., Sulpizio, S., De Angeli, A.: Visual complexity of graphical user interfaces. In: Proceedings of the 2018 international conference on advanced visual interfaces. pp. 1–9 (2018)
26. Oulasvirta, A., et al.: Aalto Interface Metrics (AIM)): A Service and Codebase for Computational GUI Evaluation. In: The 31st Annual ACM Symposium on User Interface Software and Technology Adjunct Proceedings - UIST '18 Adjunct. pp. 16–19. ACM Press, New York, New York, USA (2018). https://doi.org/10.1145/3266037.3266087
27. Paganelli, L., Paternò, F.: Intelligent analysis of user interactions with web applications. In: Proceedings of the 7th international conference on Intelligent user interfaces. pp. 111–118. ACM, New York, NY, USA (jan 2002). https://doi.org/10.1145/502716.502735
28. Riegler, A., Holzmann, C.: Ui-cat: Calculating user interface complexity metrics for mobile applications. In: Proceedings of the 14th International Conference on Mobile and Ubiquitous Multimedia. vol. 30-November-2015, pp. 390–394. ACM (11 2015). https://doi.org/10.1145/2836041.2841214
29. Riegler, A., Holzmann, C.: Measuring visual user interface complexity of mobile applications with metrics. Interacting with Computers **30**(3), 207–223 (may 2018). https://doi.org/10.1093/iwc/iwy008
30. Rosenholtz, R., Li, Y., Nakano, L.: Measuring visual clutter. Journal of Vision **7**(2), 17 (aug 2007). https://doi.org/10.1167/7.2.17
31. Taba, S.E.S., Keivanloo, I., Zou, Y., Ng, J., Ng, T.: An exploratory study on the relation between user interface complexity and the perceived quality. In: Web Engineering: 14th International Conference, ICWE 2014, vol. 8541, pp. 370–379 (2014). https://doi.org/10.1007/978-3-319-08245-5_22
32. Weyuker, E.: Evaluating software complexity measures. IEEE Transactions on Software Engineering **14**(9), 1357–1365 (sep 1988)
33. Wu, O., Hu, W., Shi, L.: Measuring the visual complexities of web pages. ACM Transactions on the Web **7**(1), 1–34 (mar 2013). https://doi.org/10.1145/2435215.2435216