UIQLab: Automatic Web User Interface Assessment

Sebastian Heil[1][0000−0003−2761−9009], Calvin Liusnando[2][0009−0002−5757−9896], and Martin Gaedke[1][0000−0002−6729−2912]

[1] Technische Universität Chemnitz, 09111 Chemnitz, Germany
{sebastian.heil,martin.gaedke}@informatik.tu-chemnitz.de
[2] Staffbase SE, 09111 Chemnitz, Germany liusnandoc@gmail.com

Abstract. Assessment of Web User Interfaces is important activity for engineering usable frontends and guiding design decisions that can be automated to reduce expertise demands, costs and time. However, existing metrics and models are too scattered and prototypical to find wider adoption in industry practice and require consolidation. Even the most advanced assessment platforms are limited in their computational performance, handling of inputs and of results, and extensibility. By applying Web Engineering knowledge and principles, we devise a novel web-based platform that provides unified access to state-of-the art assessments and addresses the identified shortcomings of existing solutions. We address the technical challenges of improving the assessment performance and supporting extensibility to facilitate the integration of new assessment models. Our experiments with 14 assessment models in two different scenarios show that implementing such platform is feasible and that the assessment performance for multiple assessments can be improved by more than 55% through shared preprocessing and parallel computation.

Keywords: Web User Interfaces · User Interface Quality · Automated Software Engineering · Human-centered AI.

1 Introduction

Web User Interface (WUI) assessment is an important activity for engineering usable frontends and guiding design decisions. It serves as a foundation for a systematic approach of iteratively building, evaluating, and improving user interfaces of web applications, allowing to optimize them towards desired target characteristics in terms of aesthetics, complexity, and usability. Automatic approaches for WUI assessment [14,8,9,11,2] reduce the need for expensive and time-consuming empirical evaluations with human users. The automation allows for a high evaluation frequency and therefore for smaller increments being created more often, which is crucial when integrating with agile software development methods widely adopted for the creation of web applications. Various automatic approaches have been proposed to evaluate WUIs and, fueled by advances in Machine/Deep Learning, shown to be able to predict relevant characteristics [8,9,3,7,6,12].

Most of these proposed metrics and models, however, remain experimentally validated proofs of concept and automatic assessment of WUIs has yet to make the transfer from academia into industry practice [4]. The effort for practitioners to adopt the proposed automatic WUI assessment approaches in their development processes is high. Information about existing assessment possibilities is scattered in publications setting additional expertise requirements to employ them and the implementations are heterogeneous with regard to their technological platforms and maturity levels. This situation represents an obstacle for a wider adoption in the software industry, which has not received much attention in academia so far

Applying Web Engineering knowledge and principles, existing approaches can be consolidated into a web-based platform to lower the adoption barrier for industry practitioners. The benefits of such a platform lie in an improved guidance for selecting the suitable assessments and interpretation of results as well as in the convenience of a setup-free SaaS access to state-of-the-art WUI assessments. Even the most advanced assessment platform, AIM [17], so far has limitations in the ability to process multiple inputs, persistence of analysis results, computational performance, and extensibility, which impede wider adoption. Reduced costs spent on user evaluations, increased consistency, and a higher evaluation frequency facilitating iterative frontend development cycles are the potential outcomes of lowering the barrier of automatic WUI assessments.

The objective of this paper is to devise a novel web-based assessment platform that provides unified access to existing WUI metrics and models and addresses the identified shortcomings of existing solutions. We address the purpose of increasing the practicality of automatic WUI assessments by improving the performance of metrics computation and by enabling extensibility to facilitate the integration of new assessment models and propose solutions for each of the challenges. The platform architecture was implemented and put to test with existing automatic WUI assessments in two different experimental scenarios. To delimit our paper, our contributions are focused on the creation of the platform described above. Support for non-web user interfaces, definition of new metrics or the aggregation of assessment results into a single holistic value are out-of-scope. The remainder of this paper is structured as follows: in sect. 2 we review existing automatic WUI assessment approaches and platforms and identify current shortcomings, in 3 we outline our solution architecture, in 4 we evaluate its computational performance, and 5 concludes this paper with an outlook on directions for future work.

2 Related Work

2.1 Automated evaluation of user interfaces

There is a wide range of methodologies and software tools based on quantitative metrics for evaluating various aspects of WUIs. They are characterized by their use of specific, measurable, and often empirically validated parameters to objectively evaluate a certain quality such as usability, aesthetics, accessibility, or

visual complexity. To systematize the following overview, we structure the field in two orthogonal dimensions: by analysis input and result. *Static code analysis methods* [14,16,11] use the source code of the web user interface to calculate metrics related to layout and page structure, which enables prediction models for classification [11]. *Image-based analysis methods* [18,19,13,15,9,8,3,5] operationalize evaluation parameters based on the visual representation of the WUI – e.g. through screenshot analysis – using symbolic CV-based models or ML models. *Combined analysis methods* [14,21,2] process both code and visual information to compute the assessment. The analysis output and intended use of existing models for automated user interface evaluation is quite diverse. Some aim at providing a basis for classification ("good" vs "bad") [11] based on structural metrics like absolute numbers of words, links, fonts etc. and ratios of text/graphics.

Quantifying aesthetics is a major research focus in which various models [18,23,24] are based on the 14 fundamental measures introduced by Ngo et al. [16], representing the well-established Gestalt principles of design. Similar work [15] has introduced 8 aesthetics metrics taking into account psychological studies on perception. Image-based DNN-based models like Webthetics [8] have been shown to outperform previous models [12,18] in predicting subjective visual appeal ratings. The perceived colorfulness of a WUI [18] can be predicted based on a quantification of image colors in CIELab color space [10].

Predicting subjective user perceptions. Even ratings on custom subjective semantic differential scales can be predicted through NN-based models , allowing the application of Kansei Engineering to tailor WUI designs to achieve a set of desired target characteristics [2].Depending on the availability of training data, both traditional NN-models based on features like compression rate and information entropy as well as DNNs operating directly on the raw image data of user interface screenshots can predict subjective user perceptions [3,5].

Complexity. Visual clutter is analyzed to reduce the cognitive load of users using metrics such as feature congestion, subband entropy [19], or edge density [13]. Similarly, assessment of visual complexity [14,21] aims at facilitating users' cognition of a WUI. The analysis result can be either a scalar value [21], or 3D representation of spatial complexity distribution similar to a heat map [14].

Saliency. Predicting the spatial distribution of saliency [9,6,7] allows to identify areas in the user interface which attract users' attention and models typically produce a heatmap as output.

While the various approaches for automated evaluation of UIs demonstrate a large potential to support UI/UX related Web Engineering activities, they remain isolated and not easily accessible for practicioners. The majority of metrics and models are research prototypes and making use of them in professional settings would require extensive search, understanding and technical setup efforts.

2.2 Multi-metric Approaches and UI Evaluation Platforms

Several assessment tools support a more comprehensive evaluation computing multiple UI metrics. While some focus on metrics from a single assessment scope

like accessibility [20], or Aesthetics [23], there are also first attempts to consolidate assessments of usability, accessibility, aesthetics, and visual complexity in a single platform [1,17]. Web accessibility evaluation tools like WAVE [20] assess the accessibility of web user interfaces against guidelines such as WCAG or Section 508, typically providing summary reports as well as in-page highlighting of identified issues. QUESTIM (Quality Estimator using Metrics) [23] is a web-based tool allowing designers to define regions of interest in a user interface screenshot for which various aesthetics metrics are computed and represented using color gradients from green to red. Bakaev et al. proposed an integration platform architecture [1] for the assessment of web user interfaces which supports static code-based as well as image-based analysis models and integration of remote evaluation services, and collecting their results and allows designers to choose a subset of the available metrics for the current assessment.

The **Aalto Interface Metrics Service (AIM)**[3] [17] is the most comprehensive web-based UI evaluation platform available. It features 26 empirically validated assessments selectable by the user across 4 categories: 1 metric accessibility, 1 metric for aesthetics, 12 metrics for color perception, and 12 for perceptual fluency. The main assessment focus is on visual characteristics. Results are presented along explanations of the chosen metrics and an indication of the strength of empirical evidence for its predictive power. However, AIM exhibits some limitations with regard to handling multiple inputs, persistence of analysis results, computational performance, and extensibility, which we consider an impediment for wider adoption. It does not support running assessment of a selection of metrics on several inputs at once, which makes analysis of larger web systems and comparative analysis (cf. split testing) tedious. Once computed, assessment results are not persistently saved, requiring repeated execution for later review and causing efforts for sharing of results with other team members, who have to manually provide the same inputs, select the same metrics and wait for the computation again. The assessment performance is limited by sequential execution of the selected metrics. Performance deficits further stem from redundancies in preprocessing (e.g. CIELab color space conversion), except image segmentation, when several selected metrics repeatedly perform the same preprocessing tasks. While AIM supports extension, adding new metrics is not possible without changing the source code and re-deploying the platform.

In the following section, we therefore introduce a novel assessment platform for web user interfaces and apply Web Engineering principles and techniques to address the limitations prevalent in existing solutions, contributing to move the field of automated UI evaluation one step closer to a wider adoption.

3 The UIQLab Platform

In this section, we present our solution to empower Web Engineers to automatically assess the quality of user interfaces of the web systems they build. To

[3] https://interfacemetrics.aalto.fi

that end, we propose a novel software architecture – UIQLab– that addresses shortcomings of existing platforms with regard to handling several inputs, persistence of analysis results, performance, and the ability to extend the scope of the available assessment. Note that for brevity, we use the term metrics for WUI metrics and assessment models together.

Figure 1 provides an overview of the main components of the UIQLab platform and their interactions with each other. The `Frontend` is managed by the `Frontend Controller` and consists of three main components: Assessment UI, Results Dashboard, and Extension UI. The `Assessment UI` allows Web Engineers to specify multiple inputs, either by specifying URLs or via upload of their screenshots or source code. Available metrics are listed along with explanatory information and can be selected in arbitrary combinations to create an assessment request. The Web Engineer is then re-directed to the `Results Dashboard`, which asynchronously displays the analysis results as they are becoming available from the server, so that results of fast computations are displayed first and long-running ones later. The dashboard visualizes the results Employing a community-driven approach, new metrics can be proposed via the `Extension UI` by specifying their dependencies and metadata and uploading their implementations. These extension requests need to be approved by platform administrators to complete the integration in UIQLab. All frontend components implement a *guided interaction* pattern, leading platform users through the necessary steps.

In the `Backend`, assessment and extension requests are handled by the `Backend Controller` via a REST interface. It invokes the Metrics Evaluator to process assessment requests, which invokes the Preprocessor for the selected metrics. To enable the asynchronous display of results in the Results Dashboard, the Backend Controller immediately returns a response containing the identifier and type (URL, PNG, or HTML) of each WUI requested, and a unique URL as persistent identifier for the assessment results. This enables Web Engineers to share results of previously run assessments with other team members. The conceptual details of the Metrics Evaluator and Preprocessor as well as of the metrics programming model are described in sect. 3.1, 3.2, and 3.3 respectively.

Our proposed approach comprises solutions to three main challenges in the context of a web-based platform for automatic WUI assessment:

1. the asynchronous parallel execution of selected metrics,
2. the specification and resolution of preprocessing dependencies shared across metrics, and
3. the extension of the platform by new metrics.

The following subsections detail our solutions to these three challenges. While the proposed UIQLab architecture is technology-agnostic, our proof-of-concept implementation is based on python due to its wide support for computer vision and machine learning models employed in automated user interface assessment.

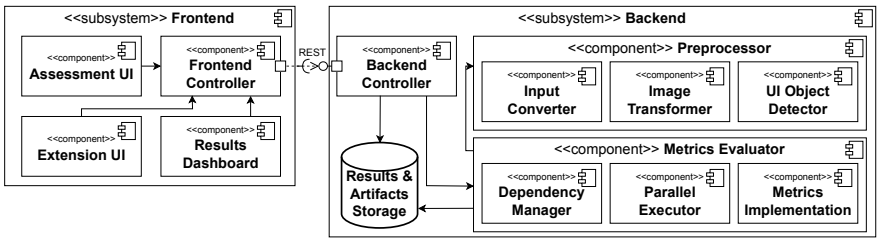

Fig. 1. Main Components and Interactions of the UIQLab platform.

3.1 Parallel Computation of Selected Metrics

Incoming assessment request are handled by the `Metrics Evaluator`. This comprises loading the metrics and executing them in parallel. In the following, we outline these two steps and further detail metrics implementations.

Metrics Loading. First, the Metrics Evaluator invokes the `Dependency Manager`. It has two responsibilities: 1) to dynamically load the implementations of the selected metrics, and 2) to request the necessary preprocessing tasks from the Preprocessor as in sect. 3.2. Dynamic loading of metrics is implemented as a lookup of the metric identifier in the metrics directory on the file system and supported through python importlib, allowing to import modules at runtime. Software dependencies are available a-priori as their installation is handled via the package manager (pip) when new metrics are accepted, cf. sect. 3.3.

Metrics Computation. The control flow then is passed to the `Parallel executor`. Process-based parallelism is used, where multiple processes execute the assessment tasks concurrently. This approach leverages multicore CPUs, avoiding the bottlenecks of thread-based parallelism, which shares a single memory space and is constrained by mechanisms like Python's Global Interpreter Lock, making it less suitable for the resource-intensive metrics computations. For each metric, a dedicated runner process is created and executed simultaneously to the others. The implementation uses a process pooling from the Pathos library due to its advanced serialization support for complex python objects such as the Pillow image objects resulting from image transformations. Once a computation completes, the result is stored in the results & artifacts storage, which supports both structured results like single or multiple numerical values as well as binary artifacts such as PNG files containing a heatmap as shown in fig. 2.

Metrics Implementations. The actual computations are defined through the metrics implementations, which are adhering to the interface specified in 3.3. As a proof of concept, we implemented 14 metrics from literature aiming to cover the range of input types, output types, methods and foci described in sect. 2. This selection does not claim quality or utility of the models. Table 1 provides an overview of these 14 metrics. **PNG file size** is the numerical value representing the size of a WUI screenshot image in bytes in PNG format with 24 bits per pixel, RGB color space. It is used for the analysis of visual complexity and aesthetics [15,5]. **JPEG file size and compression ratio** are derived from converting

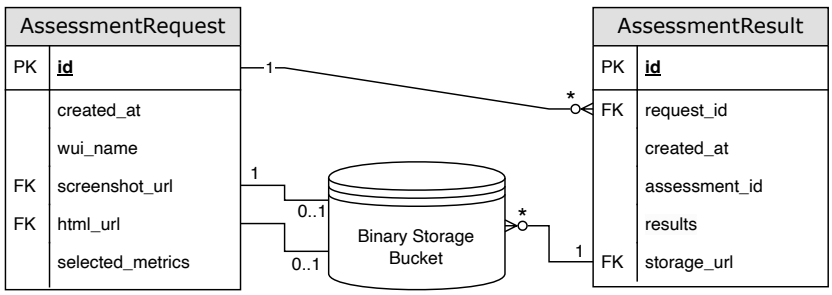

Fig. 2. Assessment Requests and Results Data Model

a WUI screenshot to JPEG (24 bits/pixel, RGB, Quality 0.8) and represented as a two-component vector, typically used for visual complexity and aesthetics [15,5]. **Colorfulness** is computed as scalar used in aesthetics analysis and computed taking into account the standard deviation and average range of colors in the RGB channel [10]. **CIELAB color average & standard deviation** similarly considers the color distribution in CIELAB space and is an example of a metric requiring an image transformation as pre-processing and producing a 6-tuple of 2 values per each of the three space dimensions. **Whitespace ratio** is computed as the share of white space of the overall screen estate of a WUI screenshot, used for aesthetics assessment [15]. It has a complex preprocessing dependency as it requires prior UI Object detection. Our implementation of this metric was adapted from AIM [17]. **UI Interpretation** is an example of a complex DNN-based assessment employing the UIED model [22]. It results in a JSON representing the identified bounding boxes and labels. **UMSI (Unified Model of Saliency and Importance)** is a predictive model of visual importance based on deep learning. It produces an image representing the heatmap of saliency and an overlaid version of the WUI screenshot. [9] **Word count** is a scalar value representing the number of visible words in the WUI, taking the DOM as input and is used as a predictive model for website ratings [11]. **Edge density** is a computer-vision based metric computed as the percentage of edge pixels in the WUI screenshot implemented as a Canny edge detector on grayscale images and used to assess visual complexity [19,13]. **Feature congestion** as measure of visual clutter [19] requires CIELAB conversion as preprocessing and is an example of an assessment producing two different types of result: a scalar representing the feature congestion value and an image visualizing the spattial distribution. Our implementation is adapted from AIM [17]. **Subband entropy** is a similar metric for visual clutter [19] based on computation of entropy in CIELAB color space, but producing only a single scalar value. **Shannon's information entropy** quantifies the amount of information in the WUI agnostic of colors, which requires grayscale conversion as preprocessing and resulting in a single numerical value. It is used for the assessment of subjective orderliness and aesthetics [5]. **Accessibility Analysis** is an assessment based on the aXe accessibility

checking engine operating on HTML input. It produces the number of violations found and a structured JSON output detailing these and was implemented using the selenium-axe-python library. **NIMA (Neural IMage Assessment)** uses a CNN to predict the distribution of human opinion scores of a WUI screenshot. It results in the numerical NIMA mean score, and the NIMA standard deviation score and was adapted from [17].

Table 1: Characteristics of proof-of-concept metrics

ID	Short Name	Input	Output	Preprocessing
01	PNG Size	Image	Numerical	N/A
02	JPEG Size/Compr.	Image	Vector	Format
03	Colorfulness	Image	Numerical	N/A
04	Color AVG/STD	Image	Vector	CIELAB
05	Whitespace	Image	Numerical	Object Detection
06	UI Interpretation	Image	JSON	N/A
07	UMSI	Image	Binary	N/A
08	Word Count	HTML	Numerical	DOM
09	Edges	Image	Numerical, Binary	Grayscale
10	Congestion	Image	Numerical, Binary	CIELAB
11	Subband Entropy	Image	Numerical	CIELAB
12	Shannon Entropy	Image	Numerical	Grayscale
13	Accessibility	HTML	JSON	N/A
14	NIMA	Image	Vector	N/A

3.2 Metrics Preprocessing Dependencies

In this subsection, we outline our concept of handling required preprocessing steps for the computation of selected metrics and improving assessment performance by avoiding redundant computations, and the currently available prepossessing components within the `Preprocessor`.

Resolving Metrics Preprocessing Dependencies. To improve efficiency of assessments, we propose to split computations into preprocessing tasks and the actual metrics. In this way, redundant computations can be reduced when several metrics share the same preprocessing requirements. Thus, the `Dependency Manager` identifies for each metric selected by the user the required preprocessing tasks in the metric's configuration (cf. 3.3). The tasks are added to a shared set of preprocessing for all selected metrics. Each preprocessing task in that set is executed only once, with the results shared to any metric implementation requiring them. UIQLab supports the most common preprocessing tasks used in automated WUI assessment models. Currently, this comprises of screenshot capture, dom analysis, image transformations and UI object detection.

Input Preprocessing Tasks. For user-provided inputs of type URL or HTML and selected metrics based on image analysis, the `Input Converter`

supports automatic screenshot capture, following the W3C's recommendation[4] using Selenium WebDriver. After the `document.readyState === "complete"` signal, it waits for an additional 5 seconds to balance a high probability of a completely rendered screenshot[5] and preventing excessive delays. In the same way, URL and HTML inputs for metrics requiring DOM access [14] are supported through DOM preprocessing. Screenshots are available as PNG, RGB color space, 2400x2400 px (window size = 1200x1200 px, device pixel ratio = 2) to accomodate even complex CV methods.

Image Preprocessing Tasks. Image-based analysis methods (cf. 2.1) often require image transformations to be applied before they can be computed. This comprises conversion to grayscale [18,13,5,19,1], image format conversions (e.g. to JPEG) [15,3,19,1,21,5], color space conversion (e.g. to CIELAB) [10,18,19] or image segmentation [21]/UI object detection [16,1,7]. Basic image transformations are performed by the `Image Transformer` using the CV libraries Pillow, OpenCV and scikit-image. UI object detection is provided by the `UI Object Detector`, implementing a version of UIED [22] adapted for Tesseract OCR and producing a list of bounding boxes and labels.

3.3 Extensible Metrics Interface

As the field of automated assessment of web user interfaces is rapidly evolving, it is important that the platform supports adding new assessment models. The architecture of UIQLab addresses this non-functional extensibility requirement through the specification of a metrics interface, the metrics metadata schema, and the extension process described in the following.

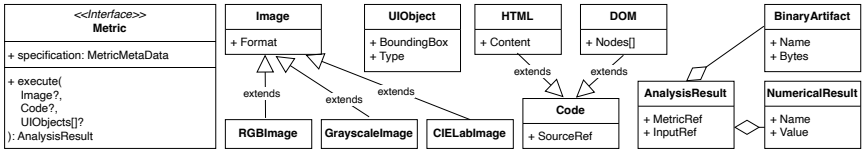

Fig. 3. Extensible Metrics Interface

Metrics Interface. In order to allow the execution of assessment models, their implementations must implement the UIQLab metrics interface shown in fig. 3. It requires the availability of a class method `execute` which is invoked when the assessment is run by the Parallel executor. The input parameters represent the different types of inputs supported by UIQLab: images in different formats and color spaces, the static HTML code, DOM or UI objects.

[4] cf. https://www.w3.org/TR/webdriver1/

[5] average load time on desktop is 2.5s, cf. https://www.tooltester.com/en/blog/website-loading-time-statistics/

Metrics Metadata Scheme. All metrics must provide a metadata description containing the information necessary for supporting an informed choice and interpretation of results for the user, and for the technical handling of inputs, preprocessing and outputs. These descriptions are represented as JSON following the UIQLab metadata scheme as in table 2.

Table 2: Metrics Metadata Scheme Properties

Field	Explanation
Name	Name of the metric
Description	Brief description to display to users
Input	Array of accepted input types, e.g. `url,html,png`, ...
Preprocessing	Array of required preprocessing identifiers, e.g. `grayscale,dom,cielab`, ...
References	Array of references to related scientific publications
Results	Array of results descriptions consisting of name, type and explanation for interpretation

Extension Process. To add new metrics to the plattform, the following extension process is defined. 1) The implementation needs to be prepared, adhering to the interface described above and described according to the above metadata scheme. 2) An extension request is created via the extension UI, uploading the required artifacts, namely a) the metrics implementation file, named according to the patter `{metric_id}_{metric_description}.py`, b) the metrics metadata description as JSON file `metric.json`, and c) the deployment dependencies as `requirements.txt` file. 4) Optionally, for required binary computational models (e.g. keras HDF5 files), the URLs can be provided, supporting deployment of implementations relying on corresponding model loaders. 5) The extension request is manually checked by the platform operator. 6) If approved, the new metric is deployed on the platform and available for subsequent assessments. The overall submit-approval process is similar to AIM, which uses GitHub's merge-request mechanism. However, UIQLab focuses on loose coupling via its defined extension interface, thus not requiring contributors to modify the platform's core itself and simplifying the addition of new metrics.

4 Evaluation

In this section, we present the experimental results from our evaluation of UIQLab. To demonstrate the scalability and showcase the impact of shared preprocessing and parallel metrics execution on computation times, we conducted a comparative performance analysis in two different scenarios, available in the replication package in the digital appendix[6] of this paper.

[6] https://vsr.informatik.tu-chemnitz.de/projects/2025/UIQLab

4.1 Method

Study Design and Rationale. The performance analysis employs a factorial design, analyzing the effect of the independent variable execution mode $M \in \{sequential, parallel\}$ on the computation time t in two different scenarios. The hypothesis tested is the difference between computation times of the two modes. This design aims at investigating the impact of the main architectural difference of UIQLab compared to existing approaches on performance. For a more detailed analysis, the two scenarios represent different levels of computational complexity of the metrics, allowing to draw conclusions for different types of metrics.

Material and Procedure. The experimental material comprises the running instance of UIQLab, two scenarios, and the instrumentation of the performance measurements. The measurements were run on a MacBook Pro (M1 Pro CPU with 10 cores (8 Firestorm and 2 Icestorm), 32 GB RAM, macOS 14.4). To enable precise measurements, the codebase was adapted for the performance experiments by disabling the storage of evaluation results, setting screenshot delay time to zero, and awaiting metric computations instead of spawning them as background tasks. cURL was used to measure the assessment request completion time, thus including the screenshot capture and preprocessing steps in our measurements. Our measurements compared sequential and parallel computation of a single WUI input in two scenarios: Scenario I represents assessment with fast to compute metrics (PNG Size, JPEG Size/Compression, Colofulness, Color AVG/STD, Edges, Shannon Entropy, NIMA), Scenario II represents assessment with moderate to slow metrics (Whitespace, UI Interpretation, UMSI, Word Count, Congestion, Subband Entropy, Accessibility).

The assignment of metrics to the two scenarios was based on single-metric requests of the platform's main UI itself for each metric, repeated 500 times, as shown in table 3. Based on this, metrics below an average of 5 seconds execution time were classified as fast to compute (Scenario I), others as medium to slow (Scenario II). Then, we ran Scenario I and II on the same input with sequential and with parallel execution and 500/100 repetitions respectively.

4.2 Results

Table 3 shows the measured average times $\bar{t}$ for single metric computation, ordered by increasing time and the separation between Scenario I and Scenario II metrics indicated as a horizontal line. All sample standard deviations s are well below 10% of the sample mean except for metric 04 (11.7%), as seen in the coefficients of variation $\widehat{c_{\mathrm{v}}} = \frac{s}{\bar{x}}$, indicating overall reliable measurements without large fluctuations. We tested for normality of the measured times using the Shapiro-Wilk test at $alpha = 0.05$ and Tukey Fence $k = 1.5$, which found metrics 03, 02 and 10 to be normal distributed. Additionally, based on the Kolmogorov-Smirnov effect sizes KS-d, the difference between the sample distributions and the normal distribution for all except metric 11 is small or very small, and can practically be assumed to be normal.

Figure 4 shows the results of the accumulated times of the 7 fast to compute metrics for scenario I and of the 7 medium-to-slow to compute metrics for scenario II. For **Scenario I**, surprisingly, the average execution time for sequential execution $\bar{t}_s = 3.66s$ is less than the time for parallel execution $\bar{t}_p = 4.38s$. As the measurements in Scenario I were not normal distributed (Shapiro-Wilk $p < 0.001$), we tested the difference using left-sided Mann-Whitney-U. The difference is found highly significant at $p < 0.001$ with a large effect of $Z/\sqrt{n1 + n2} = 0.84$ of the independent factor execution model on the dependent variable time. For **Scenario II**, the average execution times for sequential execution $\bar{t}_s = 151.9s$ is more than 1.5 times larger than the time for parallel execution $\bar{t}_p = 97.5s$. The right-sided Mann-Whitney-U test shows that this difference is highly significant at $p < 0.001$ and corresponds to a large effect size of 0.86 of the independent factor execution model on the dependent variable time.

Table 3: Assessment Times for Individual Metrics (increasing)

ID	**Short Name**	$\bar{t}$	s	$\widehat{c_v}$	SW-p	KS-d
01	PNG Size	2.41s	0.168s	0.07	0.0325	0.0498
03	Colorfulness	2.43s	0.185s	0.08	0.0807	
12	Shannon E.	2.44s	0.118s	0.05	0.0294	0.0450
02	JPEG Size/Compr.	2.46s	0.194s	0.08	0.1365	
09	Edges	2.66s	0.128s	0.05	<0.001	0.0579
14	NIMA	2.67s	0.123s	0.05	<0.001	0.0677
04	Color AVG/STD	3.05s	0.358s	0.12	0.0419	0.0470
05	Whitespace	5.42s	0.224s	0.04	0.0208	0.0516
06	UI Interpretation	5.38s	0.147s	0.03	<0.001	0.0556
08	Word Count	8.50s	0.281s	0.03	0.0072	0.0571
13	Accessibility	8.89s	0.328s	0.04	0.0034	0.0576
07	UMSI	11.55s	0.406s	0.04	0.0371	0.0380
11	Subband E.	42.34s	1.175s	0.03	<0.001	0.1887
10	Congestion	92.54s	0.781s	0.01	0.1292	

4.3 Discussion

Scenario I, representing the fast-to-compute metrics, showed a 19.6% better performance for sequential compared to parallel execution. A possible explanation for this observation is the low computation times for these metrics. In relation to the time for loading and rendering the UI from the URL input and taking the screenshot, the fast metrics' computation times are short. Thus, the overhead of parallel execution via process pools outweighed the time savings. This is an important finding, helping to better optimize future execution policies for different metrics. It indicates that the performance can further be improved through an algorithmic strategy that selectively chooses the computation mode based on the

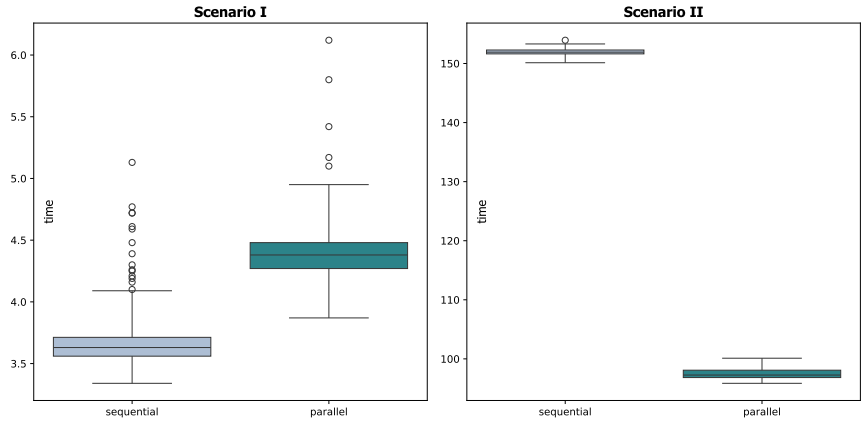

Fig. 4. Scenario Results

complexity of the selected metrics. In contrast, Scenario II showed improvements through shared preprocessing and parallel execution of 55.8%. In particular, the mean time for parallel mode of 97.5s is only 5% higher than the 92.5s of the slowest metric, 10 feature congestion. The 5% overhead reflects the necessary process pooling, as the 10 CPU cores were likely sufficient for parallel execution of the other metrics. These observations make a strong case for the advantage of the parallel computation of UIQLab for complex metrics, especially in the light of an increasing use of deep neural networks for UI assessments.

Threats to Validity. *Internal validity* can be threatened by the definition of the metrics for the two scenarios in our experiments. To address this, we decided the assignment of metrics based on the results of individual executions and the threshold shown in table 3. Another potential bias comes from the code adaptions described in sect. 4.1. All adaptions, however, are independent of metrics execution and apply to fast and medium-to-slow metrics and sequential and parallel execution mode alike. The reported times resulting from the adapted code are not used to argue about the expected platform performance itself. The reduced repetition count of 100 is due to feasibility for the long-running computations and the distributional statistics and significance tests indicate that this did not compromise the validity. All reported time measurements were automatically measured using curl without the potential for subjective biases. Furthermore, all evaluation materials are available on GitHub for replication. *External validity* of our experiments is limited by the choice of metrics and the hardware platform. The 14 implemented metrics were chosen from existing literature and representing a wide range of inputs, outputs, preprocessing requirements and implementation types as well as computational complexities. The experimental hardware affected the measured times. It represents a dedicated multi-core system to allow analysis of parallel execution. While our findings are valid for the relative comparison of execution modes, we are not making any claims towards the absolute values of

the measured times. In particular, we do not claim the times to be representative of the assessment times of the platform in production use. Generalization of our experimental results beyond the presented analysis, e.g. to other metrics, is not intended and would require dedicated experimentation. We are also not claiming that automated assessments could make subjective design choices more objective. *Construct validity* is threatened by our instrumentation of the time measurements. We used curl, a widely-used used tool, to automatically measure the entire round-trip-time for assessment requests. Potential impact of network conditions was mitigated by executing the experiments on localhost. The choice of measuring the entire assessment request time was made as it represents the delays experienced by users in the most direct way. We do not make specific numerical claims for the time measurements beyond.

5 Conclusion & Future Work

In this paper, we proposed a novel web-based WUI assessment platform that provides unified access to existing WUI metrics and models. It enables Web Engineers to automatically assess their user interfaces using 14 state-of-the-art models. Unlike previous existing platforms, it supports parallel computations, can handle multiple inputs, persists analysis results, and can be extended without changing the platform code. Based on extensive experimentation, we provided insights on the impact of the computation mode on performance, showcasing the advantage of parallel execution for medium-to-complex metrics. All implementation and experimental materials are provided to allow the Web Engineering community to replicate our experiments and extend the approach.

Future work can further improve the assessment performance through implementation of an algorithmic strategy for selecting sequential/parallel computation for subsets of selected metrics according to our identified preferences. Likewise, load balancing for the metrics computation could further improve performance in larger-scale deployments. A user study will provide insights on the usability of the UIQLab user interface. Aggregation of the individual assessment results into a holistic assessment for entire web applications as well as automatic remediation suggestions are promising directions for future research to which we plan to contribute first insights from currently ongoing experiments soon.

Acknowledgements This work is supported by the EU's HORIZON Research and Innovation Programme under grant agreement No 101120657, project ENFIELD (European Lighthouse to Manifest Trustworthy and Green AI).

References

1. Bakaev, M., Heil, S., Khvorostov, V., Gaedke, M.: Auto-Extraction and Integration of Metrics for Web User Interfaces. Journal of Web Engineering **17**(6), 561–590 (2019). https://doi.org/10.13052/jwe1540-9589.17676

2. Bakaev, M., Khvorostov, V., Heil, S., Gaedke, M.: Evaluation of User-Subjective Web Interface Similarity with Kansei Engineering-Based ANN. In: IEEE RE 2017 Workshops. pp. 125–131. IEEE (sep 2017). https://doi.org/10.1109/REW.2017.13
3. 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: ICWE 2020. LNCS, vol. 12128, pp. 146–162. Springer (2020). https://doi.org/10.1007/978-3-030-50578-3_11
4. Bakaev, M., Speicher, M., Jagow, J., Heil, S., Gaedke, M.: We Don't Need No Real Users?! Surveying the Adoption of User-less Automation Tools by UI Design Practitioners. In: ICWE 2022. LNCS, vol. 13362. pp. 406–414. Springer (2022). https://doi.org/10.1007/978-3-031-09917-5_28
5. Boychuk, E., Bakaev, M.: Entropy and compression based analysis of web user interfaces. In: ICWE 2019. LNCS vol. 11496. pp. 253–261. Springer (2019). https://doi.org/10.1007/978-3-030-19274-7_19
6. Bylinskii, Z., Kim, N.W., O'Donovan, P., Alsheikh, S., Madan, S., Pfister, H., Durand, F., Russell, B., Hertzmann, A.: Learning Visual Importance for Graphic Designs and Data Visualizations. In: Proceedings of UIST2017. pp. 57–69. ACM (oct 2017). https://doi.org/10.1145/3126594.3126653
7. Cornia, M., Baraldi, L., Serra, G., Cucchiara, R.: Predicting Human Eye Fixations via an LSTM-Based Saliency Attentive Model. IEEE Transactions on Image Processing **27**(10), 5142–5154 (oct 2018). https://doi.org/10.1109/TIP.2018.2851672
8. Dou, Q., Zheng, X.S., Sun, T., Heng, P.A.: Webthetics: Quantifying webpage aesthetics with deep learning. International Journal of Human-Computer Studies **124**, 56–66 (apr 2019). https://doi.org/10.1016/J.IJHCS.2018.11.006
9. Fosco, C., Casser, V., Bedi, A.K., O'Donovan, P., Hertzmann, A., Bylinskii, Z.: Predicting Visual Importance Across Graphic Design Types. In: Proceedings of UIST2020. pp. 249–260. ACM, New York, NY, USA (oct 2020). https://doi.org/10.1145/3379337.3415825
10. Hasler, D., Suesstrunk, S.E.: Measuring colorfulness in natural images. In: Rogowitz, B.E., Pappas, T.N. (eds.) Proc. IST/SPIE Electronic Imaging 2003: Human Vision and Electronic Imaging VIII. vol. 5007, p. 87 (jun 2003). https://doi.org/10.1117/12.477378
11. Ivory, M.Y., Sinha, R.R., Hearst, M.A.: Empirically validated web page design metrics. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 53–60. ACM, New York, NY, USA (mar 2001). https://doi.org/10.1145/365024.365035
12. Khani, M.G., Mazinani, M.R., Fayyaz, M., Hoseini, M.: A novel approach for website aesthetic evaluation based on convolutional neural networks. In: 2016 Second International Conference on Web Research (ICWR). pp. 48–53. IEEE (apr 2016). https://doi.org/10.1109/ICWR.2016.7498445
13. Mack, M.L., Oliva, A.: Computational estimation of visual complexity. In: The 12th annual object, perception, attention, and memory conference (2004)
14. 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
15. 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

16. Ngo, D.C.L., Teo, L.S., Byrne, J.G.: Modelling interface aesthetics. Information Sciences **152**, 25–46 (jun 2003). https://doi.org/10.1016/S0020-0255(02)00404-8
17. Oulasvirta, A., Miniukovich, A., Palmas, G., Weinkauf, T., De Pascale, S., Koch, J., Langerak, T., Jokinen, J., Todi, K., Laine, M., Kristhombuge, M., Zhu, Y.: Aalto Interface Metrics (AIM)): A Service and Codebase for Computational GUI Evaluation. In: Proc. of UIST2018 Adjunct. pp. 16–19. ACM Press, New York, New York, USA (2018). https://doi.org/10.1145/3266037.3266087
18. Reinecke, K., Yeh, T., Miratrix, L., Mardiko, R., Zhao, Y., Liu, J., Gajos, K.Z.: Predicting users' first impressions of website aesthetics with a quantification of perceived visual complexity and colorfulness. In: Proceedings of CHI2013. pp. 2049–2058. ACM, New York, NY, USA (apr 2013). https://doi.org/10.1145/2470654.2481281
19. 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
20. WebAIM at Utah State University: WAVE Web Accessibility Evaluation Tools (2001), https://wave.webaim.org/
21. 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
22. Xie, M., Feng, S., Xing, Z., Chen, J., Chen, C.: UIED: a hybrid tool for GUI element detection. In: ACM FSE 2020. pp. 1655–1659. ACM, New York, NY, USA (nov 2020). https://doi.org/10.1145/3368089.3417940
23. Zen, M.: Metric-based evaluation of graphical user interfaces: Model, Method, and Software Support. In: ACM EICS 2013. pp. 183–186. ACM, New York, NY, USA (jun 2013). https://doi.org/10.1145/2494603.2480331
24. Zen, M., Vanderdonckt, J.: Assessing User Interface Aesthetics based on the Intersubjectivity of Judgment. In: Proceedings of the 30th International BCS Human Computer Interaction Conference, HCI 2016. vol. 2016-July, pp. 1–12 (2016). https://doi.org/10.14236/ewic/HCI2016.25