A New Approach to Visualise Accessibility Problems of Mobile Apps in Source Code

Johannes Feiner[1], Elmar Krainz[1,2] and Keith Andrews[3]
[1]*Institute of Internet Technologies & Applications, FH-JOANNEUM, Werk-VI-Straße 46b, 8605 Kapfenberg, Austria*
[2]*IIS, Kepler University Linz, Altenberger Straße 69, 4040 Linz, Austria*
[3] *Institute of Interactive Systems and Data Science, Graz University of Technology, Inffeldgasse 16c, 8010 Graz, Austria*
johannes.feiner@fh-joanneum.at, elmar.krainz@fh-joanneum.at, kandrews@iicm.edu

Keywords: Accessibility, Code Visualisation, Usability.

Abstract: A wide range of software development is moving to the direction and domain of mobile applications. Single developer or small teams create apps for smartphones. Too often, they have not the capacity or know-how to check for usability problems and do not care for accessibility. We propose a novel workflow to bring usability issues into the development process: A quick accessibility evaluation (QAC) with 15 predefined metrics allow to collect issues. These issues are further condensed into formalised (UsabML) and the issues are tagged with the location in the source code. A dashboard view (RepoVis) showing the source code from a repository allows to spot and interactively inspect code and related issues simultaneously.

1 INTRODUCTION

The aim of software engineers is to create high quality software products including readable, structured source code and error free programs. From the user's perspective, the most important component is the perceivable part – the user interface (UI).

The quality of the user interface is defined by usability attributes, like effectiveness, efficiency and satisfaction (ISO, 2000). But, many people cannot even experience good or bad usability, in cases where the software is not accessible at all.

1.1 Problem

The UN Convention on the Rights of Persons with Disabilities (CRPD) (The United Nations, 2006) reveals that about 760 million people, i.e. 10% of the world population, are handicapped. Therefore, accessibility (A11Y) is a significant issue on software user interfaces. The ISO Standard 9241-171:2008 (ISO, 2008) defines accessibility as: *interactive system usability of a product, service, environment or facility by people with the widest range of capabilities*. This means the ability for anyone to understand and operate a software product.

Usability and accessibility are not built-in features. They require awareness of developers. To improve usability and accessibility three steps are needed in the development process:

1. Finding errors in the user interface
2. Mapping error to code
3. Improving the code

1.2 Approach

The paper at hand shows a novel approach to combine accessibility evaluation and reporting with the visualisation of reported bugs along with the source code. Following the recommendation for good research in Software Engineering (Shaw, 2002) we set up the following research questions:

- *RQ1* Which methods would help software developers to evaluate accessibility of mobile apps in a quick and structured way?
- *RQ2* How to visually present accessibility issues of mobile android apps along with corresponding source code?

To answers this questions we proceeded in following way: Fist we analysed the cyclic work flows of mobile application developers when they are improving apps; how do they extract relevant aspects from usability reports. Then we condensed this information gathering flow into a new workflow, which can

Feiner, J., Krainz, E. and Andrews, K.
A New Approach to Visualise Accessibility Problems of Mobile Apps in Source Code.
In *Proceedings of the 20th International Conference on Enterprise Information Systems (ICEIS 2018) - Volume 2*, pages 519-526
ISBN: 978-989-758-298-1

better support improvements concerning accessibility. Finally, some tools for software engineers were added (compare Sections 3.1 *Quick Accessibility Check* QAC, 3.2 *Usability Markup Lanaguage* UsabML and *Repository Visualisation* 3.3 RepoVis) to enhance the overall efficiency. The tools support comprehension of existing A11Y problems and guide during the implementation of usability improvements.

Our efforts resulted in an heuristic approach to evaluate app accessibility with the newly designed tool QAC. This includes several thoughtfully selected predefined evaluation criteria and partial support by existing tools. The reports are processed into a structured reporting format, compare previous research on UsabML (Feiner et al., 2010), and the usability problems found are linked to code locations. Furthermore, the developers can take advantage of the new tool RepoVis implemented by the authors to view source code and accessibility issues side-by-side.

To validate the novel workflow of detecting accessibility issues (RQ1) and reporting them back to developers in a visual way (RQ2) for fixing the flaws we performed an experiment with two selected mobile apps from the Google Play store.

1.3 Hypothesis

The main hypothesis we build upon is that an alternative – above all simpler – approach would improve awareness and software quality in terms of accessibility. The main aspects of our new approach can be summarised as follows:

- For quick, nevertheless useful results, a UX team is not used in any case. Small developer teams or single developers can perform a quick accessibility check (QAC) on their own.
- The integration of two domains, namely usability and software engineering is relevant. This way it is possible to connect user experience (UX), accessibility (A11Y) and source code which raises acceptance of the workflows and tools engaged.
- A dashboard overview supports faster comprehension of source code and related usability evaluation results. Combined views are helpful, because there is no need for retrieving information from different tools and data-sources manually.

The remainder of this paper is structured in the following way. In Section 2 alternative approaches are listed, in Section 3 the suggested workflow is described in detail and in Section 4 the performed evaluations of two Android apps from the Google Play store are presented.

2 RELATED WORK

Accessibility is an important factor to provide services and information to the vast majority of people. For example the European Union enacted in 2016 the directive 2016/2102 which *..aims to ensure that the websites and mobile applications of public sector bodies are made more accessible on the basis of common accessibility requirements...* (European Union, 2016).

The definitions and the principles defined in the accessibility standards provide the basis of accessible user interfaces. The Web Accessibility Initiative (WAI) offers guidelines for developers about proper implementation of websites and mobile applications. The Web Content Accessibility Guidelines (WCAG 2.0) constitute a common standard, which provides generic principles for accessible development. WCAG 2.0 outlines four principles of accessibility:

1. *Perceivable*: Information and user interface components must be presentable to users in ways they can perceive. For example one might provide an alternative text to an image.
2. *Operable*: User interface components and navigation must be operable, which means, for example, not limited to mouse usage.
3. *Understandable*: Information and the operation of user interface must be understandable in terms of readability or predictability of navigation.
4. *Robustness*: Content must be robust enough that it can be interpreted reliably by a wide variety of user agents, including assistive technologies.

Mobile operating systems like iOS and Android have their own platform guidelines for developers. Compare the design guidelines for Android (Google, 2017) and for iOS (Apple, 2012). They follow the WAI principles and have many aspects in common, but also some varieties in the implementation for the platforms assistive technologies.

Other relevant rulesets are the *Accessible Tool Authoring Tool* ATAG 2.0 (Treviranus et al., 2015) and the *User Agent Accessibility Guidelines* UAAG 2.0 (Patch et al., 2015a). The ATAG 2.0 provide guidance to make the software and services for web content creation accessible. The accessibility of browsers, media players and the interface to assistive technologies is recommended by the UAAG 2.0. All these guidelines are the foundation for development and for the assessment of accessibility.

The evaluation can be done either with user testing or in a more formal way with experts using those guidelines. The expert evaluation method is related to

the *Heuristic Evaluation* (Nielsen and Molich, 1990), but contrary to the intuitive usability heuristics, using the accessibility guidelines needs more background know-how and much more efforts even for experts.

Inspired by the idea of *Guerrilla HCI* (Nielsen, 1994) to simplify the usability engineering process and the *WAI – Easy Checks* (Henry, 2013), a method for a first accessibility review, this paper proposes a quick method to get an overview of the most significant accessibility problems of a mobile application (see Section 3).

The data acquisition during usability testing happens sometimes on paper (Vilbergsdóttir et al., 2006) according given schemes and sometimes with support of various electronic tools (Hvannberg et al., 2007; Andrews, 2008). The collected raw data is further condensed into a written report. Comparability (Molich et al., 2010) of evaluation results, as well as reusability of reported data, is still under research (Cheng, 2016). In most cases the results of usability evaluations are available to managers and software developers in printed formats or as unstructured electronic documents such as PDF. Other researchers suggest a standard method (ISO, 2006; Komiyama, 2008) for reporting usability test findings. *UsabML* (Feiner et al., 2010; Feiner and Andrews, 2012) introduces an XML structured approach to work with evaluation results. Through formalisation it is possible to programmatically process collected data. For example, issues might be pushed into bug tracking systems automatically.

A well-designed feedback loop is vital in modern user centred design (UCD) to enable developers to react on the results of an review. Usability defect reporting should address the needs of developers (Yusop et al., 2016).

Software engineers prefer the evaluation results presented within their existing toolchains. That means, the acceptance of usability suggestions is better when, for example, an accessibility issue is presented as bug in a bug tracker or with tools combining and linking the different sources of code and evaluation data. To fix bugs, code comprehension is vital. For code comprehension (Hawes et al., 2015) of larger software systems various mappings could be used. Novel approaches, such as statistical language models (Murphy, 2016) should improve software comprehension furthermore. Among many different ways for graphical representation, the SeeSoft (Eick et al., 1992) software visualisation is special in displaying all files at the same time by mapping the lines of source code to coloured pixels according to selected software metrics. Based on the SeeSoft idea, the authors introduce *RepoVis*, which tries to visualise the issues side-by-side to source code as discussed later in the paper in Section 3.

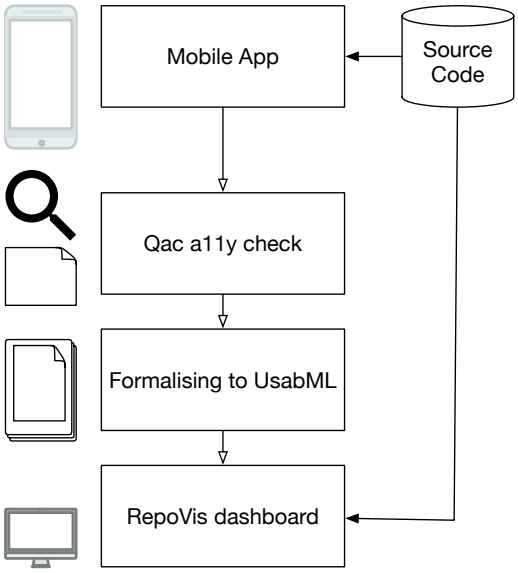

Figure 1: The novel workflow of A11Y inspection, formalising the results and combined issue code visualisation.

3 A NOVEL WORKFLOW

The questions of effective usability evaluation and reporting of mobile smartphone apps is addressed by the design of a novel workflow. This should enable developers of small development teams – in cases where no usability evaluation experts are available – to create software which takes accessibility into account.

The workflow is depicted in Figure 1. It explains the stages from checking accessibility of an app up to presenting results to developers.

3.1 Quick Accessibility Check - QAC

To integrate accessibility evaluation in the development process one needs a method which is (a) applicable to developers and testers without the deep background knowledge of usability and accessibility evaluation and (b) provides measurable result to rate and compare the improvements in the user interface.

The quick accessibility check (QAC) uses 15 criteria in 3 sections to measure and compare the accessibility of an user interface. These heuristics are based/are an intersection of the *Material Accessibility Design Guidelines* (Google, 2017), the *Web Content Accessibility Guidelines* (Caldwell et al., 2008) as the most significant standard for accessibility evaluation

Table 1: Quick Accessibility Check (QAC) with 15 questions grouped into three sections.

QAC-Section	No.	Checking
Assistive Android	1	Screenreader
-"-	2	Tabbing
-"-	3	External Keyboard
A11Y Scanner	4	Touch Size
-"-	5	Contrast Images
-"-	6	Contrast Text
-"-	7	Missing Labels
-"-	8	Redundant Labels
-"-	9	Implementation
Manual Checks	10	Text: Clear & Concise
-"-	11	Evident Navigation
-"-	12	Font Size
-"-	13	Support for Zooming
-"-	14	Position of Elements
-"-	15	Colour-blindness

and *W3C Mobile Accessibility for Mobile* (Patch et al., 2015b).

The first section *Assistive Android* gives feedback about the support for assistive technologies like the screen reader or an external keyboard. In a scale from 1 (best) to 5 (worst) these topics are evaluated with assistive technology (e.g. Talkback on Android device) enabled.

The second section *A11Y Scanner* covers automatic testable features like contrast of texts and images, suitable labels and descriptions of non-text elements and the minimum size of touchable elements. On the Android platform a handy tool is the *Accessibility Scanner* which can be found at https://play.google.com/store/apps/details?id=com.google.android.apps.accessibility.auditor. It analyses the active user interface at runtime and provides a textual (see Listing 3.1) and visual (see Figure 2) description of the findings. In the QAC for each error criteria the total number of occurrence is reported.

```
1  ...
2  Touch Target
3  com.avjindersinghsekhon.minimaltodo:id
4  /userToDoEditText
5  Consider making this clickable item
6  larger. This item's height is 44dp.
7  Consider a minimum height of 48dp
8  ....
9  Item label
10 com.avjindersinghsekhon.minimaltodo:id
11 /userToDoReminder
12 IconImageButton
13 This item may not have a label
14 readable by screen readers
15 ...
16 Image contrast
```

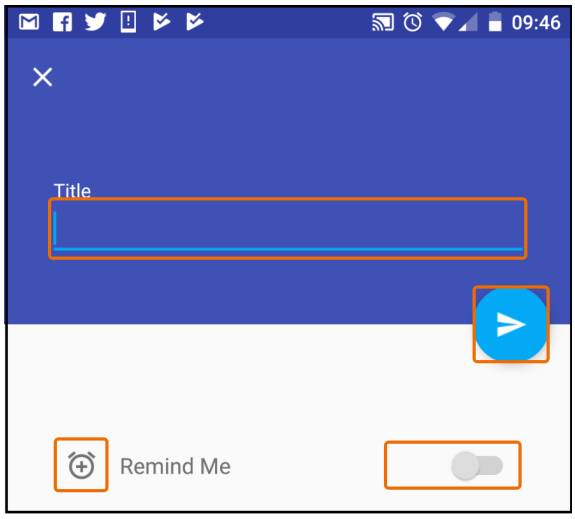

Figure 2: Tools can support the analysis of the UI accessibility by highlighting problematic elements onscreen.

```
17 com.avjindersinghsekhon.minimaltodo:id
18 /makeToDoFloatingActionButton
19 Consider increasing the contrast ratio
20 of this image's foreground and
21 background.
22 ...
23 Item description
24 com.avjindersinghsekhon.minimaltodo:id
25 /userToDoEditText
26 The clickable item's speakable
27 text("Title")is identical to that of 1
28 other item(1)
29 ...
```

Listing 1: Automated reports by the Google Accessibility Checker can provide hints about related source code.

The third and last section *Manual Checks* is a manual examination of informal criteria like the *position of interactive elements*, a *useful navigation* or *clear and understandable texts and descriptions*. These ratings are also in a scale form 1 to 5.

A complete list of all 15 heuristics is shown in Table 1. The *Quick Accessibility Check* QAC allows even accessibility amateurs to find and rate accessibility issues in a replicable and comparable way.

3.2 UsabML

The QAC results are formalised into Usability Markup Language (UsabML) for further processing. This step is especially important, as it results in an XML document format for the data, which means the information can processed by tools further on. The UsabML format allows validation checks on the data, it supports the rendering of results via various stylesheets to differently styled HTML documents for target

groups such as managers or developers. Additionally, scripts might process the data and create post issues to bug trackers.

As tools provide some hints about the source of an accessibility problem – compare the Google Accessibility Checker discussed above – this information can be part of an UsabML report. Optionally, usability managers or software developers might map issues with code locations.

Find in Listing 3.2 selected parts of the formalised report in UsabML, including the mapping of an issue to the related source file and line number.

```
<report id="rpt201334"
  gen-timestamp="2017-10-22T11:35:40Z"
  method="he">
 <title>NextcloudApp Q-Eval</title>
 ...
 <heuristic id="qac07">
  <title>Missing Labels</title>
  <description>
    Provide labels describing the
    purpose of an input field.
  </description>
 </heuristic>
 ...
 <negative-finding
   heuristic-id="heu12" rank="3"
   is-main-negative="true" id="neg3">
  <title>Button too small ...</title>
  <description>...></description>
  <found-by evaluator-id="eval_ek"/>
  <severity evaluator-id="eval_jf">
    <value>3</value>
  </severity>
  <document type="image">
    <description> The size should ...
    </description>
    <key>krainz33</key>
    <filename>krainz33.png</filename>
  </document>
  <code-reference
   project-id=" nextcloudapp"
   version-id="commit-eef54326rfe8"
   class-id="main.xml"
   package-id="layouts"
   method-id="main" line-number="24">
  </code-reference>
 </negative-finding>
 ...
</report
```

Listing 2: Formalised findings in *UsabML*.

One useful usage of the accessibility findings in the structured and formalised UsabML format is to render them with source code on demand. The idea is to allow developers to work in an environment and with tools (such as git) they are accustomed to.

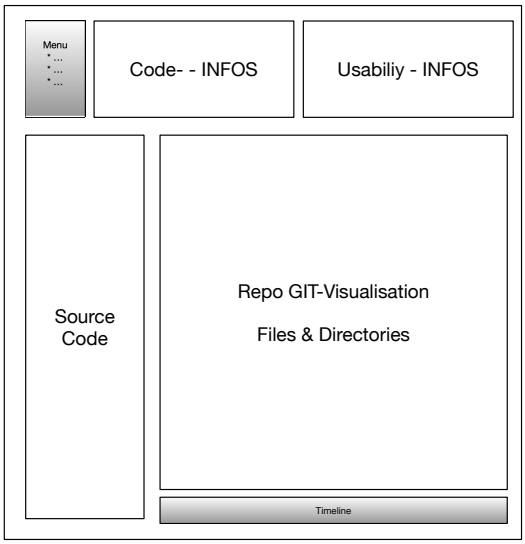

Figure 3: Visualising files of a project, source code and related accessibility issues side-by-side.

3.3 RepoVis

The web-based tool *RepoVis* consists of a backend which connects to existing git repositories and a frontend for rendering source code onscreen.

The visualisation allows to view all files within a repository at once to provide a dashboard-like overview. The layout of this frontend is shown in Figure 3, where separate – but connected – areas for the git repository overview, the source code of a single file, and the related issues are depicted.

The files are rendered as rectangular boxes with the pixels inside coloured according given metrics. For example, the pixels inside could show the author who last changed and committed single lines of the file. When drilling down, when inspecting single files, the source code related to a file is presented. In the same way an usability – in this case an accessibility – issue will be shown to the developer for selected files.

Developer can highlight on the dashboard all the files with connected accessibility issues. This way of presenting the current source code (or historical code from former commits) and related accessibility issues side-by-side should be an huge advantage for software engineers. An advantage in terms of speed for finding problematic code and in terms of code comprehension. For example, areas within the code base with hot spots of multiple issues can be detected more easily.

4 VALIDATION

To validate the approach we selected two popular (more than 10.000 installs) open source mobile apps, which on the one hand are examples where accessibility plays an important role and on the other hand apps where the source code is public available.

1. The *Nextcloud Client App* supports access to cloud documents with mobiles. Available at https://play.google.com/store/apps/details?id=com.nextcloud.client; find the source at https://github.com/nextcloud/ android; 100.000 installs.
2. The *Minimal ToDo App* allows to manage personal task lists. Available at https://play.google.com/store/apps/details?id=com.avjindersinghsekhon.minimaltodo; find the source code at https://github.com/avjinder/Minimal-Todo; 10.000 installs.

The evaluation steps follow the flow layout explained in Section 3 consisting of following steps:

- *QAC Evaluation* according the quick accessibility check criteria (again, compare Table 1 with the 15 A11Y metrics) and with tool support (in the scenario presented the Google Accessibility Scanner was used).
- *UsabML Formalisation* is done on the QAC results of Step 1. It outputs a single UsabML file, a structured report in XML format. Mapping issues to related lines of code is possible without knowing the ins and outs of a project, because the Google scanner already provides hints regarding the origin of accessibility problems within the source code.
- *RepoVis Visualisation* renders the connected git repository and uses the output of Step 2, the UsabML file. The connections specified in UsabML are extracted to present source and issues side-by-side.

The experts reviewing the two mobile apps rated for the first section of the QAC *AT Assistive technologies Android* each topic (Screenreader, Tabbing, external Keyboard) from 1 (best) – 5 (worse). Then the *Accessibility Scanner* output (number of issues found for Touch Size, Contrast Images, Contrast Text, Missing Labels, Redundant Labels and Implementation) were added. Finally, the experts rated the apps according the items 10 to 15 from the third section *Manual Checks*. There, for Clear/Concise Text, Evident Navigation, Font-size, Support for Zooming, Position of Elements and Colour-blindness their rating ranged from 1 (best) to 5 (worse). A summary of the results is shown in Table 2.

Table 2: Quick Accessibility Check (QAC) results summarised per section with notes about main reason for low ratings.

Ratings in section:	**App 1**	**App 2**	**Selected reasons for low ratings:**
Technologies Android	3,00	2,33	Screenreader support missing.
No of Accessibility Scanner Issues	4	6	Small touch sizes, low contrast, missing labels.
Manual Checks	1,67	1,83	Positioning of elements and lack of concise text

The format of the scanner and the condensed information in UsabML has been discussed in Section 3 and shown in Listing 3.2.

For the visualisation the repositories of the given apps where cloned from github to the local file systems and configured in the RepoVis system. The UsabML reports were provided by adding proper named xml file into the file system of the RepoVis backend server.

In Figure 4 the dashboard for an app is shown. Developers can view the source code and the related accessibility issues at once.

5 CONCLUSIONS

The contribution of this paper is a novel workflow with strong focus on quick accessibility checking (QAC) supporting the goal of increased willingness among software teams to perform evaluations. In a first step only 15 selected metrics deliver relevant findings about A11Y problems in mobile applications. A formalisation step results in xml based data (UsabML) which allows reuse and further (semi-)automatic processing. Finally, we contribute an integrated interactive visualisation (RepoVis) of source code augmented with the accessibility problems found.

First results with QAC in combination with RepoVis assure the authors that many further improvements for developers concerning accessible mobile applications are possible. The novel workflow presented will be a starting point to automate several steps of the process and integrate various representations of accessibility issues (for example with advanced visualisations).

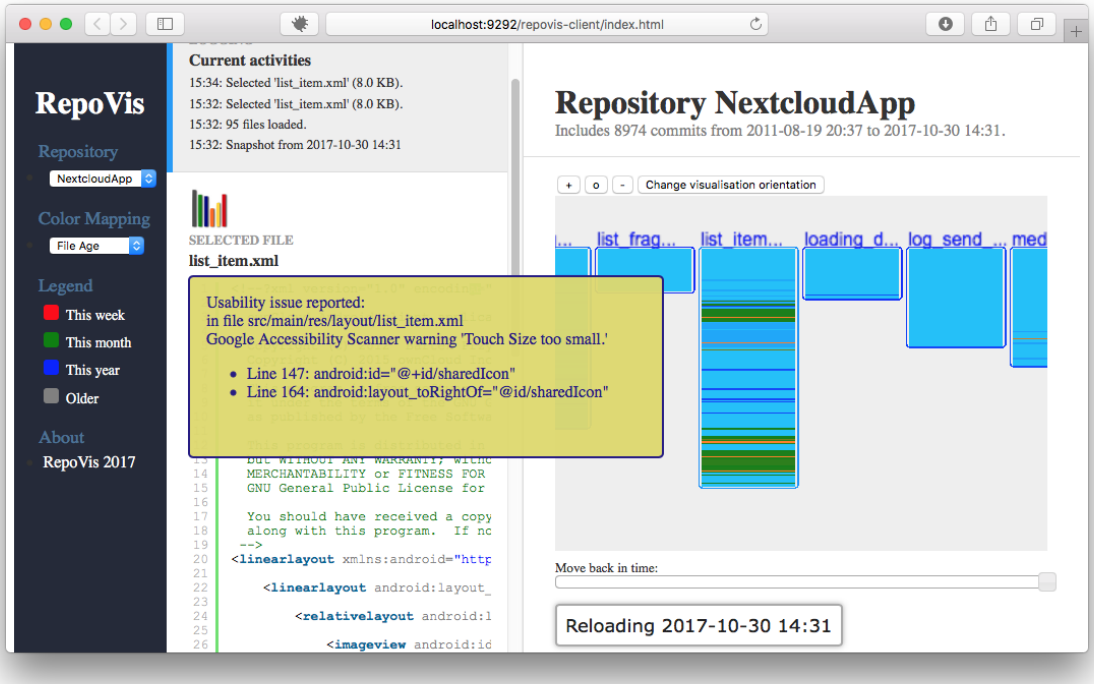

Figure 4: Visualising code and related accessibility issues side-by-side.

In upcoming research we plan to address issues, such as:

- Extending the target domain to usability in general. Not only accessibility issues, but other kinds of usability issues could be integrated.
- Including more tools to automate parts of the evaluation.
- Support for mapping issues and bugs found to the location in the source code. In many cases this mapping is not trivial and a semi-automated or otherwise tool-supported approach would be helpful.
- Extend the presented solution to support the iOS platform.
- Add tools to improve the QAC checking and mapping steps by auto-creating the suggested structured reporting format UsabML.

To conclude, we hope that in the future the computer science community will put an even stronger focus on the user experience and accessibility of mobile applications. This is necessary to support people with special needs, who rely on accessibility to use smartphone apps.

REFERENCES

Andrews, K. (2008). Evaluation comes in many guises. CHI 2008 Workshop on BEyond time and errors: novel evaLuation methods for Information Visualization (BELIV'08). http://www.dis.uniroma1.it/beliv08/pospap/andrews.pdf. Retrieved 2017-12-22.

Apple (2012). Accessibility programming guide for ios. https://developer.apple.com/library/content/documentation/UserExperience/Conceptual/iPhoneAccessibility/Accessibility_on_iPhone/Accessibility_on_iPhone.html. Retrieved 2017-12-22.

Caldwell, B., Reid, L. G., Cooper, M., and Vanderheiden, G. (2008). Web content accessibility guidelines (WCAG) 2.0. W3C recommendation, W3C. http://www.w3.org/TR/2008/REC-WCAG20-20081211/. Retrieved 2017-12-22.

Cheng, L. C. (2016). The mobile app usability inspection (MAUi) framework as a guide for minimal viable product (mvp) testing in lean development cycle. In *Proc. 2nd International Conference in HCI and UX on Indonesia 2016*, CHIuXiD 2016, pages 1–11.

Eick, S. G., Steffen, J. L., and Sumner, E. E. J. (1992). Seesoft — a tool for visualizing line oriented software statistics. *IEEE Trans. Softw. Eng.*, 18(11):957–968.

European Union (2016). Directive (eu) 2016/2102 of the european parliament and of the council of 26 october 2016 on the accessibility of the websites and mobile applications of public sector bodies. http://eur-

lex.europa.eu/eli/dir/2016/2102/oj. Retrieved 2017-12-22.

Feiner, J. and Andrews, K. (2012). Usability reporting with UsabML. In Winckler, M., Forbrig, P., and Bernhaupt, R., editors, *Proc. 4th International Conference on Human-Centered Software Engineering*, volume 7623 of *HCSE 2012*, pages 342–351. Lecture Notes in Computer Science, Springer Berlin / Heidelberg.

Feiner, J., Andrews, K., and Krajnc, E. (2010). UsabML – the usability report markup language: Formalising the exchange of usability findings. In *Proc. 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS 2010, pages 297–302. ACM.

Google (2017). Accessibility – usability – material design guidelines. https://material.io/guidelines/usability/accessibility.html. Retrieved 2017-12-22.

Hawes, N., Marshall, S., and Anslow, C. (2015). Codesurveyor: Mapping large-scale software to aid in code comprehension. In *3rd Working Conference on Software Visualization*, VISSOFT 2015, pages 96–105.

Henry, S. L. (2013). Easy checks – a first review of web accessibility. https://www.w3.org/WAI/eval/preliminary.html. Retrieved 2017-12-22.

Hvannberg, E. T., Law, E. L.-C., and Lérusdóttir, M. K. (2007). Heuristic evaluation: Comparing ways of finding and reporting usability problems. *Interacting with Computers*, 19(2):225–240. http://kth.diva-portal.org/smash/get/diva2:527483/FULLTEXT01. Retrieved 2017-12-22.

ISO (2000). ISO/DIS 9241-11 ergonomics of human-system interaction – part 11: Usability: Definitions and concepts. Standard, International Organization for Standardization. https://www.iso.org/standard/63500.html. Retrieved 2017-12-22.

ISO (2006). SO/IEC 25062:2006 software engineering – software product quality requirements and evaluation (SQuaRE) – common industry format (CIF) for usability test reports. International Organization for Standardization. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=43046. Retrieved 2017-12-22.

ISO (2008). ISO/DIS 9241-171: 2008 ergonomics of human-system interaction – part 171: Guidance on software accessibility. Technical report, International Organization for Standardization. http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?csnumber=39080. Retrieved 2017-12-22.

Komiyama, T. (2008). Usability evaluation based on international standards for software quality evaluation. Technical Journal 2, NEC. http://www.nec.co.jp/techrep/en/journal/g08/n02/080207.pdf. Retrieved 2017-12-22.

Molich, R., Chattratichart, J., Hinkle, V., Jensen, J. J., Kirakowski, J., Sauro, J., Sharon, T., and Traynor, B. (2010). Rent a car in just 0, 60, 240 or 1,217 seconds? — comparative usability measurement, cue-8. *Journal of Usability Studies*, 6(1):8–24. http://www.upassoc.org/upa_publications/jus/2010november/JUS_Molich_November_2010.pdf. Retrieved 2017-12-22.

Murphy, G. C. (2016). Technical perspective: Software is natural. *Commun. ACM*, 59(5):121–121.

Nielsen, J. (1994). Guerrilla hci – using discount usability engineering to penetrate the intimidation barrier. http://www.nngroup.com/articles/guerrilla-hci/. Retrieved 2017-12-22.

Nielsen, J. and Molich, R. (1990). Heuristic evaluation of user interfaces. In *Proc. Conference on Human Factors in Computing Systems*, CHI '90, pages 249–256. ACM.

Patch, K., Allan, J., Lowney, G., and Spellman, J. F. (2015a). User agent accessibility guidelines (UAAG) 2.0. W3C note, W3C. http://www.w3.org/ TR/ 2015/NOTE-UAAG20-20151215/. Retrieved 2017-12-22.

Patch, K., Spellman, J., and Wahlbin, K. (2015b). Mobile accessibility: How wcag 2.0 and other w3c/wai guidelines apply to mobile. Technical report, W3C. https://www.w3.org/TR/mobile-accessibility-mapping/. Retrieved 2017-12-22.

Shaw, M. (2002). What makes good research in software engineering? *International Journal on Software Tools for Technology Transfer*, 4(1):1–7.

The United Nations (2006). Convention on the rights of persons with disabilities. *Treaty Series*, 2515:3. https://www.un.org/development/desa/disabilities/convention-on-the-rights-of-persons-with-disabilities/convention-on-the-rights-of-persons-with-disabilities-2.html. Retrieved 2017-12-22.

Treviranus, J., Richards, J., and Spellman, J. F. (2015). Authoring tool accessibility guidelines (ATAG) 2.0. W3C recommendation, W3C. http://www.w3.org/TR/2015/REC-ATAG20-20150924/. Retrieved 2017-12-22.

Vilbergsdóttir, S. G., Hvannberg, E. T., and Law, E. L.-C. (2006). Classification of usability problems (cup) scheme: Augmentation and exploitation. In *Proc. 4th Nordic Conference on Human-Computer Interaction: Changing Rules (NordiCHI2006)*, pages 281–290. http://diuf.unifr.ch/people/pallottv/docs/NordiCHI-2006/LongPapers/p281-vilbergsdottir.pdf. Retrieved 2017-12-22.

Yusop, N. S. M., Grundy, J., and Vasa, R. (2016). Reporting usability defects: Do reporters report what software developers need? In *Proc. 20th International Conference on Evaluation and Assessment in Software Engineering*, EASE '16, pages 38:1–38:10. ACM.