How Artificial Intelligence Can Improve Web Development and Testing

(Invited Paper)

Andrea Stocco
Università della Svizzera Italiana
Lugano, Switzerland
andrea.stocco@usi.ch

ABSTRACT

The Artificial Intelligence (AI) revolution in software development is just around the corner. With the rise of AI, developers are expected to play a different role from the traditional role of programmers, as they will need to adapt their know-how and skillsets to complement and apply AI-based tools and techniques into their traditional web development workflow. In this extended abstract, some of the current trends on how AI is being leveraged to enhance web development and testing are discussed, along with some of the main opportunities and challenges for researchers.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**;

KEYWORDS

artificial intelligence, web development, web testing

ACM Reference Format:
Andrea Stocco. 2019. How Artificial Intelligence Can Improve Web Development and Testing: *(Invited Paper)*. In *Companion of the 3rd International Conference on Art, Science, and Engineering of Programming (Programming '19), April 1–4, 2019, Genova, Italy.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3328433.3328447

1 INTRODUCTION

Our world is changing in many ways, and one of the things which will have a huge impact on our future is artificial intelligence (AI). AI is a hot topic with many practical applications such as self-driving vehicles, voice-assisted control, automated traders, and customer service chatbots.

AI techniques have shown great results when a substantial amount of data are available. One domain in which we have a huge amount of data to analyze is *software*. However, the use of AI in software development is still in its infancy. Thus, why not take advantage of such powerful techniques to help software engineers analyze and address existing challenges in software engineering?

Programming '19, April 1–4, 2019, Genova, Italy

ACM ISBN 978-1-4503-6257-3/19/04.
https://doi.org/10.1145/3328433.3328447

The goal of AI is to provide a set of algorithms and techniques that can be used to perform tasks that humans accomplish intuitively and nearly automatically, but that are otherwise very challenging for computers. Research in AI embodies a large and diverse amount of work related to automated machine reasoning (Figure 1). However, two subfields are of interest in the scope of this paper, namely *machine learning* and *computer vision*.

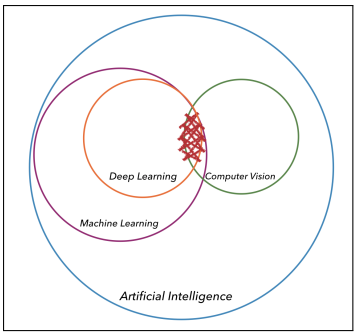

Figure 1: Image inspired by Fig. 1.4 of "Deep Learning" by Goodfellow et al. [8].

Machine learning (ML), in a nutshell, is related to pattern recognition and learning from data in order to solve classification or regression problems. The performance of machine learning algorithms depends heavily on the representation of the data they are given. Many artificial intelligence tasks can be solved by designing and extracting the right set of features for each task, then providing these features to an ML algorithm. However, for many tasks, it is difficult to know *apriori* what features should be extracted.

Deep learning solves the problem of finding the right representation by introducing hierarchies of representations that are expressed in terms of other, simpler, representations. Input information is sent through layers activated by nonlinear functions. Each layer transforms the raw input (first level) into progressively more abstract representations (inner, or hidden, levels). As such, a deep learning algorithm is able to discover underlying hidden patterns in data, which allow the algorithm to correctly perform a task, even on previously unseen data.

Another AI technique that has been growing in popularity is *computer vision* (CV), which provides techniques for analyzing and understanding images, similar to the way humans perceive them. In web development and testing, these techniques are often applied or combined together to form powerful analysis tools (Figure 1).

2 AI FOR WEB DEVELOPMENT AND TESTING

2.1 AI for Web Development

An area in which AI is being utilized is the design and development of graphical user interfaces (GUIs). The user interface design process often starts with designers sharing ideas and sketches on a whiteboard. Once a design is drawn, it is usually captured within a picture, and manually translated by the development team into a working HTML wireframe to begin the development process. This requires considerable effort and often delays the design process. Recently, several AI-based approaches have been proposed to generate HTML wireframes directly from a hand-drawn image, giving an instant working design implementation to streamline the design process. We describe some of the main existing approaches next.

Nguyen and Csallner [16] proposed Remaui, which aims to reverse engineer mobile GUIs. Given an image screenshot, Remaui first uses Optical Character Recognition (OCR) to compute an over-approximated set of textual elements. Then, it uses heuristics such as the text size to remove false positives, i.e., non-word elements that may be erroneously reported by the OCR component. Second, Remaui uses CV to infer the hierarchy of elements on the screenshot. Specifically, edge detection is used to estimate the contours of each element, and edge dilatation is adopted to merge semantically close elements. Then, the bounding box of each identified element is computed, and outputs of both OCR and CV are merged to produce the final *visually inferred GUI*. However, Remaui only classifies detected components into either text or images, and it is not able to detect the actual type of the component (e.g., button or text field).

The first method to adopt deep learning was `pix2code`, proposed in an open access paper by Beltramelli [3]. The approach consists of training an end-to-end model to automatically generate code from a single input image. Initially, it uses Convolutional and Recurrent Neural Networks to understand the given GUI image. It infers the image's objects, their positions, and types (i.e., buttons or labels). Then, `pix2code` uses a language model (i.e., a DSL for describing GUIs) to generate syntactically and semantically correct GUI descriptions in terms of the identified objects. Finally, it utilizes the latent variables inferred from the image to generate corresponding textual descriptions (i.e., source code) of the objects represented by these variables. `pix2code` has shown promising results across three different platforms, i.e., iOS, Android, and web. However, it requires a DSL that hinders its adoption in practice.

The last described approach is ReDraw, by Moran et al. [15]. First, the approach detects the bounding boxes of logical atomic elements of a GUI from a mock-up artifact using computer vision techniques and mock-up metadata. Then, software repository mining and automated dynamic analysis are used to collect screenshots and GUI metadata to automatically derive labeled training data. Such data are then utilized to train a deep convolutional neural network (CNN) that classifies GUI-components into domain-specific types (e.g., buttons). Finally, a k-nearest-neighbors (KNN) algorithm is used to generate a hierarchical GUI structure from which a prototype application can be automatically assembled. Experimental results show that ReDraw outperforms Remaui and `pix2code` in terms of accuracy of the generated GUI interfaces.

2.2 AI for Web Testing

Recently, software engineering community has witnessed an increasing adoption of CV techniques for assisting or solving common software engineering tasks.

One of the foundational approaches for computer vision applied to testing is by Chang et al. [4]. Their tool, Sikuli, allows testers to write a visual test script that uses images to specify which GUI components to interact with and what visual feedback to observe. Their work shows how this approach can facilitate a number of testing activities such as unit testing, regression testing, and test-driven development.

CV techniques have been employed to detect *cross-browser incompatibilities* (XBIs) in web applications. XBIs are frequently occurring issues in a web page's appearance and/or behaviour when the page is viewed on different web browsers [18]. Identifying such differences requires considerable manual effort, which can be effectively reduced using a visual-based technique. For instance, WebSee [13] is a visual technique that compares whole images with a perceptual difference algorithm. WebDiff [18] and X-PERT [19] utilize an image similarity technique based on image colour histogram.

Recently, computer vision has been also applied for web test migration and test repair. The tool PESTO [11, 12, 21] migrates Selenium DOM-based web tests to visual tests based on Sikuli's image recognition capability. It proposes an auto-scaling template matching algorithm for the automatic construction of visual locators. Visual locators are retrieved for each web element the tests interact with, and verified on the same web page where they have been captured. The tool Vista [22, 23] has been proposed to aid the repair of tests, and therefore matches visual locators across versions of the same web page. This is used to validate and repair tests during regression testing. Vista is based on a fast image-processing pipeline that combines feature detection and template matching to automatically suggest and apply repairs to broken web tests. The insight is using the GUI and visual technologies to support the preventive detection of breakages, by checking the GUI actions performed by the tests and validating them at runtime, timely detecting deviations from the correct behaviour.

Computer vision has also been adopted for *root cause analysis* of presentational issues occurring in PDF files [9]. To *prioritize test reports*, Feng et al. [7] use the screenshots provided by users to augment the existing textual test prioritization techniques for mobile applications.

Kıraç et al. [10] used an image processing pipeline for test oracle automation of visual output systems. Bajammal et al. [1] use visual methods to generate reusable web components from a mockup in order to facilitate GUI web development. In another work, Bajammal and Mesbah [2] use visual analysis to infer a DOM-like state of HTML `canvas` elements, thus making them testable by commonly used testing approaches.

Finally, to *automatically generate test cases*, Zhang et al. [26] use a visual-aided approach that identifies strokes that testers draw on screenshots taken from the apps. These sketches are used to define test specifications (e.g., coordinates where a visual object should be positioned to on the screen), which are subsequently used to generate test cases automatically.

3 CHALLENGES AND OPPORTUNITIES

3.1 Challenges

Hidden Complexity Demands Thorough Testing. Developing reliable ML/CV code requires substantial expertise and knowledge, arguably more than that required for tradition software development. For instance, developers implement deep neural networks (DNNs) using popular frameworks such as Keras [5]. Such frameworks hide the complexity of building computational graphs behind convenient APIs that allow the implementation of rather complex multi-layer architectures with only a few lines of code. Unlike traditional software, in most cases, the resulting DNN will perform training and compute a result despite the presence of functional bugs in the model. Hence, in case of misbehaviours, the *only* feedback to developers is the final classification score, while the debugging phase needs to be performed on the *entire* network architecture.

To this aim, testing DNNs is a fervid and growing research area. Researchers have proposed methods for testing DNNs [17, 24], testing of DNN-based autonomous vehicles [25, 27], and fault localization for DNNs [6, 20].

The Need for Explainable AI. Although the learning process is deterministic to some extent, it is almost impossible from a practical perspective to extract the model from the internal workings of a learning system due to its sheer complexity, caused by a myriad of dynamic parameters (e.g., weights or biases). As a direct consequence, the learned models cannot be easily interpreted, explained, or understood by humans.

Research community ought to work towards creating ML techniques that produce more explainable models, while maintaining a high level of accuracy. Such models should enable users to understand, manage, and therefore trust AI-based software.

3.2 Opportunities

Voice-based Search. Voice-based search has recently gained momentum due to the introduction of various virtual assistants, such as Google Assistant, Amazon Alexa, and Apple Siri. With the increase in the use of these digital assistants, web development needs to look into the evolution of voice-based search. This voice-based technology will become a complete necessity in domains such as virtual shopping. Hence, AI bots powered by voice will be an essential asset of the future of this technology.

UX & Accessibility. AI can play a vital role in the future of UX in research and practice. For instance, image recognition can be used to generate or fix alternative texts in webpages. Facial recognition may soon replace CAPTCHA. Additionally, lip reading can be used to generate video capture, or for automated text summarization. AI can also take an active role in web accessibility, i.e., proper design of websites for people with disabilities. For example, AI can be used to generate Braille texts from images for visually impaired people.

Learning Change Patterns. Software repositories are invaluable sources of information for researchers interested in automated tools for assisting software development. For instance, Neural Machine Translation has recently been used at Google to learn patterns of code changes, extracted from AST diffs between failure and resolution pairs, and to suggest candidate repairs in form of AST changes [14]. Evaluation results show that the technique generates approximately 50% correct fixes.

Automated Test Generation and Maintenance through AI. Despite their wide adoption, existing test automation frameworks, such as Selenium, have limitations. First, the development of real-size test suites for complex web apps is still time-consuming and laborious. Second, the maintenance cost of test code as the application evolves is notoriously high. Additionally, such tools typically operate at the code-level, without taking into account the visual aspects of the application.

To mitigate these issues, a paradigm-shift in web test automation is necessary to make it easier for testers to quickly verify the outcome of regression test suites and take corrective actions in the test code with low effort. To this aim, a novel unified framework for automated visual regression testing of web apps would greatly alleviate the need for writing test cases manually. Our insight to overcome the aforementioned limitations would be to join the power of automated crawling with that of advanced computer vision techniques to allow easier and more efficient (1) test case generation, (2) test case inspection and maintenance, and (3) test case repair.

However, there are several challenges towards the development of such framework. The *first challenge* encompasses studying novel state abstraction functions that take into account the visual characteristics of a web application, e.g., robust visual perceptual hashing for image comparison. Such algorithms may produce more user-centric models, representing what the end-users see and perceive.

The *second challenge* requires devising novel techniques to compare different web application models and visual test execution traces effectively and efficiently. To aid solving regression testing-related issues, such techniques must take advantage of the information derived from rendering of a page in a browser. The techniques should not only compare different models captured at different time intervals, but also highlight differences pertaining to regression bugs in the application. To decrease the false positives rate, these techniques should also be capable of automatically discarding irrelevant parts of web pages, e.g., ads or timestamps.

With regard to maintenance, as a web application evolves, the initial model should also co-evolve, which might impact the existing generated test suite. A straightforward solution may be regenerating the test suite on each new revision by re-crawling the application. A major drawback of this approach is the loss of all previous testing efforts in terms of test cases and assertions, which testers may want to retain. As a *third challenge*, we need to devise techniques that are capable of updating existing models, partially and incrementally. The results will be presented to the end users for inspection, which can help them identify the test cases that can be retained and those which need manual maintenance.

4 CONCLUSIONS

The decision on selecting and applying a certain learning system depends on the problem one wants to solve, and it is always a trade-off among efficiency, training costs, and understanding. In this extended abstract we described applications of AI to web development and testing, along with some of the main opportunities and challenges for researchers.

REFERENCES

[1] Mohammad Bajammal, Davood Mazinanian, and Ali Mesbah. 2018. Generating Reusable Web Components from Mockups. In *Proceedings of the 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE '18)*.

[2] Mohammad Bajammal and Ali Mesbah. 2018. Web Canvas Testing through Visual Inference. In *Proceedings of the 11th IEEE International Conference on Software Testing, Verification and Validation (ICSE '18)*.

[3] Tony Beltramelli. 2017. pix2code: Generating Code from a Graphical User Interface Screenshot. *CoRR* abs/1705.07962 (2017). http://arxiv.org/abs/1705.07962

[4] Tsung-Hsiang Chang, Tom Yeh, and Robert C. Miller. 2010. GUI testing using computer vision. In *Proceedings of 28th ACM Conference on Human Factors in Computing Systems (CHI 2010)*. ACM, 1535–1544.

[5] François Chollet and others. 2015. Keras. https://keras.io. (2015).

[6] Hasan Ferit Eniser, Simos Gerasimou, and Alper Sen. 2019. DeepFault: Fault Localization for Deep Neural Networks. In *Fundamental Approaches to Software Engineering*, Reiner Hähnle and Wil van der Aalst (Eds.). Springer International Publishing, Cham, 171–191.

[7] Yang Feng, James A. Jones, Zhenyu Chen, and Chunrong Fang. 2016. Multi-objective Test Report Prioritization Using Image Understanding. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE '16)*. 202–213.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[9] Tomasz Kuchta, Thibaud Lutellier, Edmund Wong, Lin Tan, and Cristian Cadar. 2018. On the correctness of electronic documents: studying, finding, and localizing inconsistency bugs in PDF readers and files. *EMSE* (2018).

[10] M. Furkan Kıraç, Barış Aktemur, and Hasan Sözer. 2018. VISOR: A fast image processing pipeline with scaling and translation invariance for test oracle automation of visual output systems. *JSS* 136 (2018), 266–277.

[11] Maurizio Leotta, Andrea Stocco, Filippo Ricca, and Paolo Tonella. 2015. Automated Migration of DOM-based to Visual Web Tests. In *Proceedings of 30th Symposium on Applied Computing (SAC 2015)*. ACM, 775–782.

[12] Maurizio Leotta, Andrea Stocco, Filippo Ricca, and Paolo Tonella. 2018. PESTO: Automated migration of DOM-based Web tests towards the visual approach. *Software Testing, Verification And Reliability* 28, 4 (2018).

[13] S. Mahajan and W. G. J. Halfond. 2015. Detection and Localization of HTML Presentation Failures Using Computer Vision-Based Techniques. In *Proceedings of 8th IEEE International Conference on Software Testing, Verification and Validation (ICST '15)*. 1–10.

[14] Ali Mesbah, Andrew Rice, Eddie Aftandilian, Emily Johnston, and Nick Glorioso. 2019. Analyzing and Repairing Compilation Errors. In *Proceedings of the 41st ACM/IEEE International Conference on Software Engineering - Poster Track (ICSE '19 Companion)*.

[15] K. P. Moran, C. Bernal-Cárdenas, M. Curcio, R. Bonett, and D. Poshyvanyk. 2018. Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps. *IEEE Transactions on Software Engineering* (2018).

[16] T. A. Nguyen and C. Csallner. 2015. Reverse Engineering Mobile Application User Interfaces with REMAUI (T). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 248–259.

[17] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*. ACM, New York, NY, USA, 1–18.

[18] Shauvik Roy Choudhary, Mukul R. Prasad, and Alessandro Orso. 2013. X-PERT: Accurate Identification of Cross-browser Issues in Web Applications. In *Proc. of the 2013 International Conference on Software Engineering (ICSE '13)*. IEEE Press, Piscataway, NJ, USA, 702–711.

[19] Shauvik Roy Choudhary, Husayn Versee, and Alessandro Orso. 2010. WEBDIFF: Automated Identification of Cross-browser Issues in Web Applications. In *Proc. of the 2010 IEEE International Conference on Software Maintenance (ICSM '10)*. IEEE Computer Society, Washington, DC, USA, 1–10.

[20] Prashanta Saha and Upulee Kanewala. 2019. Fault Detection Effectiveness of Metamorphic Relations Developed for Testing Supervised Classifiers. *CoRR* abs/1904.07348 (2019). arXiv:1904.07348 http://arxiv.org/abs/1904.07348

[21] Andrea Stocco, Maurizio Leotta, Filippo Ricca, and Paolo Tonella. 2014. PESTO: A Tool for Migrating DOM-based to Visual Web Tests. In *Proceedings of 14th International Working Conference on Source Code Analysis and Manipulation (SCAM '14)*. IEEE Computer Society, 65–70.

[22] Andrea Stocco, Rahulkrishna Yandrapally, and Ali Mesbah. 2018. Visual Web Test Repair. In *Proceedings of the joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*. 12 pages.

[23] Andrea Stocco, Rahulkrishna Yandrapally, and Ali Mesbah. 2018. Web Test Repair Using Computer Vision. In *Proceedings of 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018 - Demonstration Track)*. ACM.

[24] Youcheng Sun, Xiaowei Huang, and Daniel Kroening. 2018. Testing Deep Neural Networks. *CoRR* abs/1803.04792 (2018). arXiv:1803.04792 http://arxiv.org/abs/1803.04792

[25] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. DeepTest: Automated Testing of Deep-neural-network-driven Autonomous Cars. In *Proceedings of the 40th International Conference on Software Engineering (ICSE '18)*. ACM, New York, NY, USA, 303–314.

[26] Chucheng Zhang, Haoliang Cheng, Enyi Tang, Xin Chen, Lei Bu, and Xuandong Li. 2017. Sketch-guided GUI Test Generation for Mobile Applications. In *Proc. of ASE '17*. 38–43.

[27] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. DeepRoad: GAN-based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE 2018)*. ACM, New York, NY, USA, 132–142.