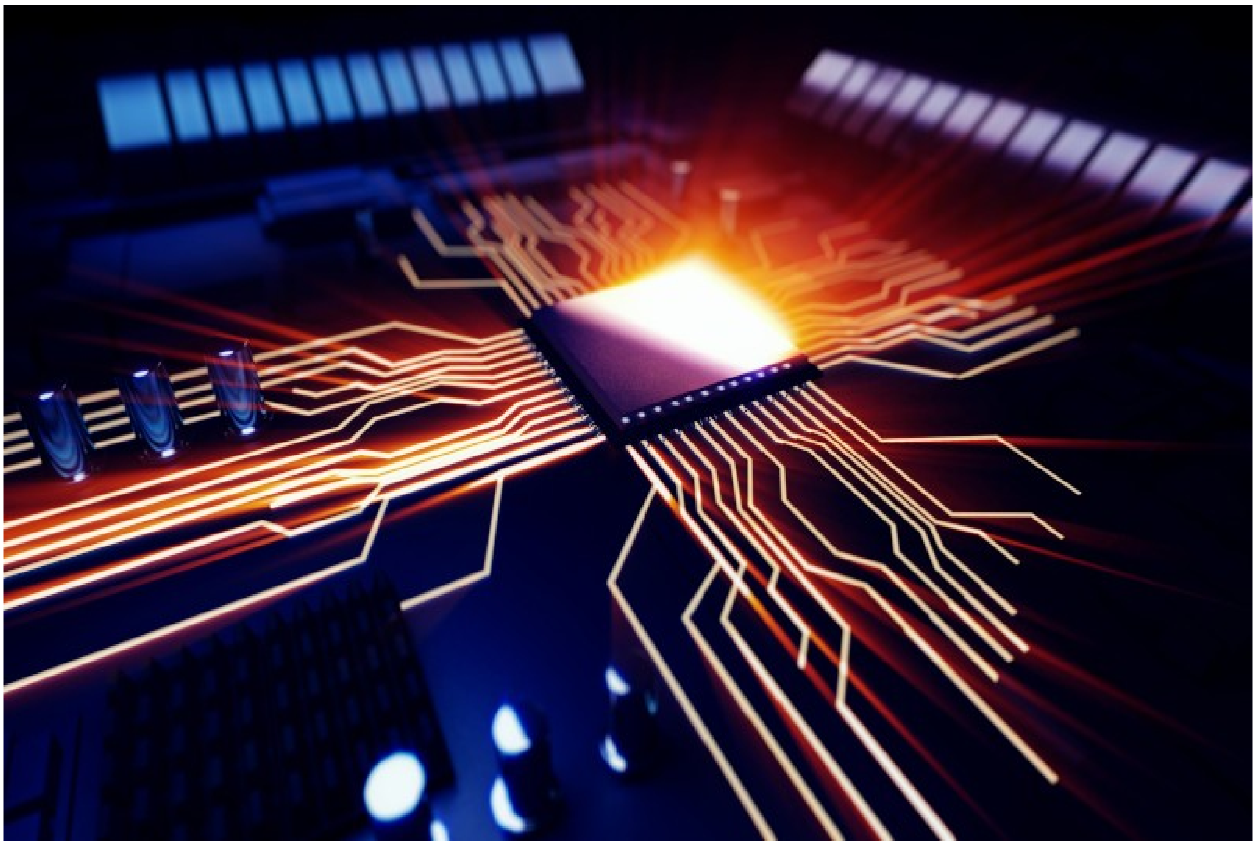
White paper [PRELIMINARY- September 28, 2020]

AI-Powered Testing Automation

Expand your testing capacity, get faster feedback, and improve quality through AI-based test automation



Overview
Bluewind shows how to apply artificial intelligence(AI) to test automation.

The presented work is based on recent developments at Bluewind exploiting AI/ML in real product testing.
Notably, an AI-powered test setup rfor medical equipment (End of Line), revealed an increase of 20% fault detection, and a decrease in manual testing (production) estimated around 45%.
Another AI-powerd test setup for Industrial Sensors, (Functional Test) provided a 15% increase in quality performance, while requiring only 25% of the previous manual test effort.

As more and more organizations have taken to Agile development, they are carrying out development and testing in multiple iterations. The concepts of "Continuous Integration" (CI), "Continuous Development" (CD), and "Continuous Deployment" are very important in Agile.

Organizations can make continuous deployment a reality only if they continuously test their applications. As development proceeds at a fast pace, the testing must keep pace with it. This makes test automation very important in the world of Agile development.

Earlier, manual testing ruled the world of testing, however, test automation increasingly became a reality in most organizations developing software. Testing continued to evolve, and it took advantage of technology innovations. Artificial Intelligence(AI) is one such technology that has made a substantial contribution to automation in general.

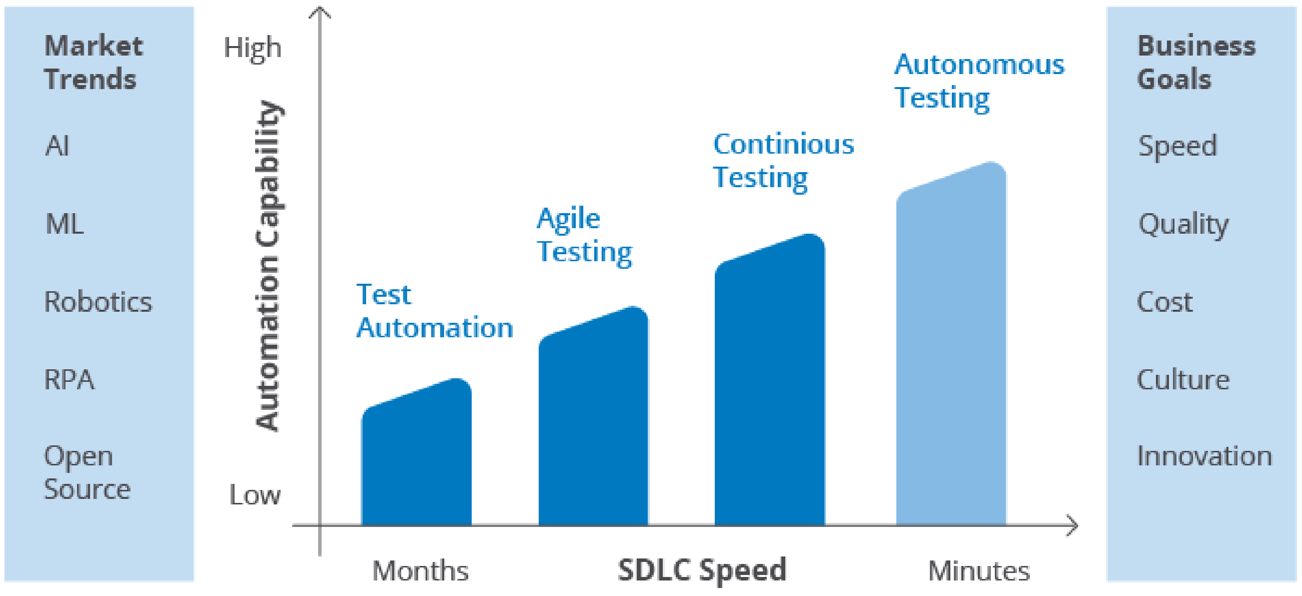
The benefits for AI in software testing are astronomical. While still in its infancy, artificial intelligence is starting to impact software development and quality assurance and seems to be the natural progression in the race to seamless and fast delivery cycles.

AI in Software Test Automation

The use of AI in software development is still in its infancy, and the level of autonomy is much lower than seen in more evolved areas such as self-driving systems or voice-assisted control, although it is still driving forward in the direction of autonomous testing.

The application of AI in software testing tools is focused on making the software development lifecycle easier. Through the application of reasoning, problem solving, and in some cases, machine learning, AI can be used to help automate and reduce the amount of mundane and tedious tasks in development and testing.
"Don't test automation tools do this already?" you might ask. And the answer is of course, "Yes! They do!" ...but they have limitations.

Where AI shines in software development is when it is applied to remove those limitations, to enable software test automation tools to provide even more value to developers and testers. The value of AI comes from reducing the direct involvement of the developer or tester in the most mundane tasks. (Human intelligence is still very much needed in applying business logic, etc.).



AI and machine learning

So, what about machine learning? Machine learning can augment the AI by applying algorithms that allow the tool to improve automatically by collecting the copious amounts of data produced by testing.

Machine learning research is a subset of overall AI research, with a focus on decision-making based on previously-observed data. This is an important aspect of AI overall, as intelligence requires modifying decision-making as learning improves. In software testing tools, though, machine learning isn't always necessary — sometimes an AI-enabled tool is best manually fine-tuned to suit the organization using the tool, and then the same logic and reasoning can be applied every time, regardless of the outcome.

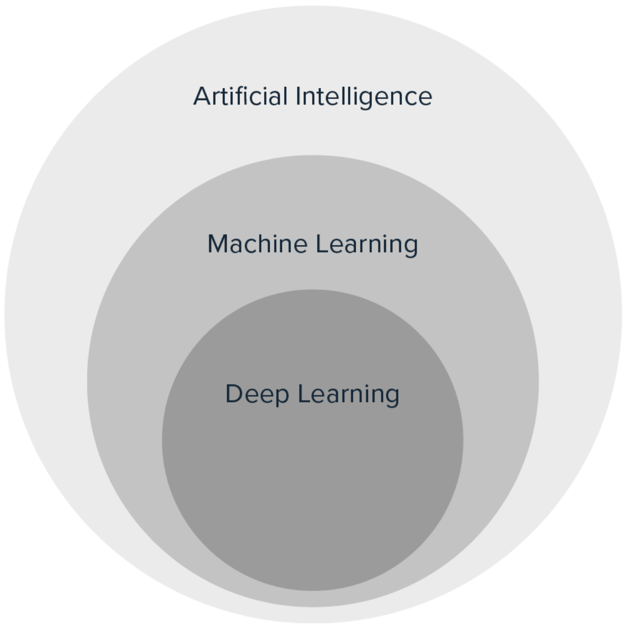
In other cases, data collection is key to the decision-making process, and machine learning can be extremely valuable, requiring some data initially and then improving or adapting as more data is collected. For example, code coverage, static analysis results, test results, or other software metrics, over time, can inform the AI about the state of the software project.

Deep Learning

Machine Learning has its own subset called Deep Learning, which is built on the processing of vast amount of data to learn from. Such data, in most cases, is represented by multilayered Neural Networks – they are models inspired by net of human neurons, helping computers to acquire new knowledge and to reason highly intelligently.

The key aspect of Deep Learning is huge amount of information represented by Neural Networks to drive decision-making process. Such amount of data is not always available or not applicable in software testing – maybe that's why we don't see many cases of Deep Learning usage in those areas yet.

One possible example would be "learning" from tens of millions lines of code to understand different types of security violations, and to implement a static analysis engine based on such deep learning model.



Examples of AI and Machine Learning in Software Testing

This is an important area of research and development at Bluewind. Excitingly, our current offerings and our ongoing research in AI and ML continues to bring new ways to integrate these technologies into our products. Here are a few ways we have already brought them in.

Using AI and Machine Learning to Improve the Adoption of Static Analysis

One of the roadblocks to successful adoption of static analysis tools is managing a large number of warnings and dealing with false positives (warnings that are not real bugs) in the results. Software teams that analyze a legacy or existing code base struggle with the initial results they get with static analysis and are turned off by this experience enough to not pursue further effort. Part of the reason for being overwhelmed is the numbers of standards, rules (checkers), recommendations, and metrics that are possible with modern static analysis tools.

Software development teams have unique quality requirements and there are no one-size-fits-all recommendations for checkers or coding standards. Each team has their own definition of false positive, often meaning "don't care" rather than "this is technically incorrect." Bluewind's solution to this is to apply AI and machine learning to prioritize the findings reported by static analysis to improve the user experience and adoption of such tools.

Bluewind uses a method to quickly classify the findings in the output of a static analysis tool as either something that the team wants to see or something the team wants to suppress by reviewing a small number of findings and constructing a classifier based on the metadata associated with those findings. This classifier is based on results of previous manual classifications of static analysis findings in the context of both historical suppressions of irrelevant warnings and prior prioritization of meaningful findings to fix inside the codebase.

The end results are classified in two ways:

- Of interest for the team to investigate
- Items that can be suppressed

This greatly improves the user experience by directing developers to warnings that have the highest likelihood of applying to their project. With these innovations, organizations can immediately reduce manual effort in their adoption and use of static analysis.

End of line testing strategy using python+Ml

End of line testing is a process in which teams will validate an entire application workflow, from start to finish.

The root goal of running end-to-end tests is to ensure critical business processes function properly, across an entire application's architecture and user scenarios.
Automating end-to-end testing is possible, but is extremely hard to maintain and scale.

Bluewind proposes an end of line testing strategy based on python as test preparation framework and on ML for bug discovery.

Python as a test preparation framework

Bluewind develop an automation framework that enables testers to write test cases easily and developing interactions with the device under test. The framework was developed with python.

Python is one of the most popular programming languages. Python was an obvious choice due to several reasons:

- Enables adding AI/ML parts very easily
- Many automation tools are available
- No compilation necessary
- Python is easy to read and is very flexible
- Python is mature with much supports behind it

ML for bug discovery

AI\ML is very useful in the end of line(also while software developing) context for bad behavior detection.
It's possible to test by learning instead of deeply understanding the good/bad behavior of a complex system.
Knowing in advance mainly what a "good behavior" is, and not knowing all possible failure modes, is a perfect fit for ML.

Even if you knew what to look for, finding anomalous behavior then connecting the dots to develop a complete picture from a huge number of activities may turn out to be humanly impossible. Especially, if you have a large group of users. The data points can easily end up into hundreds of thousands, even exceeding millions. Machine learning can be very good at crunching such large data and finding patterns outside the nominal baseline. Machine learning is also good at finding clues in datasets spread across multiple sources.

A common approach used by many solutions is 'anomaly detection', also known as 'outlier detection'. The idea is: a user's (software or hardware) behavior should match with the rest in their group or past activities, called a baseline. Events or observations that deviate from this baseline is an anomaly. Typically, such an anomaly might be an indicator of fraud, sabotage, collusion, data theft or other malicious intent. Once an early deviation is detected, the algorithm can flag the incident for further investigation or if designed to do so, compare the incident with similar events recorded in the past. This record(s) could be the result of a previously executed Supervised algorithm where the anomalies were labeled as 'normal' or 'abnormal' by a human analyst, acquired from previous training data or a crowd-sourced knowledge base.

Finally, the threat is reported with a risk score factoring in the frequency, resources involved, potential impact, number of nodes it's affecting and other variables. Here it's important to mention that the same ML approach is applied during software development (this is routing during development) in order to enhance test cases and execute a double check to filter more device problems

How AI-Powered Testing Automation Transforms Businesses

Businesses that have a commitment to implementing AI at the enterprise level are already experiencing greater operational efficiency and better product results.

Developers are renegotiating their involvement within Agile and DevOps strategies, as smart algorithms are now capable of tackling the most repetitive problems presented in testing automation. Not only is product development significantly streamlined when testing automation changes from the bottleneck to the catalyst within a CI/CD pipeline, but also, executives are provided with business intelligence previously unavailable that directly impacts the bottom line.

Functionize is partnering with Google Cloud to build advanced anomaly detection through canary testing where a small set of users are used for real-world testing of new code. AI is used to compare the experience of these users with those running the existing code. Anomalies can then be identified automatically, and details passed back to the developers.

The Future of Artificial Intelligence and Machine Learning

So what comes next?

Bluewind is very active in this space, continuing to pursue further applications of artificial intelligence and machine learning to augment our software testing tool suite.

There are many routes of research, but the end goal is clear: to help teams develop and test their code more efficiently and effectively, to create higher quality software at speed.