Challenges in Testing an Intelligent Software

Anurag Sharma, Satish Yogachar

(anurag.sharma@intel.com, satish.yogachar@intel.com)

Abstract

Artificial Intelligence is always a fascinating field. An intelligent system can perceive its environment, learn, adapt and take action to maximize the success. AI has always been considered a highly specialized field, mostly limited to research labs. Researchers are using various techniques such as Genetic Algorithms, Fuzzy Logic Approach and Data mining etc. to build problem solving, logical deduction and reasoning capabilities into machines, with limited success. On the other hand, movies such as Terminator (1984), Artificial Intelligence (2001), Matrix (1999) and Her (2013) etc. have stretched our imagination and fascinated us with possibilities of Artificial Intelligence.

Deep Blue chess playing system, IBM's Watson question answering system and more recently, intelligent person assistants such as Siri and Cortana are more close to an intelligent software. When, more and more intelligent applications become mainstream, the traditional software verification principles and practices will require a paradigm shift. Verification of these applications will present new and unique challenges. For example, a simple test case which is expected to return "FALSE" may work correctly today but may return "TRUE" tomorrow, because application learned and started behaving differently. Thus, tests will not be repeatable. Similarly, we may need to think differently while developing a regression test suite and designing automation framework. It may even lead to the test system being an intelligent system. When such intelligent systems integrate with intelligent sensors and/or web as a part of intelligent network, the testing becomes even more complex.

This paper is a commentary on testing challenges which we may face in verifying intelligent software. The purpose is to stimulate discussion on how traditional testing concepts may evolve to cater to intelligent software verification, without going into technicalities of artificial intelligence techniques such as neural networks, machine learning etc.

Biography

Satish H Yogachar *is a Senior Principal Engineer at Intel Security located in Bangalore. He has over 11 years of experience in Software product development. Prior to Intel Security, Satish was an Architect in product development in BFS domain at Tata consultancy Services and Symphony Teleca Corp. Satish holds Bachelor's degree in Computer Science Engineering.*

Anurag Sharma *is a Senior Principal SDET at Intel Security, Bangalore. He has over 9 years of experience in embedded as well as application software verification. Prior to McAfee, Anurag lead the Software Common Components verification team in SWCOE domain at Honeywell. Anurag holds Bachelor's degree in Electronics and Communication Engineering and MBA with specialization in Software Project Management.*

1 Introduction

As a general definition, an Intelligent System is a machine with an embedded Internet-connected device that has the capacity to gather and analyze data and communicate with other systems.

Intelligent systems exist all around us in point-of-sale (POS) terminals, digital televisions, traffic lights, smart meters, automobiles, digital signage and airplane controls, among a great number of other possibilities. Home Automation [1] Solutions such as Google Nest; Internet of Things (IoT) network [2]; Smartphones as well as wearable devices such as Jawbone, Fitbit etc. are all part of this intelligent ecosystem.

The Turing test [3] is a test of a machine's ability to exhibit intelligent behavior equivalent to, or indistinguishable from, that of a human. Today's intelligent systems may not pass the Turing Test, but these systems demonstrate intelligent behavior close to humanness in certain domains and will certainly get through the Minimum Intelligent Signal Test (MIST) [4], a less strict variation of Turing Test.

Throughout this paper, the terms "Intelligent System" and "Intelligent Software" have been used interchangeably, because, the word "System" is more relevant when referring to whole ecosystem of intelligent software. Still, the focus of this paper is on software testing aspects. "Conventional Software" terminology is used to describe the traditional software applications such as payroll processing software.

The content of this paper are organized as follows:

- Section 2 describes how testing an intelligent software is different compared to conventional software testing and challenges faced in doing so.
- Section 3 describes some approaches for verifying an intelligent software.

2 Why testing intelligent software is challenging?

An intelligent system is a combination of complex machine learning algorithms, contextual understanding, natural language processing and deep database represented in the form of knowledge graphs. Generally, an intelligent system initially boots up with a set of complex software algorithms and a limited input set. At this time, the capabilities of this system are limited. The longer this system is in service, more it learns about the user preferences, user behavior etc. based on user input patterns and other external stimuli: let's call it the *Learning Phase*. Thus, the database and knowledge graph of the system expands and it gains predictive capabilities, *the Analysis or Testing Phase*.

Let us consider, Google Now on Tap, an intelligent virtual assistant, as an example. When installed, it presents a few basic questions about the user like, home and office locations, hobbies and interest, etc. and the information provided by it is limited to route map and expected time of arrival at destination. Over the period of time, based on the interaction with the user, context of information and other linked google apps, the Google's knowledge graph about the user expands. Thus, it becomes more and more intelligent, tracking parcels for you, suggesting movies to watch, suggesting restaurants to order food and even making reservations by reading text messages from your wife, etc.

Testing an intelligent software is quite different compared to a conventional software. The conventional software testing principles and practices that we have known and understood for decades do not directly apply while verifying an intelligent software. Also, various software testing methods such as manual testing, white-box testing, automation, usability testing, etc. are either not applicable or have to be performed differently for an intelligent software. Each intelligent system offers unique testing challenges. Some of those are describe in the paragraphs below.

2.1 Psychology of Software Testing

Human beings are highly goal-oriented, therefore, setting the correct goal post has a psychological effect. Conventional Software Testing is the process of executing a program with the intent of finding errors. Testers select the test data which has high probability of finding errors. Equivalence class partitioning, boundary value analysis, etc. are used in designing the positive and negative test cases.

While testing an intelligent software, the intent is to demonstrate the learning ability of the system, to understand how the algorithm behaves, besides testing it. The test data and test cases designed are incremental in nature, such that the learning capability or pattern of the software is understood and verified. There is an implicit expectation that the software may legitimately behave differently (output) in each iteration.

It's an important distinction and it also implies that conventional testing is a destructive process while intelligent system testing has signs of a constructive process. This difference also has an impact on how test cases and test data are designed and how to test the program.

2.2 Test Case Design

Designing test cases for an intelligent software is difficult and more challenging in many aspects:

1. Usually, when input or output equivalence classes are applied to developing test cases for a conventional software application, the expected output for a given input is known in advance. Thus, it is pretty easy to design a test suite around it. For example, an MD5 secure hash algorithm [5] always produce the same 128 bit hash value for the same input. On the other hand, for an intelligent learning software, the expected output for a given input may not be known in advance. It depends on situational factors such as the context of the software at that point of time, the output from the previous tests and on the time when test is performed. For example, "Google Now on Tap" virtual assistant will produce different output for the same question "Who is the singer?" when different song is playing. Similarly, the answer to the question, "Who is the president of United States?" is different if the question is asked now or back in year 2007. Therefore, designing repeatable tests for an intelligent software is more challenging.
2. No reliable "test oracle" (or test suite) can be designed to indicate the correct output for any arbitrary input, therefore, it is challenging to detect subtle bugs. Although the repository of data-sets created over time can be used during regression runs to compare the result quality (code changes are not degrading the quality of output) but not for testing in strict sense.
3. As we know, testing a conventional software with all possible inputs is not always practical. For example, for a simple algorithm of identifying if a number is prime or not, the valid input dataset is infinite. Instead, by following equivalence class partitioning and boundary value analysis, a reasonable test suite capable of detecting any error in the algorithm can be designed. However, for a very basic intelligent software such as the SoundHound app (natural language processing engine), which detects the song currently playing, this task can be daunting.

2.3 Test Outcome

1. Conventional Software Testing is focused on the final output of execution. For an intelligent software testing, even though we may or may not know what final output should be, inspection of intermediate results with respect to an algorithm may reveal bugs. For example, Siri virtual assistant will not be able to provide answer to every question (perfectly) in near future, but, understanding how Siri software is breaking down the spoken question into chunks, how it is creating the knowledge graph etc. is an important step for fine tuning the software.
2. For an intelligent software, classification of output in a binary sense ("Pass" or "Fail") is not sufficient because given enough time and with a large number of executions, the probability is that every test will pass (*or fail, depending on how test vector is designed*). Therefore, the distinction between a positive or negative (boundary value analysis) test cases changes over a

period of time. Probably, a “Partial Pass” or quality of output may also need to be considered (See section 3.3).

3. Another major challenge is to figure out what test cases have best chance of identifying the errors. While performing tests for a conventional software, the requirements are generally well defined by the time software is available for formal QA validation. Therefore, the test cases with the best chance of revealing bugs, can be designed based on the requirements themselves. For a learning software, the intent is to identify the bugs in the specification, even before reaching implementation. Therefore, during the learning phase, the test data is designed carefully after analyzing the implemented algorithm such that the output can be predicted with high confidence. The expected output during execution of the algorithm ensures that the implementation follows the algorithm correctly, and any deviation can be due to the incomplete/incorrect specification of the real-world problem.
4. Most of the time, when interacting with an intelligent software, a user doesn’t have time to scan through all the available output results, therefore test scenarios should be designed to ensure that the best answer is reported (something like, Google “I am feeling lucky” answer [6]). For example, while driving you ask “OK Google, what is the route to reach A from my current location?” Then you expect that google will provide the best route (shortest distance, less traffic etc.) out of all available routes.
5. Results of an intelligent software testing are open to interpretation because of the way the requirements are defined, the number of features offered, how the tests are conducted and what dataset was used.
6. For conventional software, desktop as well as web applications, the minimum quality metrics in terms of response time, memory footprint, etc. are well established. There are standard tools available to measure these quality factors. Intelligent software are still evolving, therefore, no such standards are in place. Therefore, the results of any such tests are more prone to being biased.

2.4 Security and Privacy

In conventional software testing, the software has well defined entry points and user inputs. Most software uses industry standard cryptography algorithms for user authentication, confidentiality, encryption and integrity of data in transit/at rest. The security testing follows standard processes of vulnerability scans using standard tools, threat modelling, security assessment, security audit and security review. Privacy testing includes review of user data storage/retrieval methods (credit card information, user passwords etc.).

For intelligent software flooding the market today are built on ideas such as:

1. Proactively provide information, even before user asked for it (Google Now, Microsoft Cortana), or
2. Make tasks easier by integrating/eliminating steps from the process (For example, Apple Pay, Google Wallet for quick checkout at the store), or
3. Recommend things to buy based on previous search history, or
4. Connect the dumb devices sitting around home (like coffee maker, refrigerator, TV, locks etc.) with the internet/mobile so that these devices can do things automatically according to user behavior (Internet of Things).

To make it possible, these software apps are continuously connected to Internet and upload data to the cloud. Moreover, they collect much more data about users such as location (GPS), social friends’ contact information (Facebook contacts, LinkedIn contacts etc.), addresses, credit card information, previous shopping lists, access to e-mails, messages. It is easier for hackers to steal data or to commit fraud online by exploiting these vulnerabilities or through social engineering. Users are also becoming aware about the security risks. A recent backlash against Facebook’s privacy policies is one such example [8].

Today, with millions of internet connected devices which continuously uploading user data to cloud; traditional firewalls at the datacenters are not enough. Application design needs to consider data security

and user privacy upfront, not a side activity. Similarly, test plans should emphasize security testing as much as functional testing.

2.5 Automation

Automation designed for an intelligent software should also be intelligent in certain aspects.

1. It should be capable of not only generating test cases to cover branches and paths, but also capable of understanding the relationships between the inputs and outputs so as to simulate real world behavior.
2. It can simulate the large number of external inputs such as user actions, GPS location, network, etc.
3. Intelligent applications are inherently non-deterministic in nature, similar to us, the real people. While making decisions, we correlate various stimuli, behave accordingly. Therefore, to test intelligent applications, automation should be able to generate various sequences and permutations of inputs and to randomize the inputs, so that software correctness can be investigated in more detail.

2.6 Programming Languages and Test Tools

Intelligent software developers prefer specialized languages for development [9], which are strong in mathematical notation of computer programs, declarative programing, symbolic reasoning and language parsing, for example, Lisp, Prolog, Haskell, and MATLAB, etc. Experienced test engineers for these languages are in short supply.

Available test tools do not exactly apply when testing intelligent software. Conventional tools such as memory leak analysis tools, static analysis tools, unit test tools, etc. may be applied partially. But, most of the tools or scripts such as test data generators, tools to compare output data models, etc. need to be designed specific to each application under test.

2.7 Localization

Localization refers to the actual adaptation of the product for a specific market. It involves language translation, adapting graphics, adopting local currencies, using proper format for date and time, addresses, and phone numbers applicable to the location, the choices of colors, and many other details, including rethinking the physical structure of a product. All these changes aim to recognize local sensitivities, avoid conflict with local culture, customs and common habits. User facing portions of the software such as log messages, the user interface, etc. are candidates of localization. Localizing any product and its testing is a highly technical process. When it comes to intelligent software, especially the virtual assistants on mobiles (i.e. Apple Siri) which rely on spoken or typed natural language processing, it is still more challenging.

3 Test Approaches

Few approaches for testing intelligent software:

3.1 Testing by End User

Beta testing is very popular in the application software world. For example, beta versions of new operating system versions are released periodically for Microsoft Windows and Apple OS. Early adopters and developers, even enterprises share their feedback and bugs to Microsoft, which ultimately boils into the final release to general public. This approach is most suitable for intelligent software testing where technological boundaries are being pushed.

Testing by end users helps in understanding the use cases of new breakthrough innovations being developed and to generate curiosity among general public. Google Glass [10] Explorer program is a prime example of it. Google Glass is nowhere near the final product, but at 1500 dollars apiece, it was given to technology experts for obtaining the reviews, understanding the use cases and to acclimate the world with this new way of interacting with this system.

Testing by end users is also adopted for scenarios where it is not feasible for the developers to perform exhaustive testing because of infinite number of combinations. For example, you cannot test Google Maps, "Get Directions" feature for every possible route in the world (and where each input source and destination combination has multiple route options). Therefore, once it is tested with a very large sample set against a reference model and all obvious issues are fixed, it is rolled out to users for User Acceptance Testing.

3.2 Equivalence Classes of Test Design

While designing the test datasets, the software requirements and functionality of the real world application (to be emulated by intelligent software) should be kept in mind. Partitioning the input data into various classes helps in focusing testing and is also useful for automating the tests. For example, test datasets can be divided into:

1. Predictable Output Behavior vs. Unpredictable Output Behavior datasets. Predictable output behavior is where the answer is always known beforehand such as a mathematical calculation or a well-known fact such as "*Who was the 16th President of United States?*" On the other hand, unpredictable behavior is where the output cannot be decided on previously known information and requires modelling and analysis of facts and figures such as "*Who will win the 2018 FIFA World Cup?*" A predictable behavior test dataset covers the natural language processing and search capabilities of the software, while an unpredictable behavior dataset verifies the mathematical processing capabilities of the intelligent software.
2. Incremental Learning Behavior vs. Random Input dataset – A test dataset designed to verify the incremental learning behavior of the software takes into account recent history to make its predictions. A good example of an incremental learning software is the weather forecast; if it has been sunny and 80 degrees for the last 2-3 days, it is a pretty reasonable prediction that it is not going to snow tomorrow. In this particular case, the underlying data source is not changing. Another example is the sales model of a retail chain where the predictive model has to evolve as the company grows. For an incremental learning software, Data Horizon (how quickly new input data should be considered) and Data Obsolescence (when old data is still relevant for prediction) are important factors, which should be kept in mind while designing the test dataset. On the other hand, a test dataset designed to verify a computation search engine "Wolfram Alpha" [11] will cover a broad range of inputs covering various topics such as statistics, astronomy, mathematics, finance, etc.

3.3 Ranking the Output

As described in section 2.3, the output of a test cannot always be a "Pass" or a "Fail". Therefore, a ranking system should be developed for ascertaining the quality of the output. Most appropriate results can be ranked higher, while the most irrelevant results can be ranked lower. The testing should verify that the best result based on this ranking is presented to user.

For example, Google Maps may suggest the subway is the best method (fastest) to commute from point A to B in New York City, while in any other city, a cab/taxi may be ranked higher. Similarly, Google maps updates the best route ranking using of external factors such as diversion due to construction work or traffic situations. The reference model and the sample test dataset designed for Google Maps' "Get Directions" feature should handle and verify the correctness of this ranked output.

3.4 Comparison Test

One of the most common approaches to evaluate two or more intelligent software apps is by pitting them against one another. For example, tests such as these have been conducted to find out which of the iOS and Android personal assistants (namely, Siri and Google Now), is better [12]. This can be a very effective test when the test dataset is very large and different areas of software features are covered.

4 Conclusion

This paper is an attempt to demonstrate how traditional software testing is becoming more challenging with the increasing sophisticated software applications. Increasing sophistication demands rigorous software development and testing. Software testing has to evolve beyond just functional testing and cover other aspects such as security, user behavior etc.

In a thought provoking video “Humans Need Not Apply” [13], CGP Grey argued that every profession that relies on creativity, decision-making, and pattern-recognition could eventually get replaced by Artificial Intelligence. However, we firmly believe that the future of software testing is secure and brighter since more complex automation (Artificial Intelligence) doesn’t cause automation, it causes more testing (Human Intelligence) needs.

References

1. Wikipedia article on “Home Automation”, [Online] https://en.wikipedia.org/wiki/Home_automation (accessed July 5, 2015)
2. Wikipedia article on “Internet of Things”, [Online] https://en.wikipedia.org/wiki/Home_automation (accessed July 5, 2015)
3. Wikipedia article on “Turing Test”, [Online] https://en.wikipedia.org/wiki/Turing_test (accessed July 5, 2015)
4. Wikipedia article on “Minimum Intelligent Signal Test”, [Online] https://en.wikipedia.org/wiki/Minimum_intelligent_signal_test (accessed July 5, 2015)
5. Wikipedia article on “MD5”, [Online] https://en.wikipedia.org/wiki/MD5 (accessed July 14, 2015)
6. Wikipedia article on “Google Search”, [Online] https://en.wikipedia.org/wiki/Google_Search (accessed July 18, 2015)
7. “An Approach to Software Testing of Machine Learning Applications” by Christian Murphy, Gail Kaiser and Marta Arias [Online] Available at http://www.psl.cs.columbia.edu/publications/pubs/Murphy-SEKE2007.pdf
8. “Facebook’s new privacy policies and your data security”, Avast blog entry posted on December 11, 2014, https://blog.avast.com/2014/12/11/facebooks-new-privacy-policies-and-your-data-security/ (accessed July 5, 2015)
9. Wikipedia article on “List of programming languages for artificial intelligence”, [Online] https://en.wikipedia.org/wiki/List_of_programming_languages_for_artificial_intelligence (accessed July 5, 2015)
10. “Project Glass: Live Demo At Google I/O”, https://www.youtube.com/watch?v=D7TB8b2t3QE (accessed July 5, 2015)
11. Wolfram Alpha computation search engine [Online] - http://www.wolframalpha.com/
12. “Next-gen Siri versus Google Now on Tap: The battle to read your mind best”, CNET blog entry posted on June 9, 2015, http://www.cnet.com/uk/news/proactive-siri-versus-google-now-on-tap-compared/ (accessed July 5, 2015)
13. YouTube video “Humans Need Not Apply” [Online] https://www.youtube.com/watch?v=7Pq-S557XQU (accessed July 5, 2015)
14. “Will Software Testing Ever Succumb to Artificial Intelligence?”, posted on November 22, 2014, http://www.testuff.com/blog/will-software-testing-ever-succumb-to-artificial-intelligence/