# Improving Verification Predictability and Efficiency Using Big Data

Darron K. May

Mentor Graphics Inc.

8005 S.W. Boeckman Rd.

Wilsonville, OR 97070

***Abstract-Big Data* is a term that has been around for many years. The list of applications for Big Data are endless, but the process stays the same: capture, process and analyze. So why shouldn't this technology help improve your verification process efficiency and predict your next chip sign-off? By providing a Big Data infrastructure, with state of the art technologies, within the verification environment, the combination of all verification metrics allows all resources to be used as efficiently as possible and enables process improvements using predictive analysis. This paper will cover the technology, the metrics, and the process, and it will explore a number of techniques enabled by such an infrastructure.**

## I. Introduction

*Big Data* is a term that has been around for more than 20 years. It was initially defined as data sets captured, managed, and processed in a tolerable period of time beyond the ability of normal software tools. The only constant in Big Data's size over this time is that it's been a moving target driven by improvements in parallel processing power and cheaper storage capacity. Today most of the industry uses the 3V model to define the challenges and opportunities of Big Data as three dimensional: volume, velocity and variety. Lately this has been expanded to machine learning and digital footprints. The list of applications are endless, the process is the same — capture, process and analyze. Why shouldn't this technology help improve your verification process efficiency and predict your next chip sign-off?

Today's verification environments have to be collaborative due to the size of devices, geographically dispersed teams, and pressures on time to market. It requires the efficient use of every cycle, managing hardware, software, and personnel resources.

This paper will define the typical verification environment and the data that it often leaves uncaptured across the period of a project. It will show how the process of capture, process, and analyze can be applied to improve predictability and efficiency of the whole verification process. This requires a flexible infrastructure that allows data to be extracted from the multiple systems that make up the typical verification flow. There must be a central repository that is able to store the data in a common way, so that data can be managed to keep it clean and relevant not only over the project's duration but also into the future to allow comparisons and predictions on other and new projects. This paper will explain how new web-ready technologies can be applied to the hardware development flow to provide a plug-n-play infrastructure for collaboration. It will also highlight some of the types of analysis and insights that are possible by combining common coverage metrics with those normally lost data metrics, as well as inter-relationships between those metrics.

The ability to see gathered metrics over time can provide great insights into the process. Historical coverage data trended over time alone can give indications of how much more time is needed to complete sign-off. Being able to plot these single metrics together on the same graph also opens up information that is often lost. This paper will also show with examples other insights that can be gained by looking at cross-analytics between bug closure rates and source code churn, which when combined with coverage metrics can help predict progress towards sign-off. It will show how historical data can be used to spot similar patterns of events, and how recording a little more meta-data within existing systems allows cross-analytics to figure out information like how effective a new methodology or tool has been based on past projects. It also allows calculation of further metrics available from the data; such as mean time between bug fixes, mean time between regression failures, the last time a test passed or failed, and tool license use by specific users — allowing us to answer and predict questions like "Will we have enough formal licenses for our peak usage on the next project?"

## II. What is Big Data?

The definition of Big Data was always targeted towards sets of data so large that traditional processing software is inadequate to deal with them in a useable amount of time. Not only was there the amount of data but also the diverse types of data of which it is unclear, at the point of collection, how they will be useful when analyzed. Fundamentals include the storage of unstructured, semi-structured and structured data. The main focus is normally on unstructured data.

This tends to leave the opinion that the well understood verification process with its well-understood data generation is not really Big Data. In a 2001 research report [1] and related lectures, META Group (now Gartner) defined data growth challenges and opportunities as being three-dimensional. These three dimensions were known as the 3Vs: increasing volume (amount of data), velocity (speed of data movement), and variety (range of data types and sources). This definition was later updated as it is not only the data but the technology and methods used to transform its value that matter. The additions include machine learning to detect patterns and make predictions and digital footprint, which is a cost-free by-product of digital interaction. The definition has changed but there is no doubt of its impact on our everyday life.

Companies like Amazon, Facebook, Twitter, and Google all make considerable revenue from capturing and analyzing our data. For example, Amazon captures and stores our digital footprint, processes the data, and then uses predictive analytics. Predictive analytics are a part of machine learning and is one of the major fundamentals that make the concepts of Big Data extremely interesting when applied to the verification process.

Many industries are being transformed by Big Data analytics and machine learning. Many of these are focused on making larger revenues using predictive analysis to predict such things as the next big music hit, the right genre for the next big TV show, or when to renew a phone contract. However it is also being used to benefit other aspects of our lives, like controlling energy generation by monitoring user habits and being kinder on the environment by predicting peaks and troughs. It is also being used in healthcare where it is possible to use mobile phone and social media usage to pin-point the outbreak of disease in Africa. It is also used in law and order where, using years of history and patterns of past crime data in inner cities, it's able to predict the time and location of future crime — quite "Minority Report"! Every one of these examples follow the process of capturing data in one central place, processing and maintaining good housekeeping of the data, and employing various methods of analysis — again, mainly involving predictive analysis. All of these concepts can be easily applied to the data within the verification environment.

## III. The Typical Verification Environment

Verification is typically driven by the testplan document. This testplan is derived from the design specification and defines how to test or verify design features. These features are typically tested by either a directed test or by some form of coverage or combination of coverage that records when a particular feature has been tested. Implementation is driven by the design specification, and along with the stimulus in the form of testbenches, it is typically kept in a Source Code Management (SCM) system. A build and regression management system is used to run various verification tools to, first, locate areas of the design that do not meet the specification (i.e., bugs) and, second, ensure that all functionality within the specification has been covered.

Bugs are tracked within a bug tracking system, and coverage closure is achieved by measuring both code and functional coverage and cross-referencing the testplan. All of the systems and stages mentioned above produce data. Some of these sources are not typically gathered during the verification process.

The process described above also requires both hardware in the way of compute resources and licensed software, both of which need to be used efficiently. The compute resources are typically controlled by a load-sharing system, and the licenses by a licensing system. Again, typically data from these sources is not gathered over the period of the verification process. A complete picture of a typical verification process and the systems that produce data can be seen in Figure 1 – The Typical Verification Environment.

There are many other metrics that could be gathered during the duration of a project if there were an infrastructure to store them over time, some of which are covered in other papers [2]. Some of the systems described above have facilities to store information over time and maybe even analyze the data over time, but this only allows the metrics to be seen in isolation. With projects and teams spread geographically, what is required is a system that allows all of this data to be saved in a central repository that is visible in all locations. This allows collaborative authenticated access to the combined data and the ability for multiple projects to use the same resources. By gathering data across many projects over a period of time, the data value will improve just like any of the Big Data applications talked about in the last section. This will allow predictive analysis techniques to be used both within a project and across projects to help improve the process.

There are a number of statistics on general data growth. IBM stated in 2013 that 90% of the world's data had been generated in the last two years. Other newer reports state that data is growing now by up to 40% to 50% every year. Generally within the verification space only a small fraction of the generated data during a project is saved, with some companies saving more than others. However, without an infrastructure similar to the one explained in the next section, too much of the data detail is lost — data that could be so valuable to an organization in predicting and helping to drive future projects.

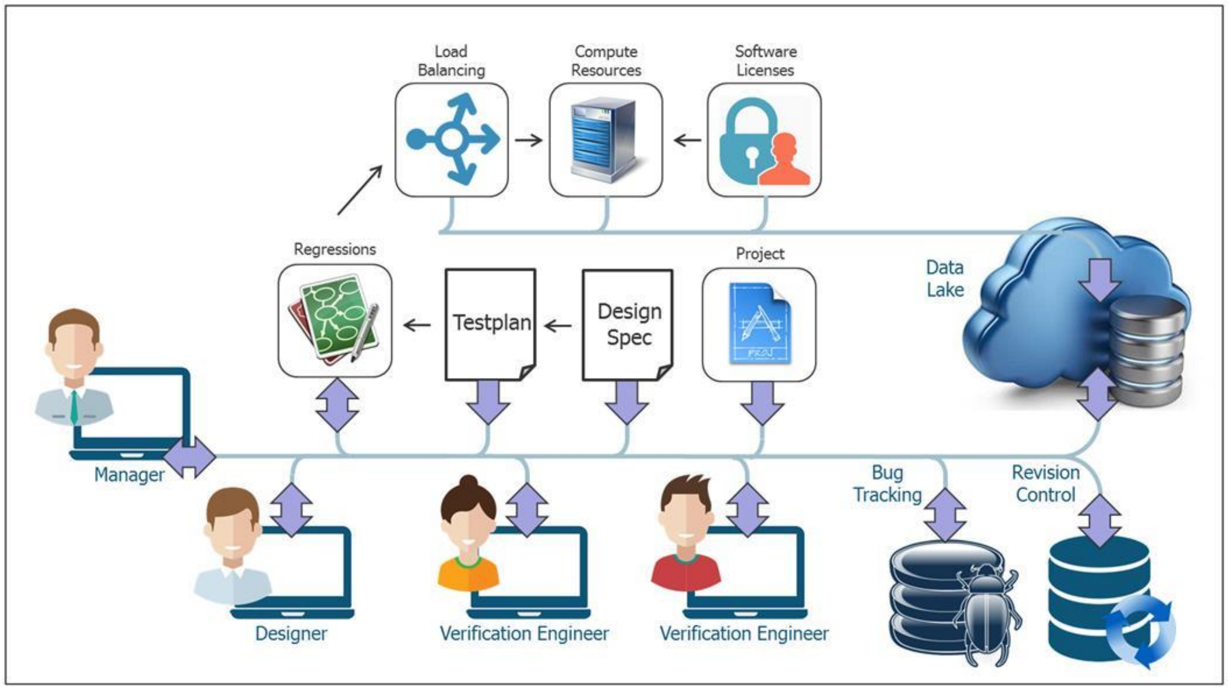

**Figure 1 – The Typical Verification Environment**

## IV. New Enabling Technologies

One of the major requirements when harvesting vast amounts of data is to have somewhere to store that data. Within the verification process some of the individual systems have a persistent store or repository, allowing a collaborative use model — for example, bug tracking and revision control. Others store data for use in that domain only; such as coverage, or documents like a testplan and design specifications.

Data management is the key requirement for this system, and there is a huge choice of database infrastructure. Server structured databases — such as Microsoft SQL Server, Oracle, the open-source PostgreSQL, and IBM DB2 — contain mechanisms to ensure the reliability and consistency of data and are geared toward multi-user applications. These databases are designed to run on high-performance servers and carry a correspondingly higher level of investment. The benefits achieved through the use of a server-based system include flexibility, powerful performance and scalability.

With the need to manipulate large sets of complex data, some of which are unstructured or semi-structured, "NoSQL" databases have become more widespread. A NoSQL database is not structured on the common column-row design of traditional relational databases, but rather uses a more flexible data model. The model varies, depending on the database. Some organize data by key/value pair, graphs or wide columns. Common "NoSQL" databases include MongoDB, Cassandra, CouchDB, HBase, and Redis.

It’s not possible to discuss Big Data technologies without mentioning Hadoop. Apache Hadoop describes an entire ecosystem of distributed computing tools centered on a file system (Hadoop Distributed File System) and a resource negotiator. The core components are based on papers authored by Google and were created at Yahoo! Hadoop runs applications using the MapReduce algorithm, where the data is processed in parallel on different CPU nodes. Data is distributed and tracked as multiple copies stored over a cluster built from commodity hardware. Data stored on a server that goes offline or dies can be automatically replicated from a known good copy.

Enterprise databases require heavy initial investment in resources, time and money. This means that a generic backend infrastructure must give the flexibility to choose the database technology but still provide security, authentication and maintenance. Once the backend infrastructure is in place, the major investment involves building the business logic to achieve the required analysis and views. The major blocks of the system can be viewed in Figure 2 – Technology Stack Block Diagram.

The frontend is another major requirement for collaborative analysis and visualization of the data. Again there are many new technology stacks giving web ready visualization. HTML5 is often used to refer to a range of modern web technologies. Simply speaking, HTML is the language that the web is written in, and HTML5 is the most recent version of it. But because there are a lot of other technologies commonly used to create rich experiences on the web (such as CSS3 and JavaScript), those get wrapped up into the abbreviation *HTML5*.

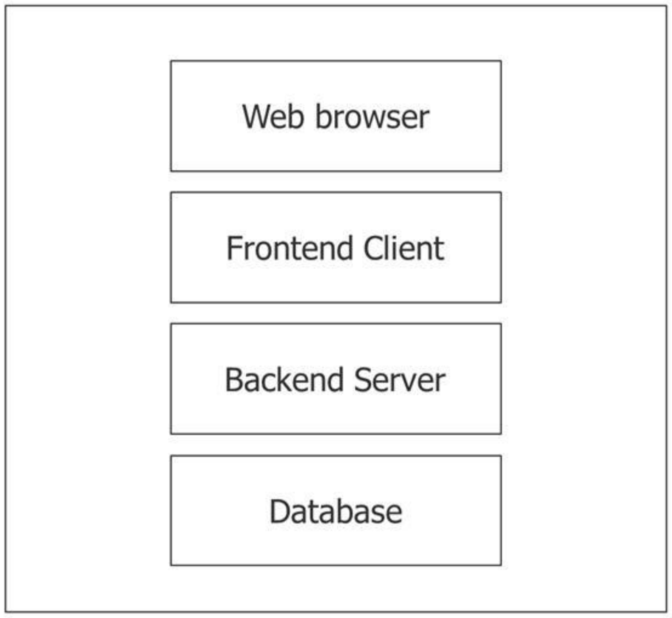

**Figure 2 -Technology Stack Block Diagram**

There are many JavaScript libraries to chose from to implement the frontend client. These include Angular 2, originally developed by Google, React, originally developed by Facebook, and Bootstrap, originally developed by Twitter. It is also possible to run a library — such as Node.js, which generates dynamic web page content on the server side — before sending it to the client. All of these libraries are extensively used in web applications and have other libraries built upon them to provide all kinds of visualization widgets, graphs, tables, gauges and more.

As more companies realize the benefits of implementing business intelligence software, there has been an increase in the number of vendors offering dashboard software. Dashboards provide a visual overview of an organization’s data and analytics in an interactive dashboard that displays key performance metrics in various charts, graphs and animations. This makes it possible to create visualizations of your application-specific data without the need, in most cases, for software language skills in web technologies.

There are also many systems that allow web solutions to be developed using scripting languages that hardware designers are more familiar with, allowing home grown processes to be more easily developed. Two such systems are “Ruby on Rails” [3] or Python and Django. From a technology standpoint, there is no clear winner between these two. They have similarities: both are server-side frameworks, each has a community supplied set of extensions to help the building of web applications, and both are popular and trusted frameworks in the programming world. The choice would probably come down to the preferred scripting language. One such system, developed for the purpose of metrics gathering in the verification environment and using Ruby on Rails, is detailed in [4]. It is beyond the scope of this paper to recommend which of the many technologies should be used to implement a Big Data system within the verification process, but a commercially available tool would need to support multiple types of databases and be web-ready using the latest technology stacks.

## V. Single And Cross Metric Analysis

Once one of these systems is in place to store data from the verification process, extractors are required to import data from all the relevant sources. The system needs the ability to run these extractors either periodically or on the occurrence of certain events.

For example, samples of the loading statistics of compute resources need to be taken on a regular basis (at least every minute, maybe several times a minute) from a grid system or from the licensing system to monitor which license features are currently in use. On the other hand, samples from systems like source code management need to be taken every commit, and samples from a regression system must be taken every time it is run, which could be either periodically for nightly runs or event-driven for commits for smoke tests. The extraction system needs to be flexible to achieve these requirements and must be adaptable to mine data from many sources. A table of the sources and examples of the type of metrics that can be extracted for the typical verification process is shown below.

**Table 1- Extracted Metrics from Individual Sources.**

| **System** | **Metrics** | **Meta Data** |
|---|---|---|
| Project | Milestones, tasks. | Per project, per user. |
| Requirements & Testplan | Design Requirements.<br>Verification Requirements.<br>Verified Requirements. | Per project, per user. |
| Source Code Management | Total lines of code.<br>Functional lines of code.<br>Comment lines of code.<br>Whitespace lines of code.<br>Code Churn. | Per project, per user, per file, per instance, design unit or test bench. |
| Regression | Number of Tests, Passes, Failures.<br>Queue times.<br>CPU times.<br>Run times.<br>Mean Time Between Regressions. | Per project, per user, per test. |
| Grid System | Host loading.<br>Host memory. | Per project, per user, per host. |
| Licensing System | Features checked out. | Per project, per user. |
| File System | Storage used.<br>Capacity. | Per project, per user. |
| Coverage | Code Coverage.<br>Functional Coverage.<br>Testplan Coverage.<br>Number of bins. | Per project, per user, per instance, per design unit, per file. |
| Bug Tracking | Open.<br>Closed. | Per project, per user, per tool, per instance, per design unit. |

|  |  |  |
|---|---|---|
|  | In-progress.<br>Mean Time Between Fixes. |  |

Trending, or looking at time-series line graphs of a number of the metrics detailed above, can give a very good indication of how much more time is needed to complete sign-off. The obvious metrics are code or functional coverage plotted over the period of the project, where the finish point is as close to 100% as possible.

Bug closure and code churn are two other metrics that give a very good prediction of where you are in the verification process. The bug's closed over the period of a project will follow a standard S-curve; whereby early in the process many bugs are closed in a shorter period of time until it slows to fewer bugs closed towards the end of the process. This S-curve relationship is also followed with the code churn rates, where more code is touched and added in the early parts of the project, while towards the end the code base stabilizes. Studying these metrics across projects, this relationship can be used with very high levels of accuracy, which means the sooner they are gathered the sooner they can be used for future predictions.

Insights are also possible by combining metrics within the same graph to help produce a more accurate picture. This can be done by using current and past data or by looking at metrics from different sources. Plotting both code coverage and functional coverage on the same time series scale tells us more about what is happening in the process. If code coverage is high but functional coverage is low, it's a good indication of the need for more tests. However if code coverage is low but functional is high, it may mean that either the functional coverage model needs enhancing or that there are unused or unnecessary areas of the design. Furthermore, if coverage percentages are viewed along with the bin counts, its possible to see that coverage numbers have decreased because of additions to the coverage model. This is only possible when there is an infrastructure allowing more than just coverage percentages to be saved over the duration of the project.

With all relevant metrics stored in the same place, smart functions can be built to allow data to be processed dynamically for generating up-to-the-minute views of the data as well as notifications of when certain events occur. For example, visualizations can be built to constantly monitor metrics such as hardware and software resource usage. These visualizations include things like league tables that show the current license features being used by individuals or project teams, or a gauge to show the current disk space usage or compute resource loadings. By building a rules-based notification system, the user could also set up automatic monitors to show events on a news ticker or send an email to a particular person when good or bad events happen. For example, the system can record a drop in a certain coverage metric of more than a particular amount over a particular period of time, or the fact that the lifetime of a particular bug has gone over a certain period of time.

These smart functions should also be able to do data house-keeping to keep the data clean and relevant. For example, down-sampling techniques can be applied to data windows for certain metrics. Thus, when sampling resource metrics at a rate of one to several times a minute, the data can be down-sampled, keeping the trends but reducing the storage needs.

With such a data platform, domain-specific analysis is also possible, introducing deeper analysis than provided by its host system of the connected source data. This adds extra value and insight. For example, by adding extra meta-data to the source code it is possible to calculate functional, comment, and whitespace lines of code for either the RTL or HVL code. It is also possible to calculate the lines added, removed or modified by each user.

There are other mining techniques, used in software development, that can be used to learn how source files end up changing together. This can be used to point to the cause of a potential problem. For example, analysis can tell that each time a particular source file changes there is a percentage chance that other files will change. If there is no reason particular files should change together, the analysis points to a part of the code worth investigating as a potential target for a future refactoring.

Adding extra meta-data to systems (for example adding the tool used to find a particular bug or the instance of the design where a bug is found) allows further analysis of the standard data. Calculating the time each bug spends in-progress for each verification method or component makes it possible to show the mean time between fixes for each component for those methods and compare which is the most effective. Mean time between state changes in many of the metrics can be very useful to get more insight into the process, using both the current value or trended values over time. This includes changes in test results, regressions from pass to fail or vice versa, and even changes on the state of a covered bin.

Big Data is not just about harvesting vast amounts of data, but more about the different relationships you can start to see when you combine data. The insights derived from the data help make better, faster decisions within projects and across projects. This process can be improved by placing cross references between metric domains — for example, by adding the bug ID reference that has been fixed to a commit into the source code revision system and recording the commit or revision that a particular bug was found in and resolved to the bug tracking system. This now allows not only cross analysis where it is very simple to recreate a particular error scenario, but also allows deeper analysis into relationships between the bugs and the code base and the methods used to fix and locate the problems.

The ability to look at these relationships is not only useful within the project, but the ability to analyze this data across multiple projects opens up opportunities to learn more from the data that would normally be lost. Other good examples of this includes looking at the license history across projects, allowing patterns of usage and peak usage to be compared with other metrics such as increases in coverage, or the opening and closing of bugs to figure out where and when adding extra licenses would improve productivity. The next section covers more in depth analysis techniques.

## VI. Data Mining And Predictive Analysis

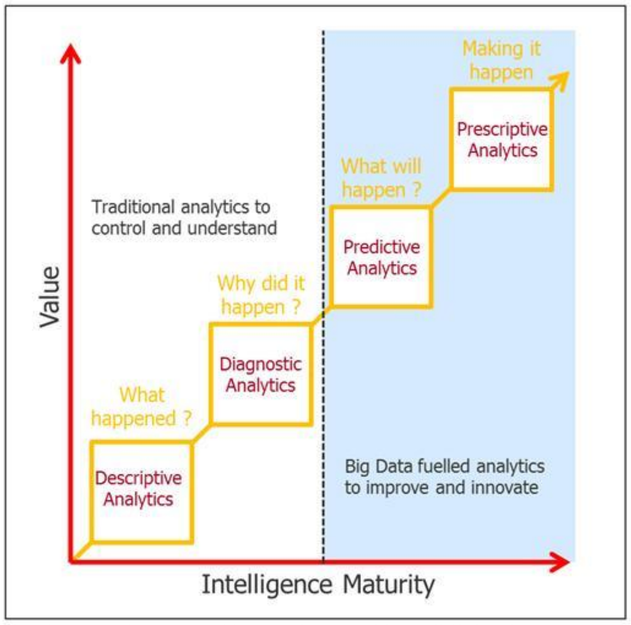

**Figure 3 – Analytic Maturity Model**

Maturing data analytics is driving the demand for automated decision-making based on reliable predictive analytics. Figure 3 highlights the maturing of data and analytics, illustrating how we have moved from descriptive and diagnostic analytics to predictive and prescriptive analytics, resulting in the unleashing of the full potential of data.

Descriptive analytics, by way of reports and dashboards, describe the dimensions and measurements of any aspect of the process, enabling users to report on goals and actual values of any metrics.

Diagnostic analytics use more advanced and critical capabilities, such as interactive visualization, to enable users to drill more easily into the data to discover new insights. Here the user is assumed to be more analytical, using the tool provided to mine the data in the desired way.

Predictive analytics is used after we become more mature in diagnostic analysis and more capable of identifying root causes for any metrics data variances as well as the variables that can predict what the measure will be in a future period. Some examples of the first three analytic types have been given in this paper, but prescriptive analytics, which uses insights from the predictive models, are integrated into processes to take corrective or optimizing actions. This section highlights some of the analytics that fall into the more mature area of this graph.

The debug cycle, and where to start it, are two of the most time consuming aspects within the verification process. With all coverage and assertion data in one place, it is possible to utilize the data from regression runs to suggest starting points for debug.

The basic principle is that if all tests that cover a particular coverage point passed, the chances of that coverage point being the cause of an error is very low. If all tests that cover a given coverage point failed, and none of the tests that did not cover the same coverage point failed, the chances of the coverage point in question being the cause of the error is very high. Taking tens of thousands of these coverage points in a typical design, it is possible to pinpoint likely causes of the failures.

This algorithm works along with a triaging process to pinpoint a single error as a starting point. The assumption is that we require significant numbers of passing and failing tests. The tests should not produce 100% coverage, and the coverage of the various tests should have minimal overlap with coverage from other tests. Essentially, what we want to calculate for each coverage point is the probability of the result of a given test being a failure if a given coverage point is covered by said test. A rating can be computed for each coverage point which tells us how much additional information about the overall pass/fail status of the tests is contributed by each coverage point, and how closely does the signature of a given coverage point match the pass/fail signature of the regression suite. The coverage point with the closest correlation to the pass/fail status signature is probably the coverage point closest to at least one cause of the failures. This is an example of what is possible in data mining when all information from the regression is gathered in one place. This approach has been patented [5].

## VII. Conclusions

Big Data is not just about harvesting vast amounts of data but also allowing the exploration of the different relationships you can start to see by combining data captured from the verification process. We've looked at the typical verification process and the systems that are used which provide many useful metrics, some of which traditionally often go uncaptured. Using some of the readily available technologies, it is possible to start to gather very valuable data, which can help today's and more importantly tomorrow's projects. Data mining and predictive analysis can be applied to the data within the verification process. Providing a Big Data infrastructure or platform with state of the art technologies for the verification environment allows the mixture of all verification metrics and can truly allow all resources to be used as efficiently as possible, significantly improving the predictability of the entire process.

## Acknowledgment

Thanks to Joe Larabell of Mentor Graphics for providing information and guidance on his patent and for reviewing this paper.

## References

[1] Laney Douglas, "3D Data Management: Controlling Data Volume, Velocity and Variety" Garner. Retrieved 6th February 2001.

[2] Andreas Meyer and Harry Foster, *"Metrics in SoC Verification"* DVCon 2012.

[3] Joe Francis, Brooke Patterson, "Ruby on Rails: The What's, Why's, and How's" AJLA-TS. February 2014.

[4] David Crutchfield et al. "Regressions in the 21st Century – Tools for Global Surveillance". DVCon 2016.

[5] Joseph Larabell. "Design Verification Test Failure Analysis". US Patent 2010.