This is the authors' version consisting of a pre-print of an article published in IEEE Software in 2019. The final authenticated version is on: https://doi.org/10.1109/MS.2019.2933681

Data Science - Technologies for Better Software

Christof Ebert
Vector Consulting
christof.ebert@vector.com

Jens Heidrich, Silverio Martínez-Fernández, Adam Trendowicz
Fraunhofer IESE
{jens.heidrich|silverio.martinez|adam.trendowicz}@iese.fraunhofer.de

Data Science rapidly gains relevance in software teams. Data science methods facilitate analyzing and predicting different quality aspects of the software product, process, and usage by combining results from different data sources such as versioning systems, issue tracking systems or static code analysis in an efficient way. Authors Jens Heidrich, Silverio Martínez-Fernández, Adam Trendowicz and I provide industry insights in using data science for improving software development. I look forward to hearing from both readers and prospective column authors about this column and the technologies you want to know more about.

—Christof Ebert

1 Introduction

Without data science, companies can't get value from their own achievements and assets. With the convergence of embedded and IT domains, such as Internet of Things (IoT) and automotive systems, the complexity of software systems is escalating. Complexity has two faces. On one hand it means more functionality, fluid delivery models and thus more value to the markets such as single customer focus. Complexity also means a growth of technical debt which decreases productivity and quality. As software engineering generates ever larger and more varied data sets, such as feature usage, code analysis, test coverage, error logs and maintenance data, companies face the challenge of unlocking the value of that data,

Those that are failing to effectively apply data science in their software engineering processes are increasingly putting themselves in a competitive disadvantage. Classical software measurement has played a key role for managing the complexity of software processes and products in the last decades but reaches its limits because of the ever-increasing amount of available data and their dependencies.

With embedded software becoming increasingly pervasive and critical to our society, engineers must ensure that their software performs as intended and doesn't fail. Delivering defective software has an impact in many areas: prestige, economic, customer experience. In the case of real-time software, it can also have an impact on the health and integrity of human beings. Software defects can have dramatic consequences specifically with respect to safety and cyber-security. So, the manufacturers of safety-critical products have long recognized the need to deliver secure, reliable, high quality software.

All software systems across domains and applications exhibit one clear trend: Complexity like cancer grows. To stay in control, we must manage intended complexity and reduce accidental complexity. Software measurement has played a key role for managing the complexity of software processes and products in the last decades [1]. Nowadays, short development cycles as introduced by recent development paradigms, such as agile development or DevOps, increase the opportunity to get vast amounts of data in relatively short time and learn from that data during the project as well as transfer this knowledge across projects.

Thus, being able to deal with Big Data and make use of intelligent algorithms, such as machine learning and deep learning for analyzing the data becomes more and more important [4]. Even though there is an increased amount of powerful (open source as well as commercial) tools and infrastructure available, a fool with a tool is still a fool. So, modern software

measurement requires profound knowledge in data science including which methods and techniques to use in which situation and which steps to follow for collecting, analyzing, and evaluating data.

The goal of this article is to demonstrate how software engineers benefit from Data Science approaches based on two practical cases, namely

- data-driven quality management
- automatic post-processing of code analysis.

First, we will give some background information on the term Data Science, then we will give details about how Data Science approaches were used in the two cases following the CRISP-DM process standard for data mining. Finally, we will summarize the lessons we learned and give recommendations for practitioners.

2 Background

According to [2], Data Science deals with the way in which very large amounts of data are collected, processed, prepared and analyzed. [3] states that data science is a cross-domain discipline that requires in-depth basic computer science and mathematics skills as well as expertise regarding the application domain). However, in practice the term "Data Science" is often used as a buzzword addressing a variety of fields of expertise in different depths. However, one central consensus is that it is about expertise in dealing with big data (i.e., an increased volume of data, increased variety of data, and increased velocity for processing data). One of the most complete considerations of data science and related competences can be found in the EDISON Data Science Framework (EDSF). In EDSF, a data scientist refers to a user who has enough knowledge and expertise in the areas of business needs, domain knowledge, analytical skills, programming, and systems engineering to continuously advance the scientific process across all stages of the big data lifecycle to deliver an expected scientific or business benefit to an organization or project [5].

One critical competence in Data Analytics is how to build and evaluate reliably models. In this context, a de-facto standard is CRISP-DM, which stands for Cross-Industry Standard Process for Data Mining [6]. It describes six steps that are essential for data mining and is widely used in research and industry:

1. Business Understanding: Understand the business objectives of the organization, define analysis goals, and setup a project plan.
2. Data Understanding: Collect, characterize, und understand the data and its meaning and analyze the quality of available data.
3. Data Preparation: Select the data to analyze, clean and format the data.
4. Modeling: Choose the right technique for model building, create the model, and analyze the quality of the model.
5. Evaluation: Evaluate and review the results and determine follow up steps.
6. Deployment: Plan how to deploy and maintain the model in practice, document the outcomes and lessons learned.

These steps are performed iteratively, and you may go back and forth between different steps. In the following, we will use these six steps as a basis for describing our activities performed in the two cases.

3 Case 1: Data-Driven Quality Management

Q-Rapids is a research and innovation project of the European Union (Grant 732253). It aims at creating a solution to exploit software product, process, and usage data with data science to improve software quality [7]. The Q-Rapids architecture is depicted in Fig. 1, which provides: scalable data ingestion from heterogeneous data sources, interfaces with data mining frameworks to derive a data-driven quality model, and evidence-based recommendations to decision makers (from awareness on identified/predicted an external quality problem to quality requirements proposals).

Let’s focus on an example on how the Q-Rapids framework helped improving external quality management for a software platform providing several digital services focusing on rural areas (such as online shopping, news portal, car sharing services).

Business Understanding: There was much potential on improving the software by understanding runtime data. The goal was to identify the key aspects from development time of the platform, which influence software quality at runtime and

how to predict the expected number of runtime issues based on that. The business goal was to reduce the cycle on exploiting runtime data and its history.

Data Understanding: For obtaining the goal above, we had external data (e.g., access/error logs, application crashes, directly received bugs from hotlines or e-mails) and internal data (e.g., code quality metrics, quality rule violations, commits) available. Based on the Q-Rapids tooling, we either used Apache connectors (see Fig. 1) to ingest data in real-time. We ended up creating a centralized repository with heterogeneous data, including usage data (e.g., crashes and logs from using mobile applications). Then, we made an analysis on relevant features of our data as not all collected data was of interest [8].

Data Preparation: We integrated the data stored in different source systems and structures, handled data quality deficits (e.g., incomplete, inconsistent, and incorrect data), and transformed data into the format acceptable for specific methods and tools used during the analysis. This has been the most time-consuming step. Software data is created automatically, but not with the purpose of be ready for analysis.

Modeling: Our goal was to find a fitting model for software reliability. We applied two alternative ensemble methods: random forest and extreme gradient boosting. With these methods, we investigated the probability of software bugs related to runtime issues in association with structural properties of software code as well as the amount and type of software changes along its evolution. The analysis provided two outcomes: relevancy of individual metrics as predictors of external quality and a quality model that captures quantitatively dependencies between the most relevant metrics and the external quality.

Evaluation: A preliminary evaluation showed the ability to show an overview of the system crashes and exceptions over time and predict them based on their historical relationships with internal quality metrics.

Deployment: The developed models were integrated into the Q-Rapids tool for guiding software developers regarding the predicted evolution of external quality or to give an indication of the most relevant factors influencing it.

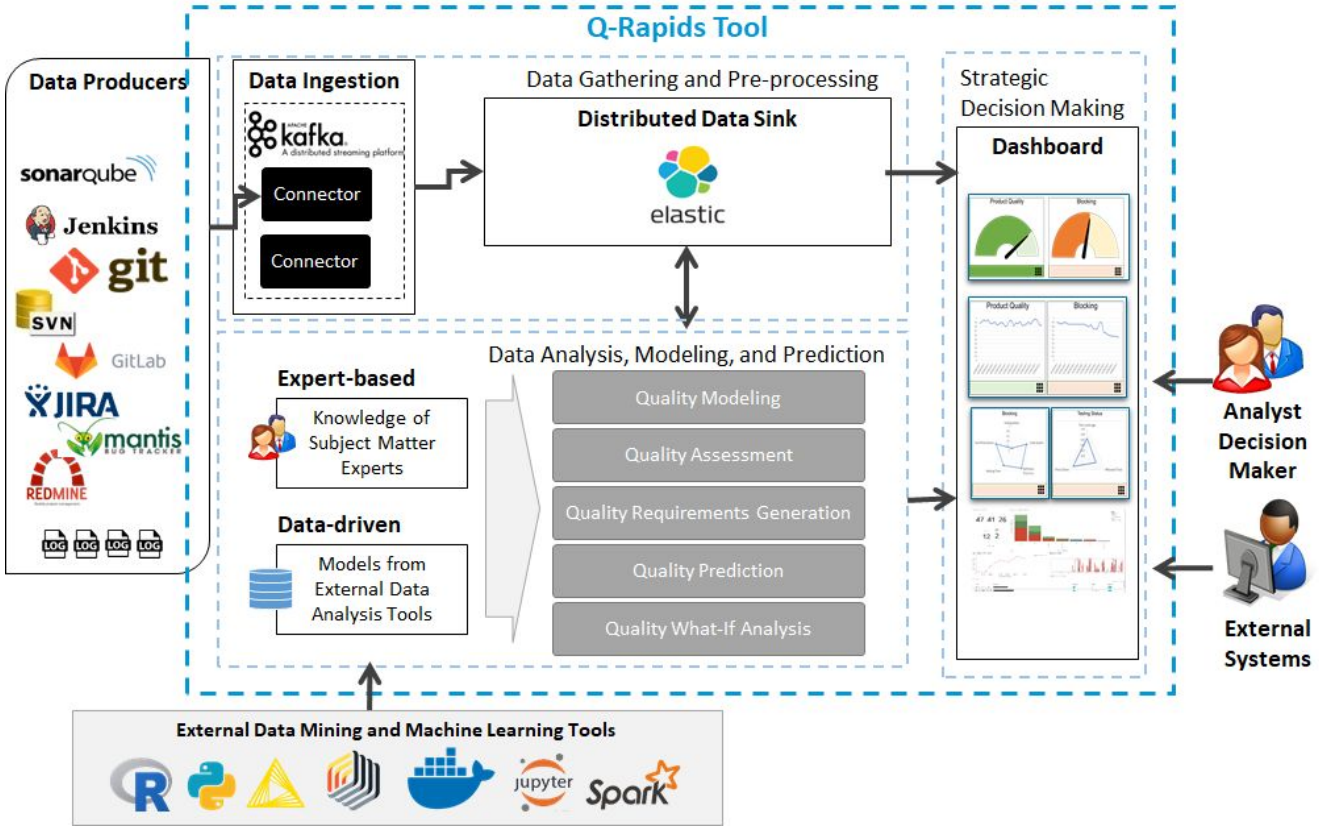


Fig. 1: The Q-Rapids framework architecture

Case 2: Data-Driven Prioritization of Code Findings

Companies increasingly use code analysis to improve quality. Often static analysis is part of Jenkins pipelines in continuous build to achieve strong entry checks before further integration code changes. For safety-critical software it is even mandatory using static analysis. However, static analysis tools are far from easy to use and provide quite complex results. In fact, it is almost impossible using the results without guidance of experienced experts. Designers are overwhelmed with the many tool reports, of which some 90% are irrelevant. Too much false positives and heterogeneous strengths of various tools demand using several tools in parallel. After few trials most companies top using such tools due to their inherent complexity. At Vector Consulting we have over many years build a competence for code quality analysis where we deploy multiple analysis tools and automatically post-process the data for the benefits of our clients.

Business Understanding: Detecting code defects in early stages of the software development not only decreases the costs of the software but also increases the quality by decreasing the technical debt and the propagation of errors to other phases [1]. With 20-30% of all software containing 40-60% of all defects, static code analysis has been shown to reduce post-release failures by over 50% and thus cut development cost by 20-30% [1]. It doesn't replace other verification and validation steps but helps remove certain defect types that otherwise wouldn't be found. However, the warning reports by different static analysis tools do not present the defects in the order of priorities according to the criticality of software modules, but according to predefined hard coded defect priorities.

The basic idea of the case is to use a ranking approach that considers the criticality of software modules to rank the defects that are shown to the developer. This is essential because often the project managers need to decide about where to focus the quality assurance resources to improve the software product.

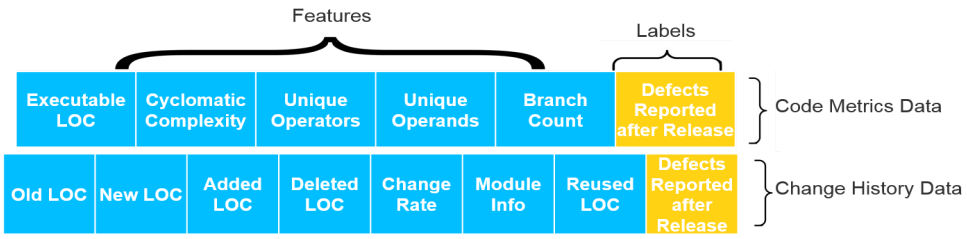
Data Understanding: Typically, there is plenty of data, which can be used to train a classifier, such as: (1) Code complexity metrics such as cyclomatic complexity etc. (2) Change history of modules such as modification rate and addition/deletions in modules. (3) Code reuse due to imports between modules. (4) Process metrics that capture the software development process used to develop the product.

Data Preparation: To give a proof of concept, two of the above aspects, namely code metrics and change history were used to train separate predictive models. Initially the data of real-time projects regarding code complexity and change history was used from the PROMISE repository [9] for training classifiers.

Modeling: The proposed method makes use of criticality prediction and defect density in a module to rank them in the order of criticality from highest to lowest. The criticality prediction is done by training a machine learning algorithm to generate a criticality classifier. The features for training were selected from the data based on the metrics that were generated by the tools used for evaluation. the upper part of Fig. 2 shows the features and labels that were used from the data for training. Using the data sets, two different classifiers were trained using four different machine learning algorithms to measure the performance and select the best algorithm that suits a particular type of data.

Evaluation: The performance metrics that were used included training accuracy, prediction accuracy and area under ROC (Receiver Operating Characteristics) curve. The lower part of Fig. 2 shows the results and the best algorithm for each data set.

Deployment: After training the criticality classifier by choosing the appropriate algorithm for the data set, the next phase was to generate module rankings using these predictions. Since there were two predictive models with binary output predictions (critical or non-critical), this makes four possible prediction combinations, which were assigned a criticality factor from 1 to 4 (most critical to least critical). The classifier which has better prediction accuracy and area under ROC gets assigned a higher criticality factor. The ranking algorithm starts first by extracting the code metrics from tool reports and change histories from version control system for each module. Each of these feature vectors for code metrics and change history are fed to the relevant classifiers to get predictions. The prediction combination for each module is assigned a criticality factor according the criteria described first. The modules are then sorted according to their criticality factors.



Data Type → / Algorithm ↓	Code Metrics			Change History		
	Training Accuracy	Prediction Accuracy	Area Under ROC	Training Accuracy	Prediction Accuracy	Area Under ROC
Naive Bayes	0.80	0.86	0.55	0.83	0.84	0.64
Support Vector Machine	0.97	0.87	0.50	0.90	0.89	0.67
Decision Trees	0.88	0.80	0.57	0.94	0.89	0.67
K-Nearest Neighbours	0.99	0.85	0.55	1.0	0.84	0.50

Fig. 2: Features and labels for training data and performance of criticality classifiers

4 Lessons Learned and Conclusion

This article demonstrates how software engineers benefit from data science approaches. Table 1 illustrates some lessons learned from these two applications sorted by the CRISP-DM steps we ran through. We have seen in two cases that data science approaches clearly provide additional value compared to classical software measurement. However, some aspects require special care: Data Science cannot help in building better software, unless you first understand your business and your data, and prepare the data accordingly.

Data Science in Software Engineering is not about applying ML/AI algorithms on cleaned data sets. You must invest on the initial steps of business and data understanding (i.e., clear objective) and data preparation (i.e., relevant and high-quality data) for delivering business value. Without these steps, there is no guarantee to find valuable insights. Yes, there are tons of available data in software engineering, but they were most often not generated for building analysis and prediction models on top of them.

Using data science for software engineering demands competences that go beyond software engineering skills. Training software developers won't help, as it demands fully new skills way beyond software development, such as statistics, algorithm design etc. Instead we recommend that software decisions makers work with data scientists and consultants with respective skills. This includes:

- Identification of use cases and the benefits of data science (and big data, AI) in the context of software and system engineering (e.g. for quality or risk management).
- Acquisition and storage of system development and runtime metrics, data quality analysis, and data curation (such as software repositories or log files).
- Analysis of measurement data, modeling, application of AI-based techniques (e.g., ML, neural networks or deep learning).
- Tools for collecting, storing, analyzing, and visualizing Big Data and building AI models.
- Data ethics and culture, data exchange, data security and model evaluation.

Nobel prize laureate Herbert Simon once remarked that "a wealth of information creates a poverty of attention." He coined the initial approach to data science as a method to focus on what really matters and thus focus our attention on the stories and essence behind the data. Let us follow his advice also in software engineering and not get drained in ever more complex software, but abstract and analyze for the goal of better quality – rather than more quantity.

Table 1: Summary of lessons learned

Data Science Step	Lessons Learned
Business Understanding	● Defining the motivation for collecting data and building models is essential for not getting lost in the later steps. One should think that after more than three decades after introducing GQM and related approaches for goal-oriented measurement this should be common sense. However, our experience says that especially with the Big Data hype, there is a risk of collecting all the data that one can get and hope for miracles to discover later. Unfortunately, these miracles seldomly happen. ● It is essential to keep in mind the associated potential of Data Science and your capabilities. Data Science comes in different flavors and specializations. It is important to get a clear picture in which area to build up or buy in competences for realizing the intended benefits.
Data Understanding	● Data availability and their quality is one of the biggest issues. Even though a lot of data in software development can be collected automatically by some tool (such as static code analysis tools), a lot of data still needs to be collected manually (such as defect tracking data) and is likely to be incomplete and not fitting to each other (such as data from effort or defect tracking systems and versioning systems). ● Collecting your own high-quality data is effort-consuming, but the best chance in getting the most benefits from making use of the data later. However, it is essential to invest into collecting the data you need according to your goals, which is not necessarily equal to the data you can easily get. ● Data analysis starts already in this phase, because understanding large amounts of data requires an analytical approach. ● It is essential to understand the meaning of the data and not “just” their format. This requires the combination of Data Science competences with domain-related competencies (e.g., from a developer, tester, or project manager).
Data Preparation	● Cleaning and correcting data that is already messed up is very expensive. AI/ML could help improving data cleansing in future, i.e. learning data deficits and applying this knowledge for curing the data. ● Correcting data creation/collection processes means starting data collection from scratch. This approach makes it easy to focus on data quality in the first place, but it takes a long time to gather enough data for model building. ● The application of hybrid approaches is needed for starting small and iteratively expanding. This also required a strong interaction between data preparation, modeling, and evaluation.
Modeling	● Thanks to the Big Data and AI hype several tools for model building and data analytics are available, including a whole stack of free and open-source tools. However, a fool with a tool is still a fool. So, organizations must build up Data Science-related competences for knowing which analysis technique and modeling approach to use on what type of data and for building meaningful models. ● Horizontal scaling supports the “start small and expand” approach giving also SMEs an opportunity to take advantage from data science for software.
Evaluation	● Evaluation should make use of well-known test-learn-improve principle (known from lean startup and QIP). ● All steps up to evaluation are an iterative process of model development. When setting up the goals related to build a model, it is essential to be clear and honest about success criteria (e.g., a certain accuracy or power of the model). Like agile development, it oftentimes makes sense to strive for short iterations and increments when building a model

	instead of spending endless time in data understanding and preparation and then discover that the analysis technique or modeling approach chosen does not work or deliver enough results.
Deployment	● Like software development, “start-small”, “incremental development”, “short cycles” and “minimal viable products” are best practices for deploying your data science solution to practice. ● People making use of the solutions in practices are most likely no Data Scientists and need more support in facilitating the solutions. ● You may have a model that is perfectly working on test data but fails in practices. This can have different reasons, such as overfitting or that the application context of the model is changing over time. When deploying such models, you should take special care about maintenance of the model. For instance, this can be done by regularly evaluating the quality of model decisions and re-learn the model if its performance falls below a certain threshold.

References

[1] Ebert, C., and Dumke, R.: Software Measurement. ISBN 9783540716488, Springer, Heidelberg, New York, 2007.

[2] Gausemeier, J.; Guggemos, M.; Kreimeyer, A.: “Auswahl, Beschreibung, Bewertung und Messung der Schlüsselkompetenzen für das Technologiefeld Data Science”, Acatech-Bericht, 2017.

[3] Conway, D: “The Data Science Venn Diagram”, available from: http://drewconway.com/zia/2013/3/26/the-data-science-venn-diagram (last checked July 31, 2019), 2013.

[4] Heidrich J., A.Trendowicz and C.Ebert: Exploiting Big Data's Benefits. IEEE Software, ISSN: 0740-7459, vol. 33, no. 4, pp. 111-116, Jul. 2016

[5] EDISON Community: “EDISON Data Science Framework (EDSF)”, https://github.com/EDISONcommunity/EDSF (last checked July 31, 2019, 2019.

[6] Chapman, P.; Clinton, J.; Kerber, R. et al.: “CRISP-DM 1.0: Step-by-step data mining guide”, White-Paper, PSSS, 2000.

[7] Martínez-Fernández S., et al., "Continuously Assessing and Improving Software Quality With Software Analytics Tools: A Case Study," in IEEE Access, vol. 7, pp. 68219-68239, 2019.

[8] Aghabayli, A.; Pfahl, D.; Martínez-Fernández, S.; and Trendowicz, A.: “Integrating Runtime Data with Development Data to Monitor External Quality: Challenges from Practice”, in SQUADE19 @ FSE/ESEC 2019: 2nd International Workshop on Software Qualities And Their Dependencies, 2019.

[9] University of Ottawa: “PROMISE Software Engineering Repository”, available from: http://promise.site.uottawa.ca/SERepository/, last checked July 31, 2019.

Biographies

Christof Ebert is managing director at Vector Consulting Services. He’s a senior member of IEEE and is the editor of the Software Technology department of IEEE Software. Contact him at christof.ebert@vector.com.

Jens Heidrich is division manager for process management at Fraunhofer IESE. He’s a member of the subcommittee for the IEEE CS / SEI W.S. Humphrey SPA Award. Contact him at jens.heidrich@iese.fraunhofer.de.

Silverio Martínez-Fernández is operative project manager of the Q-Rapids project at Fraunhofer IESE. Contact him at silverio.martinez@iese.fraunhofer.de.

Adam Trendowicz is a senior scientist for data engineering at Fraunhofer IESE. Contact him at adam.trendowicz@iese.fraunhofer.de.