From Software Development to „real“ Engineering with Software Analytics

Optimizing software production requires data-driven end-to-end process analysis

Johannes Bohnet, Seerene GmbH

This paper introduces the concept of an optimal setup of for a software development organization that produces software for embedded systems: the “Ideal Software Factory”. The paper furthermore describes an analytics method to quantify and precisely observe the software development process and helps to transform the organization into an ideal software factory. The performance of the software factory is thereby assessed by analytics-driven KPIs that capture various performance dimensions such as efficiency, quality, technical debt, time, and more. For the specific aspect of efficiency, a deep dive is given that demonstrates how analytics can be applied in practice.

Introduction: The age of software factories

Software development, as a discipline, still lags years behind traditional engineering disciplines; particularly when it comes to software production involving large teams. Such “software factories” are found in every corporate across all industry domains including for example financial services, logistics, telecommunication, manufacturing, retail, or automotive. A software factory may be part of the IT department and produces tailor-made software systems to support the core business processes of the organization. Or the software production is the core business process of the organization itself. This is the case if the produced software is sold to customers as software products or as embedded systems. The concepts that we describe in this paper are applicable to all kinds of software factories, independently of the type of software produced, the technology stack behind the software, or the process methodology (waterfall, agile, V-model, …). However, in terms of language and examples we will focus on the embedded systems domain. Companies in this sector originally come from producing hardware and are now more and more transforming into software companies. It is astonishingly obvious for them to see how well they run their “real”, hardware factories and how big their lack of transparency is with respect to their software factories. And with the missing transparency about the inner workings of the factory comes the inability to lead it to higher execution excellence and maturity.

In this paper, we will elaborate on key ingredients to drastically increase the efficiency of a software factory as well as on analytics-based methods to continuously improve the factory towards highest excellence in software production.

Running a software factory means balancing multiple dimensions

Someone being responsible for a software factory –let us call her/him the factory owner– needs to observe how the factory performs to be able to optimize it and increase maturity. Measuring the performance of a software factory means gathering quantifiable KPIs for multiple dimensions such as:

- Scope (output, delivered value)
- Budget (money spent)
- Time (speed of delivery)
- Quality (no defects)
- Efficiency (lean process without waste and time loss)
- Technical Debt (maintainable, future-proof code)
- Ability to flexibly scale teams (no knowledge lock-ins)

The analytics method described in this paper, enables factory owners to quantify the dimensions by means of KPIs which can be understood both by the managers “on top” of the factory as well as by the experts “within” the factory. The core idea of the analytics approach

is to leverage the fact that the development infrastructure tools in the factory leave technical data traces (see **Figure 1**).

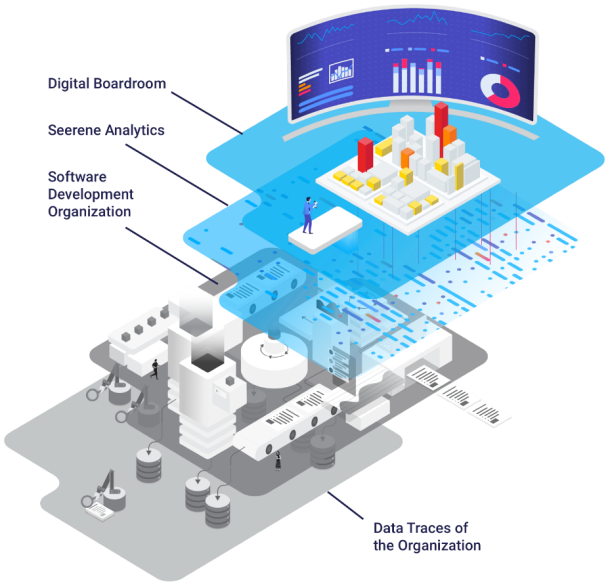

Figure 1: A software factory leaves data traces that can be processed by analytics to reveal the inner processes within the factory and translate it into management-understandable KPIs and insights.

This technical raw data is fused and algorithmically processed into KPIs and –because the insights are derived from low level data– it is additionally possible to drill down and reveal the root cause of a problem.

In this paper, we cannot elaborate on the full set of analytics-driven KPIs that can and should be used to measure the various performance dimensions. Instead, we focus on one dimension, namely efficiency, and describe in detail how analytics can help to improve. We pick efficiency because it is on the one hand side the most important one when it comes to optimize the factory; it directly correlates with saved money. On the other hand, it is the most difficult one to measure because it requires to take data probes along the entire software production process, from requirements, over planning, coding, testing to release handovers (see **Figure 2**). Luckily, end-to-end software process mining is capable of reconstructing the activity along the entire process.

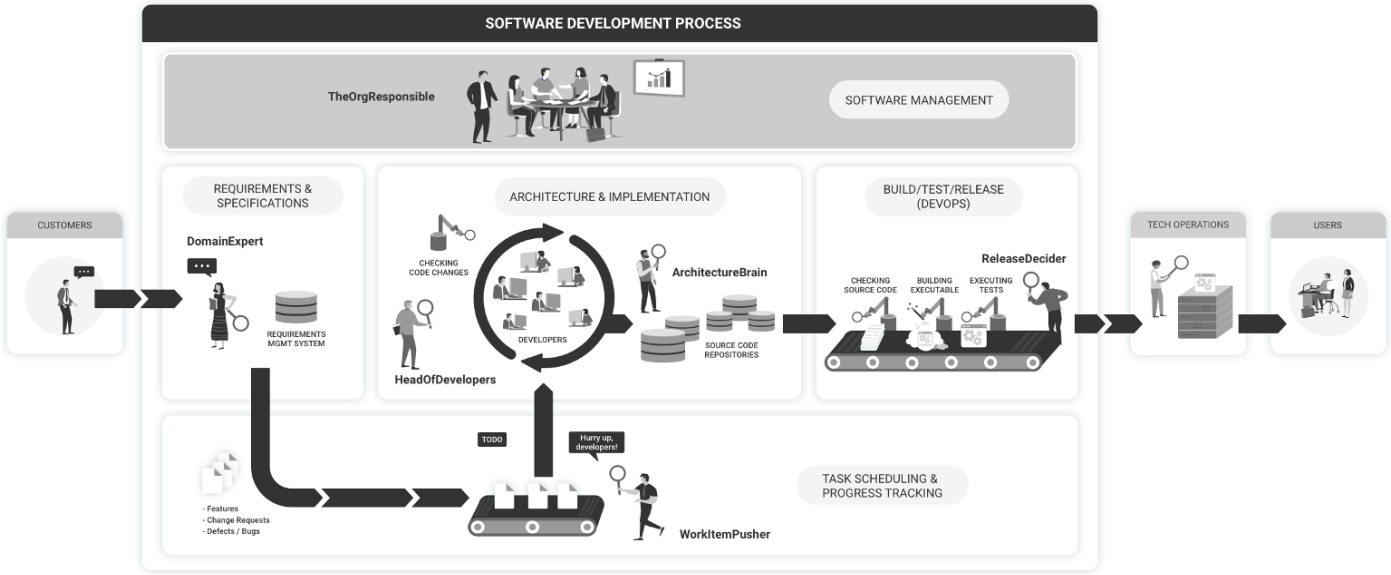

Figure 2: Illustration of a distilled version of tools and stakeholders participating in the software development process

Efficiency in software production

Efficiency refers to minimizing loss of time and money during software development. The major cost driver in software development is the money with which developer time is bought – either by paying in-house developers or by renting them from 3rd-party providers. **Figure 3** illustrates that it would be nice if 100% of the developer time could be used for creating innovation. However, in reality there is "waste" in the process, loss factors that steal developer time so that only a fraction of the 100% can be used for creating business value.

Examples of such loss factors are:

- Coding effort for defect fixing
- Coding effort in very complex code areas (paying "interest" on existing "technical debt")
- Coding effort for removing complexity ("technical debt")
- High onboarding overhead when a developer with a knowledge monopoly leaves the team

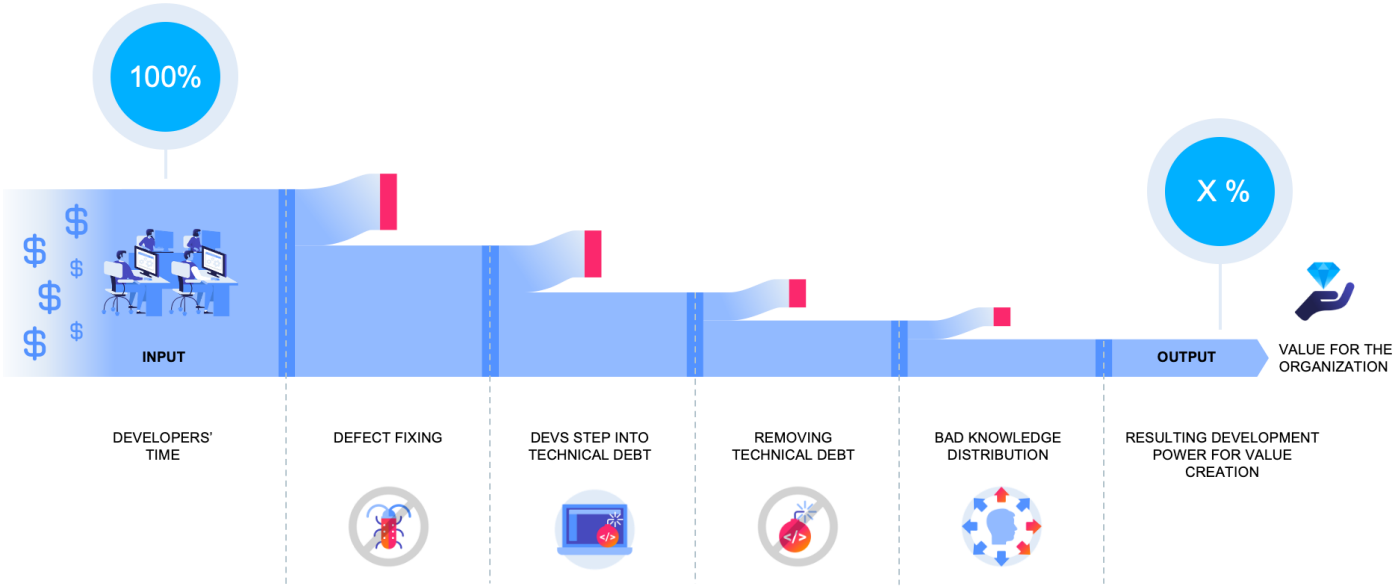

Figure 3: Example of "process waste" that causes low efficiency and reduces developer time for value creation.

All these effects can be measured and quantified with software analytics if raw data from work item tracking (e.g., Jira, IMS, ...), code versioning (e.g., Git, MKS, Mercurial, ...) is brought together and analyzed. With such an analytics technique one can observe the effect of "developer brain meets complex code". **Figure 4** illustrates how coding activities happen within the code and it introduces the information visualization technique of so-called "software maps", whereby code is metaphorically depicted as a city.

In the field of producing embedded software, there is an addition waste aspect that can create a lot of unnecessary work for developers, which is: reinventing the wheel with every new project. This aspect is somehow unique to embedded software because –in contrast to a bank's core backend IT system for example– embedded software is tailored to specific hardware constellations and the production process can be considered as a real project with beginning and end. Producing embedded software means performing many customer- and hardware-specific projects based on the same underlying code base. Key ingredients for efficient software production are therefore reusable code components.

Reusable code components are a must-have for efficient software factories

Software factories for embedded software conceptionally operate on two layers (see **Figure 5**). One layer represents what a Chief Financial Officer would call "value creation" and the other layer represents "value capturing". In the value creation layer, strategic investments are made to build up and add value to "assets", i.e., reusable code components. In the value capturing layer, customer-specific software is derived from the code components and the outcome is sold. A software factory that can run projects without coding effort, just by assembling ready-to-use code components would be a highly efficient revenue generating machine.

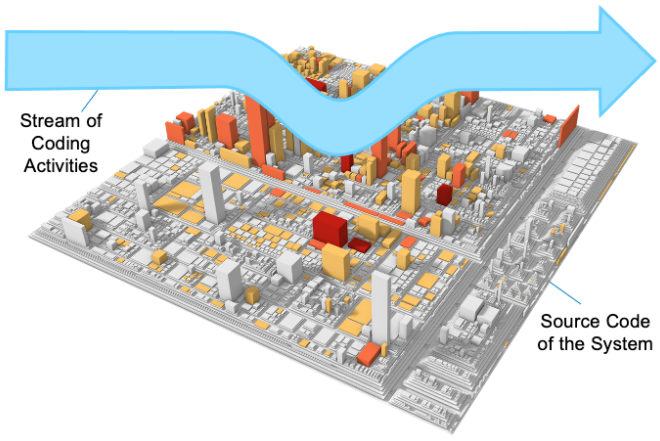

Figure 4: Coding activities within code, metaphorically depicted as a city. Buildings represent code files and are organized as city districts according to the modular structure of the system architecture.

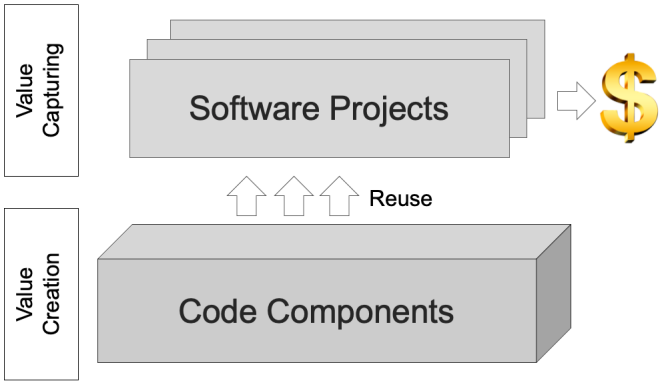

Figure 5: The financial concept of "value creation" and "value capturing" applied to the production of embedded software.

Such an “Ideal Software Factory” is depicted in **Figure 6.** The production process is split –like in a “real”, hardware-component producing factory– into two steps:

1. **Value Creation:** Project-independent ready-to-use components are built in a pre-assembly step (e.g., as static or dynamic link libraries) and stored in a package repository (“artifactory”).
2. **Value Capturing:** In a final assembly step, code components that are relevant for the project are taken from the artifactory, configured, parametrized, and glued together (e.g., by linking the libraries) to create the project deliverable.

Interestingly, the role model for this setup can be examined in every Linux-based PC/server. The basic reusable component is the operating system and then, depending on the individual user requirements, additional ready-to-use components are added. Famous artifactories for managing reusable components are for example APT or RPM. To install a driver for a specific graphics card, for example, one would simply execute the command: ‘apt install nvidia-driver-440’.

However, embedded systems often have hardware constraints. In this case, any binary code needs to be highly optimized. A software factory setup as depicted in **Figure 6** would not be suitable because the compilation process in the pre-assembly step would not know about the project-specific requirements. In such situation, the “Ideal Software Factory” would still consider reusable code components as assets. The only difference would be that

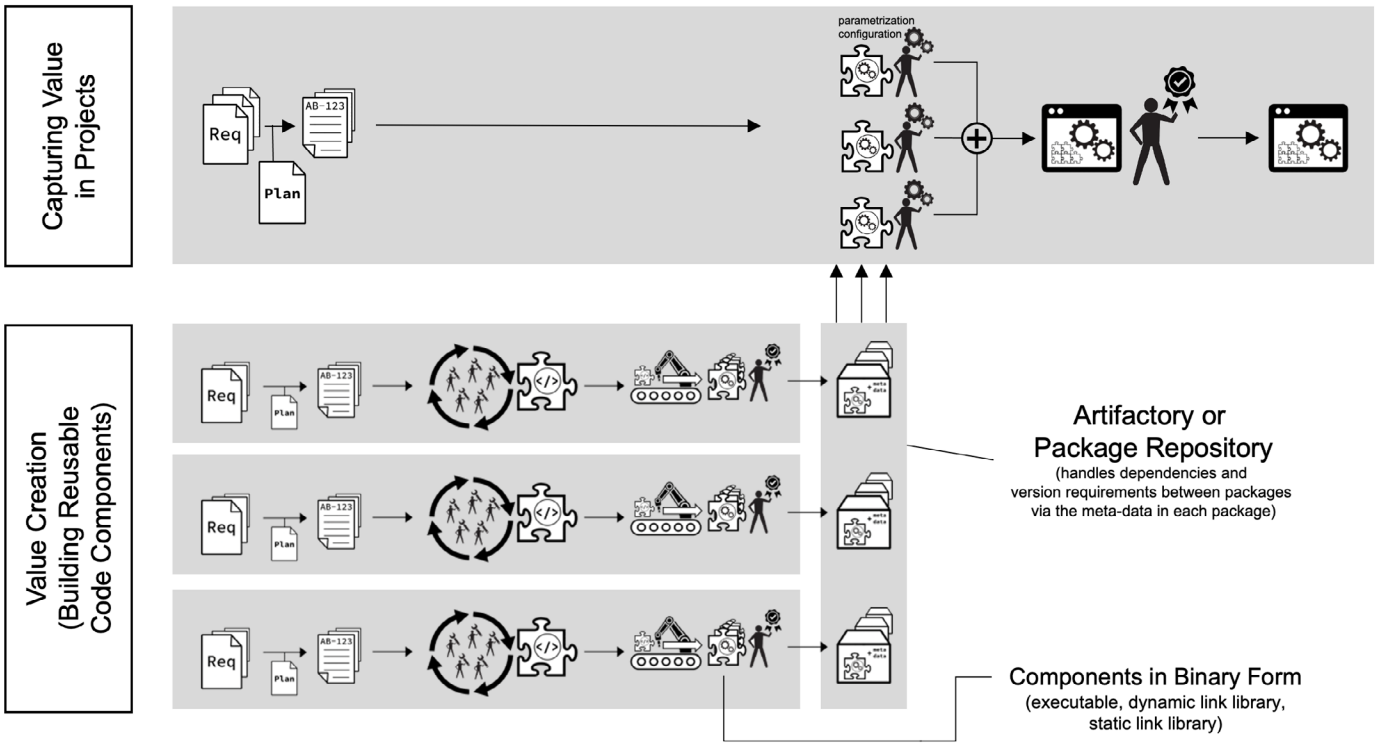

Figure 6: The Ideal Software Factory.

the components would be brought together as source code in final-assembly, not as compiled binary code (see **Figure 7**). However, such a software factory is conceptionally equal to the aforementioned “Ideal Software Factory”.

Analytics as navigation guide towards the Ideal Software Factory

In practice, a software factory is often not yet an “Ideal Software Factory” as elaborated in the last section. Instead of just gluing ready-to-use code components together, a typical project involves a significant amount of project-specific coding effort; because the projects require functionality that is not yet provided by the reusable code components. In software factories with very low maturity, only rudimentary components are provided, and each project has to build up the similar functionality from scratch – over and over again.

Analytics helps to quantify how far a software factory is still away from the ideal setup. For each work item of a project (stored in e.g., Jira, IMS, …) the code changes are revealed (stored in e.g., Git, MKS, Mercurial, …) and the effect on the code units (determined by static code analy-

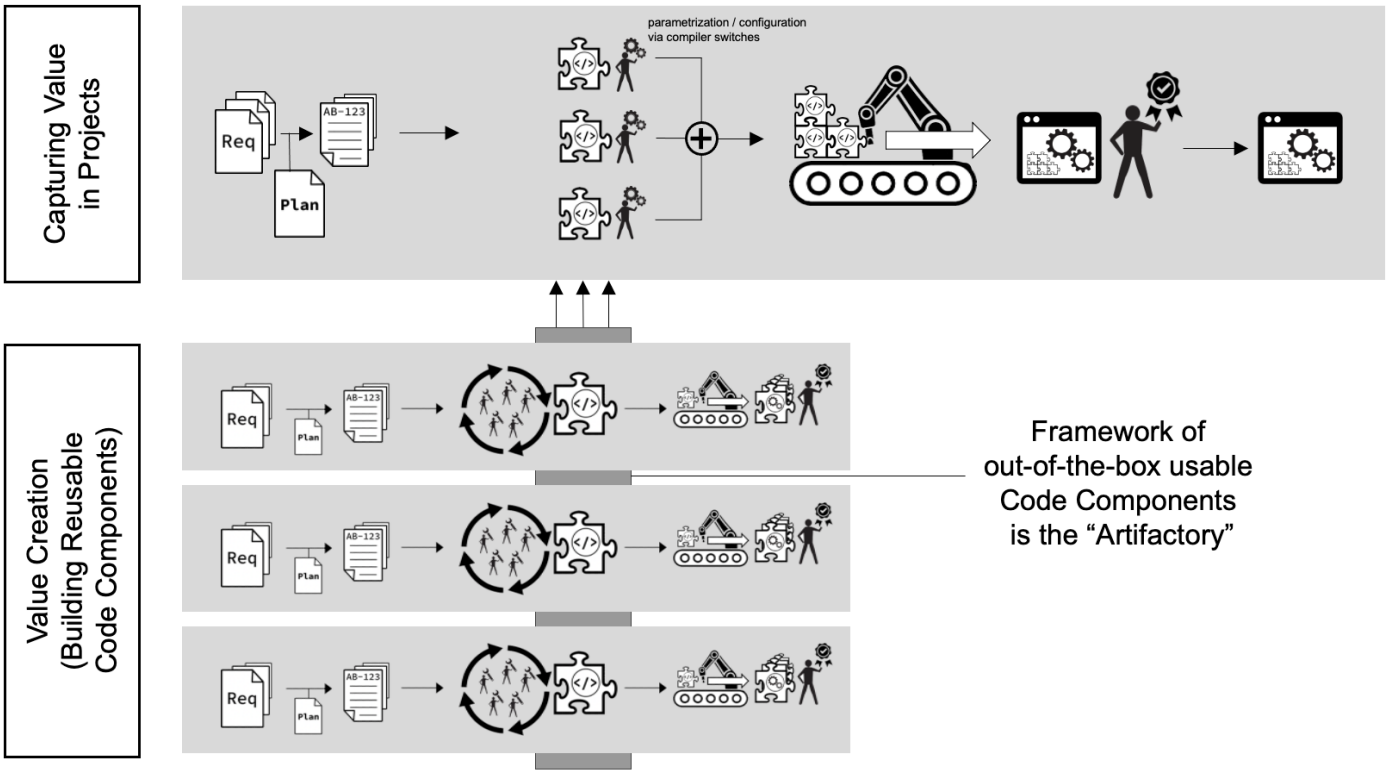

Figure 7: The Ideal Software Factory – if compile-time optimization is necessary.

sis). This way, full traceability is provided for each work item:

- How much coding effort was necessary to implement the work item (reconstructed from the anonymized developer footprints in Git, MKS, …; no time tracking information is needed)?
- Where in the architecture did the coding effort flow into? Which component received how much effort (e.g., in person days)?
- What was the effect on the code? Increased amount of logic, complexity, …?

There are two perspectives how these analytics-based insights help the software factory become better. On the one hand, one can observe the amount of additional project-specific coding effort across all projects. The goal should be to minimize such effort and make value capturing effortless. The only exception would be if there is a strategic investment into a specific component during the course of a project. However, this would mean that the “value creation” part is performed within or in parallel to the “value capturing” part.

On the other hand, one can observe in which code components the additional project-specific coding takes place. One can even drill down to individual code files that consume the effort (see **Figure 8**). This gives clear actionable insights into which components are not yet ready for out-of-the-box use. These components are good candidates for strategic investment (value creation) to free up developer time in upcoming projects (value capturing).

Summary and outlook

In this paper, we discussed challenges for the ones being responsible for a software factory; particularly, the need to continuously balance and optimize the factory with respect to various dimensions. For the dimension “efficiency”, we made a deep dive and pinpointed various aspects that reduce productivity. We then had a dedicated look into software factories for embedded software and introduced the concept of the highly efficient “Ideal Soft-

Figure 8: Drill down from a KPI quantifying efficiency loss into the code architecture to reveal the root cause of the problem.

ware Factory" that distinguishes between investments into reusable code components (value creation) and effort-free money-making by assembling customer-specific software in projects (value capturing). Furthermore, we described the concept of analytics-driven software process mining that enables software factory "owners" to observe and measure the inner workings of their factory. And we showed how analytics can help to reach the ideal software factory setup.

For reasons of brevity, we did not elaborate on the immense power of this analytics method for software engineering in general. As an outlook, it shall be mentioned that analytics-based traceability along the development process helps in a multitude of ways. Examples: It helps to improve quality by revealing defect-risks behind requirements and work items very early; it reveals disadvantageous team setups and knowledge distributions; it allows for comparison of performance KPIs across multiple software factories the hierarchy of departments, business units, and divisions within a corporate – regardless of the technology-, tool- or methodology-specifics used in software production.

Author

Dr. Johannes Bohnet is founder and Co-CEO of Seerene GmbH, an analytics and software process mining company that has its roots in the Hasso Plattner Institute, the German university center of excellence in the field of software engineering. Before starting with Seerene, Johannes completed his graduate studies in physics at the German universities of Heidelberg and Münster. After this, he worked as a research and teaching assistant at the Hasso Plattner Institute, where he built up and headed HPI's research group on software analytics and visualization. He finished his academic career phase with a Ph.D. in computer science at the University of Potsdam. In parallel to his scientific activities, he worked as an IT consultant and trainer a renowned consultancy firm in Frankfurt for many years.

Contact

Internet: www.seerene.com
Email: johannes.bohnet@seerene.com