Improving Software Documentation using Data Visualization

Oskar Hulter

Systems Sciences, bachelor's level
2018

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering
Abstract

This exploratory case study argues for the thesis that data visualization (dataviz) can have a positive impact on users understanding and perception of code documentation. The basis of the thesis is a case study performed in collaboration with SensiNet AB. The introduction describes the case and the problem area. The theoretical framework and methodology focus on theoretical foundations and how the data collection was planned and performed. The paper concludes with a brief discussion of related work and several suggestions for future studies.
# Table of Contents

**Introduction**  
- Research Question  
- Case Introduction  
- Working with Legacy Code  
- Non Disclosure Agreement  
- Visualization Examples  
- Goals of the Documentation  
- Study Limitations  

**Theoretical Framework**  
- Basic Definitions  
- A Brief History of Data Visualization  
- Unified Modelling Language  
- Design Principles  
- Software Product Line  
- Ontology  
- Knowledge Management  

**Methodology**  
- Case Study Criteria  
- Motivation for Method Selection  
- Case Study Type  
- Experimental Guidelines  
- Triangulation  
- Object of Study  
- Proposition  
- Study Steps  
- Data Collection  
- Summary of data collection methods  
- Changes During Data Collection  
- Data Analysis  
- Case Study Validity  

**Result**  
- Questionnaire Results  
- Answer to the Research Question  

---
## Analysis

- Study Problems  
- Alternative Explanation  
- Questionnaire Results  
- Interview Commentary  
- UML Critique  

## Conclusion

- Concluding statement  

## Future Studies

- Summary of Suggested Studies  
- The Need for Ontologies  
- Machine Learning  
- Suggested studies  

## Appendices

- Data Tables  
- Interview Questions  
- Ware’s Design Principles  

## References
Introduction

The growing dependence on technology is a fact that cannot be ignored. Evidence points toward higher productivity per worker and as well as growing productivity but stagnant employment (Rotman, 2013). Technology is one important factor contributing to this development. Therefore the likelihood that the average employee will work with technology is increasing. This means that the learning process of untechnical employees have great economic importance since it will constitute an ever-growing part of the total work time.

Data visualization (dataviz) may be an opportunity to make complex concepts within technical documentation easier to understand. This paper argues for the thesis that data visualization can help users more easily understand code documentation since it can allow users to process more information in less time.

Research Question

“Can data visualization make legacy code documentation easier to understand?”

This study is an exploratory case study focused on evaluating whether data visualization can have a positive impact on the learning experience of subjects using legacy code documentation.

Case Introduction

The study was executed in collaboration with the swedish company SensiNet AB. SensiNet AB develops IoT-sensors under the trademark TermoSense. The sensors have a wide range of applications but the focus is on temperature surveillance systems for grocery stores and laboratories. SensiNet needed new documentation for their software systems. Most knowledge about the system is not written down and a formal documentation has never existed even though the source code is over 10 years old. When I was making the first version of the documentation I got the idea for this study. The documentation I wrote is supposed to be accessible for a large audience, even without prior programming experience.
Working with Legacy Code

Legacy code presents a few difficult problems that I briefly want to outline based on my own experiences. Legacy code can be hard to understand even for experienced developers. Software has a tendency to grow sporadically as new needs arise and this can make the purpose of the code harder to understand. Every developer that has started a green-field project has experienced the god-like power of creation. This is in stark contrast to the feeling of working with legacy code. Legacy code makes you feel constrained. Every change can have fatal consequences. To minimize the risk of missteps there are a few techniques that can be used, the most important being testing. Testing allows us greater awareness of our surroundings but it is not a silver bullet. Code that is built without the goal of testability can often prove very difficult to test, bordering on impossible. To enable testing the code can be refactored and before you can refactor the code, it has to be understood. Refactoring without understanding is dangerous. The goal of this report is to assess if data visualization is an efficient learning aid for understanding legacy code. Data visualization techniques will complement the internal documentation of the SensiNet source code.

Non Disclosure Agreement

Specific information cannot be discussed since both the documentation and the code is under NDA. The documentation is divided into two main sections. The first section is a general introduction to programming that goes over the basics needed to understand the source code. The second section is about the code architecture and it is more technical. Below are a few examples not covered by the NDA showing how data visualization is used within the documentation to illustrate basic programming concepts and hardware layout. All visualizations are created by the author of this paper.
Visualization Examples

The following examples illustrates how data visualization was used to change the way information was communicated. The text version of the information is shown first followed by a variation utilizing data visualization.

```c
int square(int x){
    int squareOfX;
    squareOfX = x * x;
    return squareOfX;
}
```

A basic C function example with explanations of terms and color coding.

The function named square takes in a parameter called int x and returns the result of x * x as an integer.

```c
int square(int x)
{
    return x * x;
}
```

This function does the same thing as the previous one.

They illustrate the flexibility of the C programming language.
A Linked list is a data structure consisting of a linear collection of data elements. Their order does not correspond to the physical placement in memory. Instead, every element has a pointer to the next element of the collection. The final pointer points to NULL which signifies the end of the linked list.

\[
\begin{align*}
A & \rightarrow B & \rightarrow C & \rightarrow \text{NULL} \\
\text{A, B, C} & = \text{Data} \\
\text{P} & = \text{Points to next element} \\
\text{NULL} & = \text{Nothing}
\end{align*}
\]

*Diagram illustrating the linked list data structure.*

The Central Processing Unit communicates with RAM and EEPROM. The Input or Output ports are sending data to Central Processing Unit. The CPU saves the input into the Long term memory of the RAM and EEPROM. This is how a computer operates on any scale.

*Simplified diagram of how a microprocessor works.*
A sensor is able to take input from the outside world and enable the processor to process the data. The processor can then broadcast the output to several sources.

![Diagram](image.png)

*Illustrates how a sensor can process input and generate output.*

The main takeaway from these examples is that visualizations are a powerful way of communicating relationships between entities. The relationship can be understood on a superficial level without the need to understand all of the terminology and jargon.

**Goals of the Documentation**

The main reasons why the new documentation is needed are listed below.

- **Knowledge Management**
  - Having a reliable source of knowledge about the code that is accessible independently of any one person is essential for knowledge management.

- **Prepare future development**
  - The documentation will play a crucial role in preparing plans for future developmental efforts.
  - Examples of future steps include adding features and prototyping architecture changes in order to reduce technical debt.

- **Learning**
  - The documentation will enable employees without prior programming experience to gain a better understanding of how the products work.
  - The documentation will also make it easier for the employees to learn more about the relevant topics through links to reliable sources.
The design of the documentation follows from the program design. The information is divided into several units that are explored in turn according to importance. The documentation is written in markdown files on github and is only available online.

Study Limitations

The main limitation of the study is the low number of subjects that will be available. The code that is documented is under NDA and therefore only employees are eligible subjects.
Theoretical Framework

Basic Definitions

**Software**: “Intellectual creation comprising the programs, procedures, rules, and any associated documentation pertaining to the operation of a data processing system” (ISO, 1991).

**Software Documentation**: Formal writing in print or electronic format that supports efficient and effective use of software in the intended environment (Barker, 2003).

**Data Visualization**: A collection of techniques that aim to present data to a human in a way that accurately communicates information and requires minimal effort for comprehension (Lee, 1998).

**Ontology**: Explicit formal specification of the terms in a domain and relations among them (Gruber, 1993).

A Brief History of Data Visualization

The purpose of this summary is to give the unfamiliar reader some context about the field of data visualization. The first use of data visualization was probably limited to cave paintings. The earliest cave paintings are at least 35,000 years old (Zorich, 2012). This means that dataviz has been practiced long before it was formally defined.

Calendars, tree-diagrams (tracing lineage), musical notation and structured diagrams followed (Berinato, 2016). Tables were popularized in the late 17th century and they were the primary way of arranging data points in a way that made them easier to read (Berinato, 2016). The first known use of modern data visualization in the form of charts and graphs is attributed to William Playfair.
He published *The Commercial and Political Atlas* in 1786 which featured bar and line charts. From that point the field of data visualization has continued to evolve. It is no surprise that charting became more popular during the industrial revolution. Since visualization is an abstraction it became an important way of handling the increased complexity caused by the industrial revolution (Berinato, 2016).

Edward Tufte wrote the seminal work that marks the beginning of the modern era of dataviz, *The Visual Display of Quantitative Information* in 1982. Since then the field has exploded in popularity. This happened in large part thanks to the popularization of computers.

Today the world data visualization is all around us. The internet and later smartphones has made data visualizations a big part of our everyday experience. The rest of dataviz history is beyond the scope of this paper.

Goals of data visualization include clear and efficient communication via statistical graphs, plots and information graphics. The term visualization has until recently meant constructing a visual image in the mind (Little et al., 1972). The meaning has now been shifted toward a graphical representation of data or concepts. The terms meaning has changed from an internal construct of the mind to an external artifact that supports decision making (Ware, 2012).

Unified Modelling Language

UML is a form of visual documentation that is widely acknowledged as the leading modelling language for source code. It is a detailed modelling language that can be used to model various types of system architectures. One of the creators of UML, Ivar Jacobson (2009), admits that UML is not the silver bullet it was sold as 10 years ago. He goes on to add that it is complex and clumsy though he suggests using 20% of UML for 80% of software projects. The pros and cons of the Unified Modelling Language (UML) is discussed in the analysis section.
Design Principles

The visualizations are primarily based on the design principles extracted from Colin Ware’s book *Information Visualization: Perception for Design* (2012). Ware’s principles were formulated as a result of his extensive review of current research about human perception. The principles were used as guidelines in the design of the visualizations. The appendix features a selection of principles from the book that were applicable to the creation of the visualizations.

Software Product Line

The codebase adheres to the software product line architecture. “A software product line is a set of software-intensive systems that share a common, managed feature set satisfying a particular market segments specific needs or mission and that are developed from a common set of core assets in a prescribed way.” (Northrop, 2002)

Ontology

In the context of computer science ontology means explicit formal specification of the terms in a domain and relations among them (Gruber, 1993). Ontologies can be simple keyword-hierarchies or complex concept-hierarchies with properties, value-restrictions, and axiomatised relations between the concepts (van Harmelen et al., 2001).

The analysis section recommends formulating a ontology describing the most common software patterns as a future study. The most effective visualizations for the types of relationships that the ontology describes can then be studied. This would enable us to choose the right types of visualizations but also lay the groundwork for utilizing machine learning in this problem space.
The transformation of raw data to visual presentation goes through a two step process (van Harmelen et. al., 2001).

1. Raw Data gets converted to an intermediary semantic structure. This means that the raw data is arranged into a meaningful structure. The purpose of this step is to determine what to visualize.
2. The semantic structure is used as a foundation for creating a visualization representing the meaningful structure.

The paper *Ontology-based Information Visualisation* by van Harmelen et al. (2001) shows how visual representations of information can be based on ontological classifications of that information. This is the basis of my suggested future studies.

Knowledge Management

Knowledge management is the main reason why Sensinet AB wanted to invest in their documentation and try to improve it with data visualization. The goal of Knowledge management (KM) in software organizations is to preserve the most important asset of the company: intellectual capital (Rus & Lindvall, 2002). Software organizations have to guard against the fact that their experienced employees will leave the company while inexperienced employees have to be hired. KM seeks to capture the knowledge of the experienced employees to minimize the loss of intellectual capital when they inevitably leave.

The popularity of KM has increased since the 1990s: 80 percent of the largest global corporations now have KM projects (Rus & Lindvall, 2002).

There are two types of knowledge according to KM theory: tacit knowledge and explicit knowledge. Software documentation provides a way of transforming tacit knowledge into explicit knowledge that can easily be shared between individuals and organizations (Rus & Lindvall, 2002).
Methodology

This section begins with a review of how the case study is constructed and which theoretical frameworks support it. The end of this section focuses on the practical steps of data collection and analysis.

Case Study Criteria

Perry et al. (2005) states that case studies are expected to follow the criteria listed below (the part of the report that fulfills the criteria is in parenthesis).

- Research questions should be set from the beginning of the study. (Introduction)
- Data should be collected in a planned and consistent manner. (Methodology)
- Inferences should be made from the data to answer the research question. (Result)
- The study should explore a phenomenon, or produce an explanation, description, or causal analysis of it. (Conclusions)
- Threats to validity should be addressed in a systematic way. (Conclusions)

Motivation for Method Selection

Case studies are well suited for studying phenomena that do not have a strict boundary between the object of study and its environment (Runeson & Höst, 2008). It is impossible to untangle this specific legacy code documentation from the context of the company, their products and the source code.

Case studies are most common in areas like sociology, psychology, business, political science, community planning and social work (Yin, 2003). This study focus on the combination of data visualization and software development. The combination could be considered dataviz and technical writing as well.
In this case the documentation is an integral part of the software development process which makes it a more fitting classification. A case study is also a good fit for this study because software development involves a combination of the same considerations as the fields summarized by Yin (2003).

Case Study Type

Robson (2002) defines four types of case studies based on their purpose: Exploratory, explanatory, descriptive & improving. This study is an exploratory case study because its objective is to evaluate if further studies about the subject are worthwhile.

Experimental Guidelines

The following guidelines were followed when writing this report. The list is a summary of the guidelines defined by Kitchenham et al. (2002). The list has been reduced to contain the parts that are relevant for this study. The part of the report that documents the guideline content in parenthesis.

**Experimental Context** (Introduction)

- Background information about the circumstances in which the empirical study takes place.
- Discussion of the hypothesis.
- Related research. (analysis)

**Experimental Design** (Method)

- The population being studied.
- Rationale & technique for sampling.
- Methods for reducing bias and determining sample size.

**Conduct of the Experiment and Data Collection** (Method)
Analysis (Analysis)

- Two main approaches of analyzing results: Classical analysis & Bayesian analysis.

Presentation of results (Result)

Interpretation of results (Analysis & Conclusion)

(Kitchenham et al. 2002)

Triangulation

Using two or more methods to check the result of the same study is called triangulation. This is important to increase the reliability of research. Stake (1995) defines four types of triangulation.

- **Data triangulation**: using multiple data sources.
- **Methodological triangulation**: combining data collection methods.
- **Theory triangulation**: using different theories.
- **Observer triangulation**: using more study observers.

Methodological triangulation is used in this study. It features qualitative data from the interviews and quantitative data from the questionnaire.

Object of Study

The user’s perception of code documentation is the primary object of study. The secondary object of study is data visualization and how it might affect the primary object of study.

Proposition

The main proposition is whether documentation can become easier to understand with the help of dataviz. This proposition is tested with interviews and a few quantitative metrics to achieve triangulation.
Study Steps

1. Analyze a legacy code base.
2. Document the code.
3. Create diagrams following conventions established by research in the field of data visualization.
4. Give the documentation without diagrams to users that are new to software development but that have a real desire to understand the code.
5. Give the documentation with diagrams to the same users.
6. Evaluate their experience in interviews.
7. Compare and analyze the findings.

Iteration over the case study steps are common (Andersson & Runeson, 2007).

The study was conducted at the headquarters of SensiNet. The study featured seven participants from the staff. Their job duties at SensiNet ranged from assembling the actual parts of the sensors and doing repair or other types of maintenance. Prior experience of programming varied among the participants, from very experienced to not having read a single piece of code. This contributed to a wide range of prior programming experience with a bias toward less experience. The subjects answered the questionnaires in a separate room with their own computer. The interview took place right after they had completed the second questionnaire.

Data Collection

Most case studies are based on qualitative data but using both qualitative and quantitative data often results in a greater understanding of the studied phenomenon (Seaman, 1999). This is why the study features interviews and questionnaires.
Each subject are given two sets of questions each. They answer the questions using the documentation. The first set of questions are answered using the documentation without dataviz while the second set of questions are answered with dataviz. Each subject completes as many questions as possible in a 20 minute timespan for each set. The amount of time the subjects need to complete the tasks as well as amount of questions completed are recorded to generate additional quantitative data. The type of documentation changes order on every subject, starting with dataviz first and then dataviz second on next subject etc. This makes sure that each question set gets tested with both versions.

Summary of data collection methods

This is a single-case study with a holistic design. The objective is exploratory. The study focuses on user sentiment towards dataviz in documentation and it features quantitative measurements to contrast or confirm the result of the interviews.

Data Collection Steps

- Question set 1 & 2 (four questions each, 20 minutes each)
- Question set completion metrics (time to completion, no. right answers)
- Semi-structured interviews

The qualitative data consists of answers to an interview after the study. The questions focus on how they perceived the documentation and what they felt.

The interviews were semi-structured as it is a good fit with the exploratory study objective. The maximum number of subjects is ten because the subjects are limited to company employees. The data is de-identified.
Changes During Data Collection

A lot of learning took place in the first few data-gathering interviews which resulted in three modifications of the original data collection plan.

<table>
<thead>
<tr>
<th>Timing</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>after 1st Interview</td>
<td>Added 20 minutes of documentation browsing before questions were posed.</td>
</tr>
<tr>
<td>after 1st Interview</td>
<td>Removed one question to make the process less stressful for the subjects.</td>
</tr>
<tr>
<td>after 2nd Interview</td>
<td>Simplified the questions to “was this information easy to find in the docs?” yes/no</td>
</tr>
</tbody>
</table>

Reasoning

- Most of the subjects had no prior experience of navigating documentation before. They needed some time to understand the layout and get comfortable with navigating the docs on a computer. A 20 minute browsing period was therefore added before the subjects answered the questions.

- One question was removed because high stress levels was reported by the subject.

- The questions were originally very specific, requiring only an unreflective answer. This made it difficult for participants to reflect on the content of the documentation. The atmosphere of the study felt like it was a test and it did not matter that the participants were told that they were not being tested based on their performance. It might be a very familiar setting that subjects associated with taking a test in school. When the questions were changed the subjects provided much more valuable information. They started to read more of the documentation instead of just scanning mindlessly for the answer. The original questions also had the consequence of only really testing the experienced programmers on general programming knowledge that they could guess without the help of
the documentation. That was a fault of the study design that I were not able to foresee but it was quickly adjusted. This was achieved by adding a comment to the top of the questionnaire that asked the subjects to only answer yes/no if the answer was easy to find in their opinion. An interesting observation was that when I asked participants to not focus too hard on the specific programming related content, they were more inclined to investigate it themselves.

Data Analysis

The interviews will be judged according to the opinions of the subjects. Their opinions will be summarized and categorized as one of the following:

- Dataviz had a positive impact
- Dataviz had a negative impact
- Dataviz had no impact
Case Study Validity

Yin (2003) describes four types of validity that are important for case studies.

**Construct Validity**
The study focuses on how the users think about two versions of the documentation. To understand the users thoughts and feelings it is suitable to conduct an interview. Since the study is exploratory this is a good starting point. Future confirmatory studies can focus on other types of data like measuring brain activity or testing learning retention. These methods of data gathering are more expensive and still profit from earlier indications by exploratory research. For the purpose of this study understanding is interpreted as the ability to use. If the user reports that dataviz has a positive impact on their ability to use the documentation then dataviz has helped the users understanding of the documentation.

**Internal Validity**
Internal validity is ensured by rotating the order of doc types for each subject. This means that each question will get tested with and without visualizations.

**External Validity**
External validity concerns to what extent it is possible to generalize the findings of the study. The results of this study should be interpreted as an indication which can encourage further studies. This is because the number of possible subjects in the company is small. Since the objective of the study is exploratory this fact does not hinder the goal of the study to the extent that it would not be worthwhile.

**Reliability**
Reliability concerns whether the data and analysis is dependent on a specific researcher. Reliability is achieved by documenting the study steps so that other researchers can conduct the same experiment.
Result

Every employee that was available at the time of the study participated. The participants came from a broad range of experience levels. Their day-to-day work tasks were also varied. The work tasks ranges from trouble-shooting the sensors to installing sensors on site, adding new features in the codebase, sensor maintenance, bookkeeping and website development. Two subjects had prior experience of programming. Every participant signed a contract that ensured de-identification and clarified the terms of the study. The study was conducted in a small room where the participant could bring their own computer to use the documentation on. The rationale behind the study design can be found in the method section.

The majority of the subjects reported that visualizations helped them find the information more easily. Both the quantitative and qualitative data support the thesis of this paper. The following data was compiled from the interviews.

![Documentation Preference](image)

The result is in favour of data visualization with 71% of the subjects saying they prefered the dataviz version in the interview.
An interesting observation was that the only subject to prefer text only was an experienced programmer. The subject reported that it was easier to search using shortcuts when the diagrams were missing. This could imply that the value of dataviz is affected by the user experience level.

![Average Time per question set](image)

The average time taken to answer the question sets tell a surprisingly similar story. The subjects used 74% more time on average when using the text only version.
Questionnaire Results

Tables featuring all of the data are available in the appendix, p. 35. The importance of the questionnaire result is reduced after the change to the questions after the second subject. The purpose of the questions changed since the questions were simplified to better suit the overall experience level of the participants. This means that the questions did not test the knowledge that the subject was able to extract from the documentation. Instead they were used as a foundation for the discussion and questions during the interview. The questionnaire answers were overwhelmingly positive with just three negative answers in total among all subjects. There were no significant difference between the documentation types.

Answer to the Research Question

“Can data visualization make legacy code documentation easier to understand?”

This study indicates that users found the visualizations helpful for navigating and reading the documentation. The study did not adequately measure understanding because the data collection changes made the measurement of subjects knowledge more difficult. The result does show that subjects have a positive perception of how visualizations can help them process the information. The quantitative data does support the proposition. The data collection changes affected the questionnaire answers but the time measurements were strongly in favor of data visualizations. The opinion of the subjects was generally positive towards dataviz in the context of documentation. Most subjects agree that visualizations are important for processing and remembering information in general.
Analysis

The main goals of the analysis is to comment on the result and lay the foundation for future study suggestions. The secondary goal is to put the study into context by analyzing results of studies in related fields. The analysis explores the broader question of how software documentation can evolve from here and how dataviz can contribute to that development.

Study Problems

Before I motivate the steps listed above I want to highlight some problems of this study. The study did not adequately test understanding since a few changes made the measurement of the subjects knowledge harder. The result does show that subjects have a positive perception of how visualizations can help them navigate and process the documentation. The study also included a low number of subjects which questions the statistical significance of the study. Most of these problems can be minimized in replications thanks to the lessons learned from this study.

Alternative Explanation

One alternative explanation for the results that has to be addressed is the repeated exposure to the same documentation. This is the biggest issue of the study. The warm-up period that was introduced by one of the changes always showed the doc version with visualizations. This complicates the interpretation of the results since the subjects might score better with the more familiar version with the documentation. The study can be improved by having a third warm-up version of the documentation that is similar but not exactly the same. That could reduce the possible bias while still providing the subjects with an opportunity to get familiar with the experience.
Questionnaire Results

The fact that the questions were simplified after the second interview changed what they were testing. The original idea was that the subjects real world knowledge had to be tested to ensure that they could understand the documentation. This would have been a good approach if the average programming experience was higher. The subjects that were new to programming felt overwhelmed and therefore a compromise had to be made. The change resulted in a better study but it also changed what the data is able to tell us. The data does not empirically test the understanding as it was meant to. Instead the data indicates whether the subject thought they would be capable of utilizing the information contained in the documentation easily. For most subjects this remained a hypothetical question since many were missing the foundational skills needed to utilize programming information in general.

The fact that programming experience was low among subjects brought an improvement to the study as well. The study came closer to the conditions of how untechnical users looking to increase their technical knowledge feels during usage. If every subject was an expert then that implies some level of mastery of the topic that the documentation covers. Therefore it is a totally different use-case that might warrant the creation of a separate documentation that is optimized for different objectives. An expert is probably more likely to have a well defined question that they want answered with the help of the documentation. The expert wants to find specific details quickly and easily. A documentation that is optimized for finding narrow technical details might be intimidating for the beginner that does not even know which question to ask. A beginner would probably need a more defined route that increases the likelihood that they will encounter the important general information. Even though not part of the original objective of the study, this observation offers an advice that is applicable in many situations: know your audience. When you are creating documentation it is very important to think about who is going to read it and try to understand their needs.
Interview Commentary

One expert user reported that they were likely to navigate documentation using search commands. This indicates that expert users might find less use in visualizations. Another experienced user was of the opinion that visualizations are very helpful. That subject reported that linking the concepts to a picture made the concept easier to remember. This means that the data about experienced users benefit is inconclusive. Comparing two groups of various experience levels could be a good starting point for a study to learn more about this issue. Considering both these perspectives is important when writing documentation.

UML Critique

The Unified Modeling Language is one approach for visualizing code. Having a unified language to express ideas in is very useful but there are a few important drawbacks that hinders its usefulness. My main criticism of UML is that it tries to do two things at the same time: communicate to humans as well as computers. I think it is more useful to employ tools specifically made for each purpose. An example could be writing on a whiteboard during a meeting. The ideas can come across without the complex rules of UML. When the computer needs instructions we already have code which contain the exact structure, in greater detail than UML. There are a few scenarios where a middle of the road approach is the right choice. When developers have to agree on a class structure remotely or in the case of sign-offs to other departments or outsourced labour. UML can decrease the ambiguity of conceptual whiteboard models and they can cause a greater amount of accountability on the part of the developers. I do however think these scenarios are less common now because of the rise of agile methodology.
Conclusion

The result suggests that dataviz can help users more easily understand documentation. The meaning of understanding is ambiguous and it has to be clarified in this context. It does not imply mastery of the subject or even remembering it for later. The user understands the documentation if she can use it to answer a question. This study also indicates that users are generally positive toward dataviz in docs.

Concluding statement

This exploratory study shows that further studies about dataviz in connection to documentation are worthwhile pursuits. As society is increasing its reliance on technology there is a greater need for technical understanding for people of all walks of life.

This is necessary for many reasons. Companies want to increase the amount of value adding time. If employees learn faster they will provide value quicker. It is also necessary in a greater perspective of enabling a greater portion of the population to access knowledge.

As jobs continue to be eliminated people will have to learn new skills. Dataviz can be a key component in giving less privileged parts of the population an introduction to highly technical topics. This can enable them to learn new skills that there will be a market for in the future. If the barrier to education can be successfully lowered it can help the pursuit of higher equality among people. This is the ultimate goal in which dataviz can play a small part.
Future Studies

The suggested studies outlined in this section are summarized below.

Summary of Suggested Studies

- Create a ontology that describes the most common software patterns.
- Build a tool for parsing source code into a model based on the ontology.
- Perform studies to determine which type of diagram is most effective at illustrating the relationships described by the ontology.
- Build a tool that automatically generates the most suitable visualization based on the ontological model generated by the parser.

The Need for Ontologies

Ontologies are the explicit formal specifications of the terms in the domain and relations among them (Gruber, 1993). Ontologies can also be understood as meta-data that provides a vocabulary of terms, each with an explicitly defined and machine processable semantics (Van Harmelen et. al., 2001). Class-hierarchies from Object-Oriented programming are similar to ontologies but with a small difference. The objective of a class-hierarchy is to capture the structure of source-code that evolves slowly. The content of an ontology can change a lot but the structure remains the same. A great example of an ontology is the Semantic Web Standard defined by W3C which aims to describe different types of content on the web to enable better machine processing.

The importance of automatism in keeping documentation updated increases as the industry is embracing shorter software cycles in order to decrease time-to-market (Klieber et. al., 2009). Software documentation (in combination with the source code) often formulates an ontology to facilitate learning, understanding and composability. One example of an ontology that might be familiar to software engineers are components. A component can have different meanings in different contexts and it’s important that the documentation makes
The ontology explicit. A great example of a well defined ontology is found in the documentation of the JavaScript framework React (developed by Facebook). Under the heading “Components and Props” a component is defined as follows: “Components let you split the UI into independent, reusable pieces, and think about each piece in isolation. Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called “props”) and return React elements describing what should appear on the screen.” (React Documentation, 2018)

The quote from the React docs clearly explain what a component is in this ontology. This ontology can of course be described in more detail using tables or a relational tree diagram.

There are a few reasons why ontologies are helpful:

- Enables the sharing of common understanding about the structure of information among both people and software agents.
- Enables reuse of domain knowledge.
- Makes domain assumptions explicit.
- Separates domain knowledge from operational knowledge.
- Enables analysis of domain knowledge.

(Noy & McGuinness, 2001)

Machine Learning

Van Harmelen et. al. (2001) describes how ontology will be important for the development of the semantic web. Almost twenty years later we stand before another development that can have even broader implications: machine learning. Machine learning is a collection of techniques based on statistics that give computers the ability to progressively improve performance on a given task (learning) using data without being explicitly programmed (Samuel, 1959). It is an old field but thanks to exponentially increasing amounts of data and processing power the field has recently been able to produce incredible results. From beating the best chess and go players to diagnosing difficult diseases.
Machine learning can potentially be applied to almost every field and I think it could have a great impact on the process of writing documentation. Ontologies enable machine processing and therefore it is an important aspect to consider as the importance of machine learning increases.

The paper *Ontology-based Information Visualisation* (van Harmelen et. al., 2001) describes how ontologies are represented by visualizations. If dataviz for documentation can be categorized as an ontology of the most efficient ways of illustrating certain relationships, that would have numerous benefits. This could possibly help AI with the task of writing documentation and making dataviz that matches it. That would automate a portion of the technical writing process which could then be finalized by a human domain expert.

Other problems that software engineers experience in regards to documentation can also be addressed with machine learning. These issues are presented in this paper because they strengthen the argument for the creation of a common pattern ontology, which could help both dataviz and machine learning.

Lethbridge et al. (2003) lists several issues that software engineers reported about their day to day experience of documentation. The list below highlights the problems I want to address.

**Documentation Problems**

- Documentation is often out of date.
- Documentation is often poorly written.
- Finding useful content in documentation can be so challenging that people might not try to do so.
- Much mandated documentation is so time consuming to create that its cost can outweigh its benefits.
- A considerable fraction of documentation is untrustworthy.

(Lethbridge et al., 2003)

Machine learning can potentially solve or mitigate most of the problems listed above. If an algorithm can analyze the text corpus of the source code and classify
it according to the pattern ontology then a basic outline of the documentation can be generated automatically. The source code analysis should be part of the build system so that every piece of code that gets added to the codebase is analyzed automatically. This would mitigate several of the problems listed by Lethbridge et al.. The documentation would never be out of date. The documentation would be written in a very sparse style consisting of basic descriptions of the functions, their input and output and perhaps pre-generated text descriptions based on the common patterns described by the ontology. This can be used as a starting point for a domain expert to review and change as needed.

There are risks that should be accounted for in regards to this development. The users that will ultimately use the documentation should be involved in the process of integrating machine learning as their feedback is invaluable. It’s important for the users to feel involved in the process of laying a foundation that can be adjusted by the machine learning algorithm. One way of achieving this is to let the AI show different versions of the same documentation and adjust them based on what gets most interaction, clicks or questions. The AI can then fine-tune the variables while the users provide feedback by using the documentation. In this way humans can get a sense of ownership while the power of machine learning can also be utilized. Human-friendly, computer-generated documentation might become a reality if two ontologies about source code and dataviz could be combined and then interpreted by an algorithm.

It must be noted that automation of documentation writing is nothing new but the projects I’ve found at the time of writing are still reliant on the programmer commenting the code. The software engineer is essentially defining her own ontology for each file. The drawback of this approach is that the maintainer might forget to change the comment as well as the source code which would lead
to out of date documentation. This is a manifestation of the fundamental programming issue of duplication which has a negative impact on scalability, maintenance, testing and refactoring.

I believe this study indicates that dataviz adds value to a lot of people and therefore its effect on learning in general has to be explored in-depth. Documentation is one area where dataviz can be applied but the applications can be much farther reaching.

Suggested studies

The analysis generated two types of suggestions for further studies: Dataviz focused and Machine Learning focused. The goal is improving documentation but the complexity and scale of the projects are different (machine learning being more complicated). They both build on the foundation of the common source code pattern ontology.

Dataviz Focus

- Develop a foundational ontology for common source code patterns.
- Determine which diagram types are most effective at illustrating the foundational source code patterns.

Machine Learning Focus

- Build a software tool utilizing a machine learning classification algorithm for classifying source-code according to the common pattern ontology.
  - Build an interpreter of the foundational source-code ontology that generates basic text descriptions about the source code. These descriptions can be used as a starting point for the documentation.
- Build a software tool for generating the correct diagram types based on the ontology.
## Appendices

### Data Tables

#### Preference

<table>
<thead>
<tr>
<th>Subject</th>
<th>Preference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1*</td>
<td>Data viz</td>
</tr>
<tr>
<td>2**</td>
<td>Data viz</td>
</tr>
<tr>
<td>3</td>
<td>No difference</td>
</tr>
<tr>
<td>4</td>
<td>Data viz</td>
</tr>
<tr>
<td>5</td>
<td>Data viz</td>
</tr>
<tr>
<td>6</td>
<td>Data viz</td>
</tr>
<tr>
<td>7</td>
<td>Text Only</td>
</tr>
</tbody>
</table>

#### Time to completion

<table>
<thead>
<tr>
<th>Subject</th>
<th>Time taken with dataviz</th>
<th>Time taken without data viz</th>
</tr>
</thead>
<tbody>
<tr>
<td>1*</td>
<td>10:36</td>
<td>9:39</td>
</tr>
<tr>
<td>2**</td>
<td>4:31</td>
<td>5:03</td>
</tr>
<tr>
<td>3</td>
<td>4:41</td>
<td>10:15</td>
</tr>
<tr>
<td>4</td>
<td>2:32</td>
<td>6:10</td>
</tr>
<tr>
<td>5</td>
<td>3:40</td>
<td>4:20</td>
</tr>
<tr>
<td>6</td>
<td>4:46</td>
<td>20:00</td>
</tr>
<tr>
<td>7</td>
<td>7:49</td>
<td>12:26</td>
</tr>
</tbody>
</table>

*Added 20 min browsing time, Removed one question.

**Changed questions to “was this easy to find?” YES/NO
Questionnaire Results

<table>
<thead>
<tr>
<th>Subject</th>
<th>Q result with data viz</th>
<th>Q result without data viz</th>
</tr>
</thead>
<tbody>
<tr>
<td>1*</td>
<td>4/4</td>
<td>3/4</td>
</tr>
<tr>
<td>2**</td>
<td>3/3</td>
<td>3/3</td>
</tr>
<tr>
<td>3</td>
<td>3/3</td>
<td>3/3</td>
</tr>
<tr>
<td>4</td>
<td>3/3</td>
<td>3/3</td>
</tr>
<tr>
<td>5</td>
<td>3/3</td>
<td>3/3</td>
</tr>
<tr>
<td>6</td>
<td>3/3</td>
<td>2/3</td>
</tr>
<tr>
<td>7</td>
<td>2/3</td>
<td>3/3</td>
</tr>
</tbody>
</table>

Prior Experience

<table>
<thead>
<tr>
<th>Subject</th>
<th>Prior Experience of programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>1*</td>
<td>None</td>
</tr>
<tr>
<td>2**</td>
<td>A lot</td>
</tr>
<tr>
<td>3</td>
<td>None</td>
</tr>
<tr>
<td>4</td>
<td>None</td>
</tr>
<tr>
<td>5</td>
<td>None</td>
</tr>
<tr>
<td>6</td>
<td>Little</td>
</tr>
<tr>
<td>7</td>
<td>A lot</td>
</tr>
</tbody>
</table>

Interview Questions

1. How did it feel to participate in this study?
2. What is your previous experience with programming related topics?
3. Did you perceive a difference between the versions?
4. Do you have a preference?
5. What would you like to see more of?
6. What would you like to see less of?
7. Would you like to see more Swedish in the documentation?
8. Any other feedback?
Ware’s Design Principles

The following are design principles extracted from Colin Ware’s book *Information Visualization: Perception for Design* (2012). The principles were formulated as a result of his extensive review of current research about human perception. The principles were used as guidelines in the design of the visualizations.

G1.2 Important data should be represented by graphical elements that are more visually distinct than those representing less important information.

G1.3 Greater numerical quantities should be represented by more distinct graphical elements.

G.14 Graphical symbol systems should be standardized within and across applications.

G1.7 Unless the benefit of novelty outweighs the cost of inconsistency, adopt tools that are consistent with other commonly used tools.

G3.2 Consider using Cornsweet contours instead of simple lines to define convoluted bounded regions.

G3.3 Consider using adjustments in luminance contrast as a highlighting method. It can be applied by reducing the contrast of unimportant items or by locally adjusting the background to increase the luminance contrast of critical areas.

G3.4 Use a minimum 3:1 luminance contrast ratio between a pattern and its background whenever information is represented using fine detail, such as texture variation, small-scale patterns, or text.

G4.1 Use more saturated colors when color coding small symbols, thin lines, or other small areas. Use less saturated colors for coding large areas.

G4.6 If large areas are defined using nearly equiluminous colors, consider using thin border lines with large luminance differences > (from the colors of the areas) to help define the shapes.
G4.7 If using color saturation to encode numerical quantity, use greater saturation to represent greater numerical quantities. Avoid using a saturation sequence to encode more than three values.
G4.11 Consider using red, green, yellow, and blue to color code small symbols.
G4.12 For small color-coded symbols, ensure luminance contrast with the background as well as large chromatic differences with the background.
G4.13 If colored symbols may be nearly isoluminant against parts of the background, add a border having a highly contrasting luminance value to the color, for example, black around a yellow symbol or white around a dark blue symbol.
G4.14 Ensure variation in the yellow-blue direction.
G4.15 Do not use more than ten colors for coding symbols if reliable identification is required, especially if the symbols are to be used against a variety of backgrounds.
G4.16 Use low-saturation colors to color code large areas. Generally, light colors will be best because there is more room in color space in the high-lightness region than in the low-lightness region.
G4.17 When color coding large background areas overlaid with small colored symbols, consider using all low-saturation, high-value (pastel) colors for the background, together with high-saturation symbols on the foreground.
G4.18 When highlighting text by changing the color of the font, it is important to maintain luminance contrast with the background. With a white background, high-saturation dark colors must be used to change the font color. Alternatively, when changing the background color, low-saturation light colors should be used if the text is black on white.
G5.1 To minimize the cost of visual searches, make visualization displays as compact as possible, compatible with visual clarity. For efficiency, information nodes should be arranged so that the average saccade is 5 degrees or less.
G5.6 Use strong preattentive cues before weak ones where ease of search is critical.
G5.8 Use positively asymmetric preattentive cues for highlighting.
G5.9 For highlighting, use whatever feature dimension is used least in other parts of the design.

G5.11 To make symbols in a set maximally distinctive, use redundant coding wherever possible; for example, make symbols differ in both shape and color.

G5.12 If symbols are to be preattentively distinct, avoid coding that uses conjunctions of basic graphical properties.

G5.13 When it is important to highlight two distinct attributes of a set of entities, consider coding one using motion or spacial grouping and the other using a property such as color or shape.

G5.16 When designing a set of glyphs to represent quantity, mapping to any of the following glyph attributes will be effective: size, lightness (on a dark background), darkness (on a light background), vividness (higher saturation) of color, or vertical position in the display.

G5.17 Ideally, use glyph length or height, or vertical position, to represent quantity. If the range of values is large, consider using glyph area as an alternative. Never use the volume of a three-dimensional glyph to represent quantity.

G6.1 Place symbols and glyphs representing related information close together.

G6.2 When designing a grid layout of a data set, consider coding rows and/or columns using low-level visual channel properties, such as color and texture.

G6.3 To show relationships between entities, consider linking graphical representations of data objects using lines or ribbons of color.

G6.4 Consider using symmetry to make pattern comparisons easier, but be sure that the patterns to be compared are small in terms of visual angle (<1 degree horizontally and <2 degrees vertically). Symmetrical relations should be arranged on horizontal or vertical axes unless some framing pattern is used.

G6.5 Consider putting related information inside a closed contour. A line is adequate for regions having a simple shape. Color or texture can be used to define regions that have more complex shapes.

G6.6 To define multiple overlapping regions, consider using a combination of line contour, color, texture, and Cornsweet contours.
G6.7 Use a combination of closure, common region, and layout to ensure that data entities are represented by graphical patterns that will be perceived as figures, not ground.

G6.22 When developing glyphs, use small, closed shapes to represent data entities, and use the color, shape and size of those shapes to represent attributes of those entities.

G6.23 Use connecting lines, enclosure, grouping and attachment to represent relationships between entities. The shape, color, and thickness of lines and enclosures can represent the types of relationships.

G6.24 As an alternative to arrows to represent directed relationships in diagrams, consider using tapered lines with the broadest end at the source node.

G6.28 Consider using a treemap to display tree structured data where it is only necessary to display the leaf nodes and where it is important to display a quantity associated with each leaf node.

G6.29 Consider using a node-link representation of a tree where the hierarchical structure is important, where internal (non-leaf) nodes are important, and where quantitative attributes of nodes are less important.

G8.17 Consider using pictorial icons for pedagogical purposes in infographics. Use them only where a canonical or culturally defined image is available.

G8.18 When a large number of data points must be represented in a visualization, use symbols instead of words or pictorial icons.

G8.19 Use words directly on the chart where the number of symbolic objects in each category is relatively few and where space is available.

G8.20 Use Gestalt principles of proximity, connectedness, and common region to associate written labels with graphical elements.

G9.1 Use methods based on natural language (as opposed to visual pattern perception) to express detailed program logic.

G9.2 Graphical elements, rather than words, should be used to show structural relationships, such as links between entities and groups of entities.

G9.3 Use methods based on natural language (as opposed to visual pattern perception) to represent abstract concepts.

G9.4 “Pick the right type of medium for the right information”
G9.5 Place explanatory text as close as possible to the related parts of a diagram, and use a graphical linking method.

G10.5 Consider providing an overview map to speed up the acquisition of a mental map of a data space.

G10.6 Consider providing a small overview map to support navigation through a large data space.

G11.1 Design cognitive systems to maximize cognitive productivity.

(Ware, 2012)
References


[Accessed 18-04-22]


https://www.technologyreview.com/s/515926/how-technology-is-destroying-jobs/
[Accessed 18-04-23]


