Optimizing API Documentation

Some Guidelines and Effects

Michael Meng
Department of Business Studies and Information Sciences, Merseburg University of Applied Sciences, Merseburg, Germany
michael.meng@hs-merseburg.de

Stephanie M. Steinhardt
Department of Business Studies and Information Sciences, Merseburg University of Applied Sciences, Merseburg, Germany
mail@steinhardt-dokumentation.de

Andreas Schubert
Department of Business Studies and Information Sciences, Merseburg University of Applied Sciences, Merseburg, Germany
andreas.schubert83@gmail.com

ABSTRACT

The growing importance of APIs creates a need to support developers with effective documentation. Prior research has generated important findings regarding information needs of developers and expectations they form towards API documentation. Several guidelines have been proposed on the basis of these findings, but evidence is lacking whether such guidelines actually lead to better documentation. This paper contributes the results of an empirical test that compared the performance of two groups of developers working on a set of pre-defined tasks with an API they were unfamiliar with. One group had access to documentation which was optimized following guidelines for API documentation design proposed in the literature whereas the other group used non-optimized documentation. Results show that developers working with optimized documentation made fewer errors on the test tasks and were faster in planning and executing the tasks. We conclude that the guidelines used in our study do have the intended effect and effectively support initial interactions with an API.

CCS CONCEPTS

• **Software and its engineering** → Software creation and management; Software post-development issues; Documentation; • **Human-centered computing** → Human computer interaction (HCI); HCI design and evaluation methods; User studies.

KEYWORDS

API documentation, user studies, information design, usability

ACM Reference Format:
Michael Meng, Stephanie M. Steinhardt, and Andreas Schubert. 2020. Optimizing API Documentation: Some Guidelines and Effects. In *Proceedings of the 38th ACM International Conference on Design of Communication (SIGDOC '20), October 03, 04, 2020, Denton, TX, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3380851.3416759

SIGDOC '20, October 03, 04, 2020, Denton, TX, USA

ACM ISBN 978-1-4503-7525-2/20/10…$15.00
https://doi.org/10.1145/3380851.3416759

1 INTRODUCTION

The growing importance of Application programming interfaces (APIs) in modern software development creates a need to support developers in learning new APIs. A dimension that has become critical in this respect is API usability, i.e. the extent to which APIs fit the needs and expectations of the target audience [1, 2]. API documentation plays an important role in making APIs more usable. As pointed out by Myers & Stylos [3], API documentation is essential in mitigating API usability problems that cannot be resolved via API design decisions or tool support. Moreover, it has been shown that poor documentation leads to frustration, loss of time and can cause developers to abandon an API [4, 5].

Various research efforts have been made to improve API documentation. One line of research has focused on tools that facilitate access to information, e.g. by recommending information that could be relevant in the context of a certain task [6]. A different approach – which also guides the research presented here – has been to identify content elements that are critical for learning an API, and dimensions of information design that support developers in locating and processing this content [4, 7].

Prior research following this approach has identified various aspects regarding content and design that are important from a developer perspective. Based on such findings, a number of guidelines for optimizing API documentation have been proposed [7–9]. However, it is unknown whether these guidelines have the intended effect and effectively support developers learning an API.

The goal of this paper is to contribute the results of an empirical test that examined effects of applying guidelines for API documentation design we proposed in earlier work on developer performance [9]. The test was designed to find out whether changes to the documentation in response to the guidelines lead to more successful initial interactions with the API, with focus on success (i.e., errors made on a set of pre-defined tasks) and effectiveness (i.e., the time required to solve the tasks).

2 LITERATURE REVIEW

2.1 Research on API documentation

API documentation [4, 10] and documentation on software systems in general [11, 12] have often been described as poor and incomplete. In fact, Robillard [13] and Robillard & DeLine [7] identified documentation as main obstacle for learning a new API. In response to the findings of Robillard & DeLine, much research has been devoted to explore what users of API documentation want, expect and need by conducting interviews and surveys [4, 14], by observing

developers solving tasks with a new API [8, 9, 15], and by analyzing API documentation developers produce themselves [16] or refer others to [17]. In this section, we summarize main results of these research efforts. Guidelines for optimizing API documentation that have been derived from this research will be presented in the next section.

2.1.1 API overview. Developers approach API documentation with two broad goals in mind: orientation (e.g. to find out whether the API offers solutions to a specific problem) or learning (e.g. to try out features or to solve a specific programming task). Regardless of the goal, developers initially expect high-level information that provides a brief overview of the API, e.g. explains the scope of the API, the main features it offers, and technical requirements [7, 14, 16].

2.1.2 Opportunistic vs. systematic approaches to learning and programming. Developers use different strategies when starting to work with a new API. The different strategies have been described in terms of developer personas presented in Clarke [18]. Clarke distinguishes between opportunistic, pragmatic and systematic developers. Opportunistic developers work in an exploratory manner and attempt to start working on specific tasks as soon as possible. They look for code example that relate to their problem and attempt to extend those examples step by step. Systematic developers try to understand how the API works before diving into the details of a task. They deal with concepts even if the concepts do not relate directly to the task at hand. Pragmatic developers combine elements of both strategies.

2.1.3 Importance of task orientation. The importance of task orientation belongs to the core insights of Minimalism [19] and this insight is relevant in the context of API documentation as well. Developers approach documentation with a problem or task in mind [14]. They do not intend to learn an API as a whole, even if they follow a systematic approach to learning. Learning a new API proceeds in a task-directed manner [20], and can be described as a form of "just-in-time learning" [16].

2.1.4 Role of conceptual background knowledge. Background knowledge of the domain covered by an API (e.g. business processes involved in B2C or B2B commerce, procurement or enterprise resource planning) greatly facilitates the learning process [8, 21]. Furthermore, links to documentation developers include in answers on StackOverflow often point to description of concepts, emphasizing the importance of conceptual background information as well [17]. However, it is a challenge that developers tend to ignore documents which focus on conceptual information, hence potentially convey the domain-related background knowledge of an API, with the reluctance to refer to concepts documents apparently being independent of the learning strategy the developers adopt [14].

2.1.5 Selective access to documentation. Like most users, developers use API documentation in a highly selective manner [12, 14]. In addition, Meng et al. [9] observed that the developers participating in their observation study used documentation to roughly the same extent, but differed with respect to the amount of attention devoted to individual sections, in particular the concepts section. Findings like this suggest that documentation has to provide means to facilitate selective access, for example by clearly separating text and code sections, as well as by providing a powerful search function and consistent navigation.

2.1.6 Identifying entry points. A main challenge for developers getting into a new API is to identify appropriate entry points and to relate specific use cases or business processes to particular API elements [7]. This involves discovering API resources or classes that represent particular business objects, as well as discovering functions and methods to access and transfer data across business objects. Therefore, specific means should be taken to help developers map business objects and processes to API resources, e.g. by highlighting API elements mentioned in a textual explanation in a corresponding code example.

2.1.7 Role of code examples. Several studies identified code examples as an important learning resource [1, 13] and as an important starting point for solution development [22, 23]. Developers often use code examples that seem to address the task at hand, and then extend and modify the code example to make it fit the current needs.

2.2 Guidelines for API documentation design

This section summarizes and slightly extends the heuristics and guidelines for API documentation design proposed in Meng et al. [9]. The guidelines derive from findings as discussed in the previous section. Note that the guidelines are not intended to be exhaustive. For additional guidelines, see Jeong et al [8] and Robillard & DeLine [7]. Note also that the guidelines are merely "rules of thumb" and can be implemented in various ways. They do not determine a specific solution but are intended to guide the search for solutions.

2.2.1 Heuristic 1: Enable efficient access to relevant content. The design of API documentation should facilitate effective and selective access to content that is relevant to developer tasks.

- Guideline 1.1: Organize the content according to main usage scenarios supported by the API and typical tasks that developers will solve with the API.
- Guideline 1.2: Present important conceptual information integrated with the description of tasks or usage scenarios where knowledge of these concepts is needed.
- Guideline 1.3: Provide transparent and consistent navigation options and a powerful search function. If integrating search is difficult, e.g. due to technical limitations, facilitate simple search by presenting information on a single page instead of distributing the information across multiple linked pages.
- Guideline 1.4: Structure the content at section level consistently. Use appropriate verbal and visual signaling techniques to make the structure transparent.

2.2.2 Heuristic 2: Facilitate initial entry into the API.. API documentation design should support developers in identifying entry points into the API. Relating particular tasks or usage scenarios to specific elements of an API are key issues for successful API learning.

- Guideline 2.1: Provide clean and working code examples. Code examples need to be complete and accurate and should enable direct re-use via copy-and-paste.

- Guideline 2.2: Provide relevant background knowledge on the domain covered by the API and explain important concepts that can support the learning process.
- Guideline 2.3: Support developers in relating concepts to API elements. Signal to the developers how concepts are represented, e.g. by emphasizing relevant elements in a code example.
- Guideline 2.4: Provide a concise API overview that tells developers and other stakeholders the purpose of the API, its main features and important technical characteristics. Make sure the overview can be accessed easily.

2.2.3 Heuristic 3: Support different strategies for learning and development. API documentation has to serve different strategies for learning and development. In particular, API documentation has to address the information needs of developers that adopt an opportunistic and exploratory strategy.

- Guideline 3.1: Enable selective access to code, e.g. by using a multi-column layout and formatting means that clearly separate text from code.
- Guideline 3.2: Signal text-to-code relations in order to help developers mapping concepts to code. This reduces reading efforts and facilitates the identification of relevant information.
- Guideline 3.3: Provide important conceptual information redundantly. Present it wherever needed. If possible, present important conceptual information as part of source code comments to make sure developers focusing on code will discover and process it as well.
- Guideline 3.4: Enable fast and productive use of the API. Include code examples and integrate try-out functionality that can be used to test API elements immediately without much effort.

Beyond the guidelines presented here, it is necessary to ensure that API documentation meets standard quality criteria of technical documentation. Developers expect API documentation to be accurate and complete in the sense that it provides all information required to solve tasks with the API, building on the knowledge which they bring to the tasks. Moreover, developers expect documentation to be easily comprehensible, even when (and in particular if) the API is inherently complex.

2.3 Research Questions

The guidelines presented in the previous section are motivated by research based on introspection, self-report, analysis of documentation artefacts produced by developers and observation of actual developer activities. So far, no attempts have been made to demonstrate that these or similar guidelines proposed in the literature support better developer performance. However, understanding the effects of such guidelines and the design decisions they encourage is important both from a research and from a practitioner perspective [24]. The goal of our research was to contribute empirical data on this issue.

- (RQ1) Our first research question is to examine whether the guidelines for API documentation design presented in the previous section have the intended effect on developer performance when using documentation following these guidelines to learn a new API. In particular, we want to know whether the guidelines support more successful initial interactions with an API.
- (RQ2) Given that effects on developer performance can be determined, our study also aims to provide initial insights into how such effects come about. According to the model proposed by Guthrie et al. [25], interacting with procedural documents in order to solve tasks involves cognitive processes along two main dimensions: processes involved in locating, studying and comprehending information provided in a document, and processes involved in planning, preparing and executing tasks. Following this distinction, our study asks whether optimizations of API documentation on the basis of guidelines proposed in the literature primarily have an impact on locating and studying information, or on actually performing the tasks.

3 METHODS

3.1 Study Design and Materials

In order to address the research questions, we set up a study in which developers were asked to solve a set of tasks with an API that was unfamiliar to them. Whereas one group of developers worked with documentation that closely mirrored the original documentation of the API provider, the other group had access to an improved version of that documentation. Modifications of the documentation followed the guidelines discussed in the previous section. The setup enabled us to measure the joint effect of the guidelines – as implemented by our specific modifications – on variables that can inform us about developer performance, such as the time required to solve the tasks and the proportion of tasks solved correctly.

The purpose of the API selected for the test (http://shipcloud.io) is to facilitate the integration of online shops with the services of shipping providers, making it easier for shop owners to offer a broad range of shipping services to customers. The API, which we already used in an earlier study [9], is based on the REST paradigm, which means that business objects such as shipments, carriers or service rates are represented as resources that are addressable over the Internet via a URL and standard HTTP requests. For a nontechnical introduction to REST APIs, see Johnson [26].

Documentation and other resources to get started with the shipcloud API are provided on a developer portal (http://developers.shipcloud.io.. For our test, we partly rebuilt the developer portal using Flatdoc, a simple HTML-based site generator (https://ricostacruz.com/flatdoc/.. The rebuilt version of the developer portal included all parts of the original site that were relevant for the purposes of our test.

Two new version of the developer portal were created: a control version and an optimized version. The control version retained the structure of the original documentation, including all main content categories: "Concepts", "Recipes", "API Reference", "Integration", and "Samples." The structure of the sections within each content category, the sub-headings and all content elements were left unmodified. Mirroring the original site, the control version did not offer a search option.

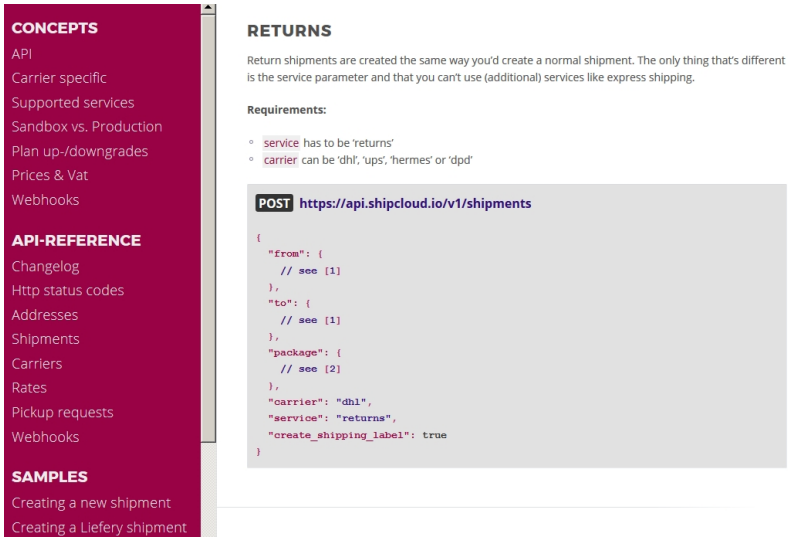

Figure 1: Example page of the API documentation serving as control version.

The optimized version was modified following the guidelines discussed above and included the following changes:

- A three-column layout separating navigation, textual descriptions and sample code was imposed in order to facilitate selective access to code. (guideline 3.1)
- All sections of the documentation were checked and modified if necessary to ensure consistent structure at section level. (guideline 1.4)
- Conceptual information was added, for example to explain the concept of a shipment, the stakeholders and resources involved in a shipment, as well as more specific concepts such as return shipment. In addition, a new section "Terms you need to know" was added to explain important terms. Note that this modification was restricted to sections of the documentation that developers would potentially use when solving the tasks. (guideline 2.2)
- When conceptual information was added, it was integrated with the description of use cases where it was needed. (guideline 1.2, 3.3)
- Graphics and visualizations were added to help developers map concepts and business processes onto API elements. Process components and their relation to stakeholders were sometimes visualized using animation. (guideline 2.3)
- Likewise, visual signaling and color coding were used to relate API elements mentioned in a textual description to the respective API elements in the code example accompanying the description (e.g., parameters in the JSON request body). This was intended to support selective reading and to help developers identify and select relevant information. (guideline 3.2)
- The structure of the documentation was changed to increase task orientation and to better reflect important use cases. Instead of using "Concepts", "Recipes", "Samples", "Integration" and "API reference" as main categories, we used: "How the API works", "How to run the API" and "How to use the API." (guideline 1.1)
- A short overview of main API features and use cases served by the API was added. This overview was presented on the starting page to make it easily accessible. (guideline 2.4)
- A sample request for creating a shipment was added as part of the overview. This sample represents a central use case. The sample supports developers in getting productive fast. (guideline 3.4)
- All code examples were reworked to eliminate placeholders and to make them ready for copy-and-paste use. (guideline 2.1)

An example page of the control version is shown in Figure 1. The page describes how to configure a return shipment. Figure 2 shows how this page changed following the optimization process, including the three-column layout, the graphic that was added to explain the concept and to demonstrate how the concept is related to parameters of the JSON code transferred via the request body. Moreover, placeholders were removed from the corresponding code example and visual signals added to highlight a parameter that is particularly relevant in the context of the current topic and mentioned both in the graphic and in the textual description.
Note that for practical reasons, guideline 1.3 could only be implemented in a limited way. According to guideline 1.3, a powerful search function should be offered to facilitate selective access to relevant content. We added a search function to the optimized version. However, due to technical restrictions, only a simple search function was possible.

3.2 Participants

We recruited 22 participants (21 male, 1 female) for our test. Seventeen participants were recruited from software companies that we contacted leveraging our personal network. To obtain the participants, we asked the management whether they would be interested

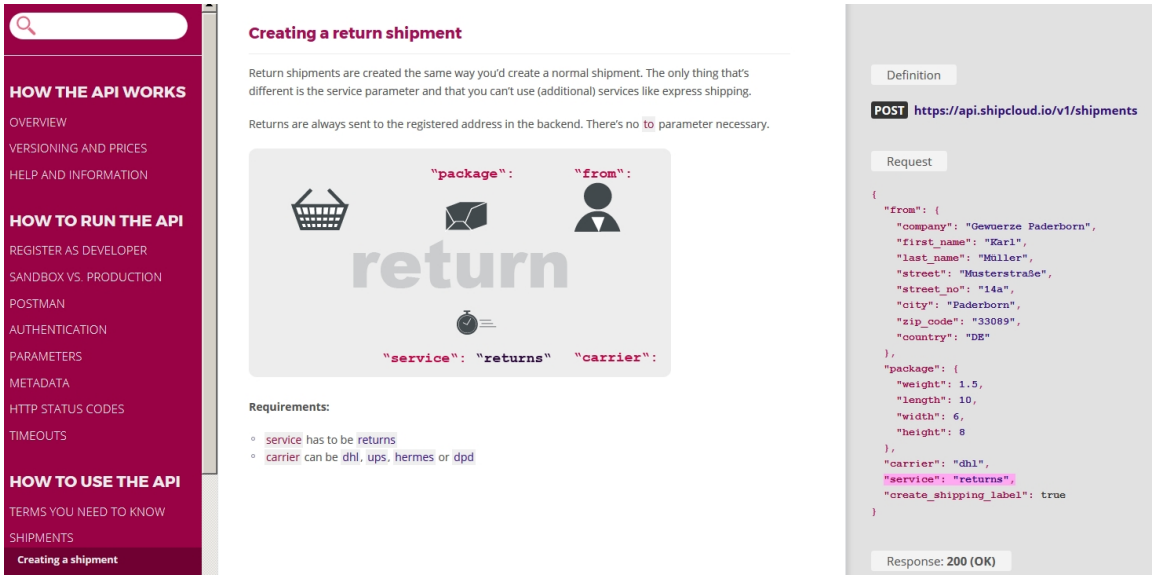

Figure 2: Example page showing the page of Figure 1 in the optimized version of the documentation.

in supporting our study and requested volunteers. Since the API used for our test does not presuppose any specific skills with respect to programming language or technology, the only inclusion criteria we defined was for participants to have practical experience with REST APIs and to be unfamiliar with the shipcloud API. Volunteers that were proposed by the partner companies as test participants included both employees as well as students working as interns. Five additional participants (all advanced computer science students) were recruited from our university. All participants received 10 EUR for completing the test.

In a questionnaire administered to all participants before the test, ten participants identified themselves as "developer", two as "team lead" and ten as "student." Participants were assigned randomly to one of the two experimental groups: the group working the control version of the documentation ("control") and the group working with the optimized version ("optimized"). We defined a test schedule beforehand and assigned developers to groups depending on the order in which they appeared.

The mean age of participants was 33.4 years (Min: 21, Max: 53). The two experimental groups did not differ with respect to age, with a mean age of 33.1 for the control group and of 33.7 for the group working with the optimized documentation. In order to assess the level of programming expertise of our participants, we asked for developer experience in general, and for developer expertise in a professional context. The two groups were similar with respect to both parameters (general developer experience in years: control = 9.3, optimized = 9.5; developer experience in a professional context: control = 8.0 years, optimized = 7.8). Both groups were also similar with respect to experience with REST APIs, which participants were to assess on a scale ranging from 1 (= "no experience") to 5 (= "highly skilled"), with means of 3.36 for both groups.

In addition to developer experience and experience with REST APIs, we also assessed e-commerce experience. First, we asked whether participants had gained any developer experience in e-commerce, for example by working as a developer for an online shop or a company developing online shop software. In both groups, six participants indicated to have such experience, and five to have no such experience. We also asked participants to assess their e-commerce developer experience on a scale ranging from 1 (= no experience) to 5 (= highly skilled). Means obtained were 2.0 for the control group and 1.8 for the group working with optimized documentation. None of the between-group differences was statistically significant.

3.3 Procedure

Tests were conducted in the usability lab of our university for the university students, and on-site at conference rooms provided by the partner companies for industry participants. Before starting the test, we obtained informed consent from participants using a document which explained the study, the procedure and policies regarding processing personal data. Participants were also informed that they could terminate the test at any point without having to give a reason. Note that no IRB or similar institution is in place at our university.

Test sessions started with a short questionnaire to obtain data regarding age, gender, developer experience and experience with e-commerce and the shipcloud API. Afterwards, participants received test instructions and were shown the resources available to them during the test. When participants had completed the test tasks, a second questionnaire was administered which requested them to rate the structure and the overall quality of the API documentation. Moreover, a short, semi-structured interview was conducted to provide participants with the opportunity to comment on positive and negative aspects of the documentation. Participants were then debriefed, which ended the session. Test sessions lasted between 40-60 minutes.

The test consisted of five tasks that covered typical API use cases, such as creating return and express shipments, requesting pickup for a shipment at a predefined time and directing a shipment to a dedicated drop-off station. The tasks were identical to the tasks used in our earlier study [9]. The tasks did not require any programming. Rather, participants had to identify the appropriate endpoints, resources and parameters. Moreover, the tasks required to add request body information to the API calls using JSON notation. To facilitate the configuration and execution of API calls, participants had access to a Postman client that was prepared with respect to authentication information and the request body format. Tasks were provided in written form. Along with the tasks, participants received a sheet summarizing basic activities in Postman. Besides Postman, participants were provided with the shipcloud API documentation and the shipcloud dashboard, each presented on a separate browser page. Participants were free to use the documentation at any point they wanted.

During the test session, we took audio and screen recordings of all participant activities. In addition, we recorded participant's eye movements using a video-based eye tracker (SMI 250mobile, SensoMotoric Instruments) attached to the monitor of the test laptop. Recording of screen activities and eye movements started immediately before participants received the task sheet. Recording stopped after the last task had been completed. To record the interviews performed after the tests, the retrospective think-aloud (RTA) function of the software package BeGaze was used.

3.4 Data Analysis

Data analysis was based on the screen videos with eye tracking data overlaid. Screen videos were loaded into the INTERACT 17 (Mangold International) software for manual coding. Two coding schemes were applied:

- **Task**. The code for "Task" provided the basis for analyzing accuracy on task and task duration. As part of the instruction, participants were asked to notice the experimenter when they had completed a task and when they started to work on a new task. Using this information, we coded interval events for task 1 to task 5 ranging from the beginning to the end of each task.
- **Window**. The coding schema "Window" was used to determine the time participants spent in the documentation and the time they spent in the different types of resources outside the documentation. Values of the coding schema "Window" were applied on the basis of the eye tracking data. On the one hand, we coded interval events whenever participants viewed sections within the shipcloud API documentation. We also coded events to mark intervals in which participants viewed resources outside the API documentation, such as the Postman client, the shipcloud dashboard or other resources (e.g. when participants opened a browser window to look up an unknown term in an online dictionary).

Each coding schema was applied in a separate coding run. Coding was always performed by a single researcher only.

3.5 Results

Quantitative data analysis focused on three dependent variables: accuracy on tasks, time on tasks and time spent in documentation versus time spent in resources outside the documentation. All statistical tests were conducted using the R software package, version 3.6.1 [27]. The data were analyzed with mixed-effects regression modeling [28] using the lme4 R package [29]. Mixed-effects models can be used to analyze both continuous variables (such as time on task) as well as categorical variables (such as task accuracy, scored as "correct" or "incorrect"). Moreover, they make it possible to capture effects of multiple random factors in a single analysis, such as random effects due to systematic differences between participants or experimental tasks in the current study. For an in-depth discussion of advantages of mixed-effects modeling, see Baayen et al. [28] and Jaeger [30]. A non-technical introduction can be found in Balling [31]. For applications of mixed-effects modeling in the context of usability studies, see Balling [32] and Meng [33].

Note that there are different proposals with respect to the construction of random effect terms in a mixed model. Following Matuschek et al. [34], we attempted to identify the most parsimonious model predicting the observed data pattern.

3.5.1 Accuracy on tasks. The five tasks constructed for the test represented typical API use cases. However, they differed in complexity. We therefore defined 3-5 sub-tasks per task in order to take the varying degree of complexity between the tasks into account, resulting in a total of 18 sub-tasks. For example, task 1 ("Creating a shipment that is delivered to a specific drop station") involved the sub-tasks "Identifying the resource to create a shipment request", "Identifying and modifying the correct parameter to have the shipment delivered to the specific drop station", "Constructing the URL to invoke the resource correctly." Task 2 ("Creating a return shipment for the shipment just created") involved the sub-tasks "Identifying and modifying the parameter that defines a return shipment", "Changing the addresses [sender becomes recipient and vice versa]" and "Constructing the URL to invoke the resource correctly." We then checked on the basis of the screen recordings whether subtasks were solved correctly or not, using binary coding. Sub-tasks solved correctly were scored as 1, sub-tasks that were not completed correctly or only with the help of the experimenters were scored as 0. Hence, for task 1, a total of three points was possible if participants solved all sub-tasks completely. Often, task solution was partially correct in that only some of the sub-tasks were solved correctly. For example, for task 1, participants sometimes identified the correct resource and correctly constructed the URL to invoke the resource, but failed to identify and modify the parameter that would have the shipment delivered to the drop station, resulting in a score of 2 out of 3.

Results for accuracy on tasks (reported as percentage values) were as follows. The overall mean was 80.29 (SD=14.09). Accuracy was higher for the group working with optimized API documentation (M=86.35, SD=11.46) compared to the control group (M=74.24, SD=14.32). To assess the effect, we fitted logistic mixed-effects models using the glmer function of the lme4 package (Table 1). The analysis included "doctype" (optimized vs. control) as fixed effect, and "subject" and "sub-task" as random effects. Hence, every participants contributed 18 data points to the analysis, one for each of the

Table 1: Modelling results for task accuracy.

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	1.3222	0.3982	3.320	< 0.001
"optimized" vs. "control"	2.2210	0.8973	2.475	< 0.05

Formula: score_correct ~ doctype + (1|subject) + (1 + doctype|sub-task)

Table 2: Modelling results for time on tasks.

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	380.704	67.589	5.633	< 0.01
"optimized" vs. "control"	-16.831	38.717	-0.434	n.s.

Formula: time ~ doctype + (1|task)

18 sub-tasks (see Table 1 for the formula call of the model finally selected). As shown in Table 1, the observed difference between the two groups is significant.

3.5.2 Time on tasks. Did optimization have an effect on the time participants needed to solve the tasks? The analysis of time on tasks is based on the interval codes using the coding schema "Task." Hence, each participant contributed five data points, repesenting the times needed to solve each of the five test tasks. Prior to analysis, data were screened for outliers using boxplots and Q-Q plots (quantile-quantile plots). Since inspection of the plots suggested that there were outliers, data were corrected by removing all values more than 2.5 standard deviations away from the overall mean. This procedure removed 3 out of 110 data points, hence 2.8 % of the data.

The overall mean time participants spent on a test task was 369 s (SD=234). The numeric difference between the two groups was rather small, with means of 364 s (SD=224) for the group working with optimized documentation and a slightly higher 374 s (SD=245) for the control group. To check whether the difference between groups is significant, linear mixed-effects models were fitted which included "doctype" (optimized vs. control) as fixed effect, and "subject" and "task" as random effects (see Table 2 for the complete formula call of model finally selected). As shown in Table 2, the observed difference between the two groups is not significant.

3.5.3 Reading versus acting. To address RQ2, an additional analysis was performed to explore time on tasks further. The time participants spent on a task aggregates two different types of activities: the time they spent locating and studying information in the documentation, and the time they needed to plan and execute the tasks, such as configuring values in the Postman client or inspecting the response received from the API. If the efforts to optimize the API documentation work as expected, they should facilitate task execution. Predictions regarding the time participants spend in the documentation are less clear. On the one hand, it could be argued that the optimizations should support participants in locating relevant information, hence reduce the time they spend in the documentation. On the other hand, the optimizations increased the amount of content contained in the documentation, e.g. because graphics were added to explain important concepts. Moreover, participants could possibly devote more time to reading in the documentation because they find it more helpful.

To operationalize this question, we grouped time values depending on whether participants viewed some part of the documentation, or whether they viewed resources outside the documentation, such as the Postman client or the shipcloud dashboard. This analysis was based on the coding schema "Window." Evidently, this approach is only a rough approximation to activities that reflect "reading" and "acting." Participants also attend to the documentation while executing a task, e.g. when copying example code to be included as part of their own solution. However, the operationalization should be good enough as a start. Moreover, the criterion used to distinguish between time to read and time to act is sufficiently clear to enable reliable coding.

Data were analyzed on a by-task basis for each participant. Therefore, each participant contributed 10 data points: two data points (a viewing time for documentation and a viewing time for other resources) for each of the five test tasks. Prior to analysis, data were screened and corrected for outliers, which affected 7 out of 220 data points, corresponding to 3.3 % of the data. The overall mean obtained was 179 s (SD=124). Condition means, factor means and the respective standard deviations are shown in Table 3.

Data were subjected to a 2x2 analysis using linear mixed-effects models. The models included the factors "window" (documentation, other) and "doctype" (control, optimized) as fixed effects. Initial modeling started with "subject" and "task" as random effects. The final model arrived at and the respective model values are shown in Table 4.

The main effects were not significant, but there was a significant interaction of "window" and "doctype." To explore the interaction, we performed post-hoc Tukey HSD comparisons using the emmeans R package [35]. The comparisons revealed a significant difference between the conditions "control" and "optimized" for viewing times outside the documentation (189 s vs. 149 s, $t(205) = 2.20$, $p < .05$) whereas the difference was not significant for viewing times within the documentation (179 s vs. 202 s, $t(205) = -0.93$, n.s.). Hence, optimizations facilitated task execution, but did not significantly affect the time spent in the documentation.

Table 3: Mean viewing times in or outside the documentation.

doctype	window		
	documentation	other	mean
control	179 (129)	189 (133)	184 (131)
optimized	202 (127)	149 (105)	175 (119)
mean	190 (128)	168 (121)	179 (124)

All times in seconds, standard deviations in parentheses.

Table 4: Modelling results for viewing times in or outside the documentation.

	Estimate	Std. Error	t-value	Pr(>|t|)
(Intercept)	181.379	31.396	5.777	< 0.01
doctype	-12.103	14.807	-0.817	n.s.
window	-21.138	14.796	-1.429	n.s
doctype:window	-63.326	29.595	-2.140	< 0.05

Formula: time ~ doctype * window + (1|task)

3.5.4 Post-test questionnaire data. After participants had completed the tasks, we administered a questionnaire in order to get feedback on perceived task difficulty and documentation quality. The questionnaire consisted of three items, all associated with a scale ranging from 1 to 5.

The first question asked participants to rate the difficulty of the tasks (1=very easy, 5=very difficult). For the control group, a mean rating of 2.27 (median=2) was obtained, for the group working with optimized documentation a mean rating of 2.09 (median=2). The second question asked participants to evaluate the overall quality of the documentation (1=very bad, 5=very good). Mean values for the control group (3.73, median=4) and the group using optimized documentation (3.91, media=4) were again very similar.

Finally, participants were asked to evaluate the structure of the documentation (1=very bad, 5=very good). For both groups, the mean rating was 3.72 (median=4). None of the differences reached statistical significance.

3.5.5 Observations and comments. This section describes some observations and comments noted during the test or during post-hoc inspection of the screen videos, as well as participant comments. Note that interviews were not transcribed and no attempts were made at a more formal analysis. Observations and comments described here should be treated as anecdotal evidence.

We noted that developers differed greatly with respect to how they prepared for the first task. For example, P03 and P13 took much time to read through the documentation, whereas P04 and P08 jumped almost immediately into the first task. Whether this difference correlates with the distinction between opportunistic and systematic strategies for learning and development is an interesting question that future research should take up.

We also noted great variability among developers with respect to their ability to reflect and report on their strategies and their approaches to the tasks. While some developers clearly recalled the approach they followed towards individual tasks and sometimes even remembered individual steps, others had a lot more difficulty to provide this type of self-report.

Several participants working with optimized documentation mentioned that they appreciated the number of examples and that the examples could be re-used directly. Participants of the control group appreciated the number of examples provides as well, but noted that some examples were not ready to be re-used.

When asked what they liked or did not like about the documentation and which suggestions they would have for improvements, participants of the group working with optimized documentation expressed that they liked the presentation of explanations and code in separate columns. Participants from both groups mentioned that the documentation would benefit from more powerful search. Other issues that were mentioned by participants regardless of test group were that they missed more elaborate explanations of error codes, more comfortable navigation options and more links between related sections.

4 DISCUSSION

The results of our quantitative analyses suggest that the guidelines for API documentation design used in our study do have an effect on the initial interactions of developers with an API that is unfamiliar to them (RQ1). We observed an effect of documentation type on task accuracy: participants working with documentation that was optimized following the guidelines proposed in Meng et al. [9] made fewer errors on a set of pre-defined tasks. This suggests that initial interactions were more effective. The average time participants needed to solve the test tasks was slightly lower for participants working with optimized documentation, but this difference was not significant.

Having confirmed that the guidelines do have an overall effect, our second research questions attempted to reveal how this effect comes about. With reference to the model proposed in Guthrie et al. [24], we asked whether the guidelines for API documentation design

have an effect on actual task execution (e.g. by making task execution faster), whether they affect the amount of time participants spend searching and studying information in the documentation, or whether they would possibly affect both types of activities (RQ2).

Analysis of the viewing times of the participants in the documentation versus in resources other than documentation revealed an interactive effect. We observed a significant difference in viewing times in resources outside documentation, a variable which we used as a rough estimate to assess the effort involved in planning, preparing and executing the test tasks. Participants working with modified documentation spent less time in resources outside documentation, hence actually performed the test tasks faster. Viewing times in the documentation did not differ significantly between the groups and were slightly elevated for participants working with optimized documentation. As discussed above, only limited conclusions can be drawn from this comparison, since optimized documentation contained more content.

The fact that initial interactions were more successful when participants had access to optimized documentation did not lead to better post-test ratings of overall documentation quality and documentation structure, nor did it affect the assessment of perceived task difficulty. We assume that the absence of any effects here is due to the simple instruments used to measure perceived documentation quality and task difficulty.

Overall, we conclude that the guidelines for API documentation design are successful in the sense that they jointly drive performance of developers in the intended direction. Our results confirm the usefulness of design guidelines that have been developed on the basis of prior research exploring information needs of developers working with APIs, their expectations regarding the content and design of API documentation, and the strategies they adopt for learning and programming.

5 CONCLUSIONS

Given the growing importance of APIs and the critical role of API documentation for API usability, there is a need to support best practices that emerge in the field with evidence-based principles on which practitioners can base decisions regarding API documentation design. Research on API documentation has generated many specific findings related to content elements and content presentation. From these findings, several guidelines have been derived to help make API documentation fit the needs of the target audience. However, evidence is lacking whether the guidelines proposed in the literature actually have an effect on the performance of developers trying to get into a new API. The goal of the current study was to examine whether guidelines proposed in Meng et al. [9] lead to more successful initial interactions when developers have access to documentation following these guidelines.

Our study revealed two main results. First, we found that documentation that was modified based on the Meng et al. guidelines enabled developers to solve initial tasks with the API more successfully in that fewer errors were made. As a second finding, our study provides some initial evidence that the modifications which were applied to the documentation facilitated task execution, hence reduced the time developers needed to plan, prepare and execute the API calls. The modifications did not reduce the time developers spent searching and studying information in the documentation. However, as already pointed out in the Discussion section, there are confounding factors that make it difficult to draw conclusions from a comparison of the time participants spent viewing the documentation. Despite uncertainties that remain with respect to whether the design decisions affect only task execution or possibly cognitive processes involved in searching and reading as well, we conclude that the guidelines do have the intended effect and support developers trying to get into an API that is unfamiliar to them. If validated by future studies, the guidelines can hence contribute to evidence-based principles of API documentation design that enable practitioners in this domain to make informed decisions.

5.1 Limitations

A limitation of the current study arises from the small sample size that formed the basis for our quantitative analyses. The main reason why we decided to terminate data collection with 22 participants was that we faced severe difficulties in our attempts to recruit developers as participants from outside university. A solution could have been to scale up the study by including more students. But as argued by Ko et al. [36], relying primarily on students potentially introduces additional biases, as students are likely to have less experience and skills.

With a small sample size, statistical power is low and studies run the risk of missing effects that would emerge if a larger sample was used. Note that we were able to mitigate this problem to some extent by analyzing the dependent variables on a by-participant and by-task or sub-task basis. Although we are confident that the effects reported here are sufficiently robust to draw preliminary conclusions, follow-up studies using larger samples are clearly called for.

A related problem arises with respect to participant selection. We accepted every participant who was proposed to us by the partner companies supporting our study and who met our inclusion criteria. Hence, our participants all volunteered and are likely to have been more interested in the topic of API documentation than the average developer.

Another limitation concerns the type and complexity of the API used for the test. The shipcloud API is a small API of limited complexity that represents a specific API type: REST APIs. Although there is at least some evidence indicating that approaches to learning and programming adopted by developers do not differ depending on API type and complexity [14], additional research is needed to determine whether the conclusions drawn here generalize to more complex APIs and different API types.

5.2 Future Research

The conclusions that can be derived from our study pertain to the entire set of modifications that were applied to optimize API documentation for the test. We have observed an overall effect jointly produced by the guidelines which motivated our modifications and the specific design decisions which we used to implement the guidelines. It is not possible to derive conclusions with respect to the contribution of individual guidelines and design decisions to the overall effects. Hence, it is impossible to decide whether all or only some of the guidelines contributed to the effects, and whether

some guidelines contribute more to the effects than others. Future research should focus on individual design variables and clarify their importance.

Related to this point, future research should also examine interactions of design variables and experience more closely. Research in other areas (in particular, multimedia learning) has demonstrated repeatedly that instructional design decisions that effectively support beginners in a field are less effective – and sometimes even adversely – for learners that are more knowledgeable, a phenomenon known as expertise reversal effect [37]. Applied to API documentation, future research should examine the role of general programming experience and domain-related background knowledge more closely, and it should also address whether guidelines that effectively support developers in the initial learning process also support developers that have already reached a certain level of expertise with an API.

Our study provides only limited information with respect to the effect of guidelines for optimizing documentation on perceived usability. Future research should use more sophisticated instruments to evaluate perceived usability, e.g. by using standardized questionnaires such as the System Usability Scale (SUS) or the Computer System Usability Questionnaire (CSUQ), which both have been found to be informative for a wide range of products [38].

ACKNOWLEDGMENTS

The work reported here has been supported by the German Federal Ministry of Education and Research (BMBF), FHprofUnt, grant 13FH014PX4 to Michael Meng

REFERENCES

[1] Samuel G. McLellan, Alvin W. Roesler, Joseph T. Tempest, and Clay I. Spinuzzi. 1998. Building more usable APIs. IEEE Softw. 15, 3 (May/Jun 1998), 78-86. DOI: https://doi.org/10.1109/52.676963

[2] Steven Clarke. 2004. Measuring API usability. Dr. Dobb's Journal, 29, 6–9. Retrieved from http://www.drdobbs.com/windows/measuring-api-usability/184405654

[3] Brad A. Myers and Jeffrey Stylos. 2016. Improving API usability. Commun. ACM 59, 6 (May 2016), 62-69. DOI: https://doi.org/10.1145/2896587

[4] Gias Uddin and Martin P. Robillard. 2015. How API documentation fails. IEEE Softw. 32, 4 (July/Aug. 2015), 68-75. DOI: https://doi.org/10.1109/MS.2014.80

[5] Barthélémy Dagenais and Martin P. Robillard, 2010. Creating and Evolving Developer Documentation. Understanding the Decisions of Open Source Contributors. In *Proceedings of the 8th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE-18)*. ACM, 127-136. DOI: https://doi.org/10.1145/1882291.1882312

[6] Jeffrey Stylos and Brad A. Myers. 2006. Mica. A web-search tool for finding API components and examples. In *Proceedings of the 2006 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'06)*. IEEE, 195-202. DOI: https://doi.org/10.1109/VLHCC.2006.32

[7] Martin P. Robillard and Robert DeLine. 2011. A field study of API learning obstacles. Empir. Softw. Eng. 16, 6 (December 2011), 703-732. DOI: https://doi.org/10.1007/s10664-010-9150-8

[8] Sae Young Jeong, Yingyu Xie, Jack Beaton, Brad A. Myers, Jeffrey Stylos, Ralf Ehret, Jan Karstens, Arkin Efeoglu, and Daniela K. Busse (2009): Improving Documentation for eSOA APIs through User Studies. In: *Proceeding of the 2nd International Symposium on End-User development. (IS-EUD 2009)*. Heidelberg: Springer (Lecture notes in computer science, 5435), 86-105. DOI: https://doi.org/10.1007/978-3-642-00427-8_6

[9] Michael Meng, Stephanie M. Steinhardt, and Andreas Schubert. 2019. How Developers Use API Documentation: An Observation Study. Commun. Des. Q. Rev 7, 2 (July 2019), 40-49. DOI: https://doi.org/10.1145/3358931.3358937

[10] Chris Parnin. 2013. API documentation: Why it sucks. [blog post] Retrieved from: http://blog.ninlabs.com/2013/03/api-documentation/

[11] Lionel C. Briand. 2003. Software Documentation: How Much is Enough? In *Proceedings of the 7th European Conference On Software Maintenance And Reengineering (CSMR'03)*. IEEE, 13-15. DOI: https://doi.org/10.1109/CSMR.2003.1192406

[12] Timothy C. Lethbridge, Janice Singer, and Andrew Forward. 2003. How Software Engineers Use Documentation: The State of the Practice. IEEE Softw. 20, 6 (Nov./Dec. 2003), 35-39. DOI: https://doi.org/10.1109/MS.2003.1241364

[13] Martin P. Robillard. 2009. What Makes APIs Hard to Learn? Answers from Developers. IEEE Softw. 26, 6 (Nov./Dec. 2009), 27-34. DOI: https://doi.org/10.1109/MS.2009.193

[14] Michael Meng, Stephanie M. Steinhardt, and Andreas Schubert. 2018). Application Programming Interface Documentation. What Do Software Developers Want? J. Tech. Writ. Commun. 48, 3 (July 2018), 295-330. DOI: https://doi.org/10.1177/0047281617721853

[15] Ekwa Duala-Ekoko and Martin P. Robillard. 2012. Asking and answering questions about unfamiliar APIs. An exploratory study. In *Proceedings of the 34th International Conference on Software Engineering (ICSE)*. IEEE, 266-276. DOI: https://doi.org/10.1109/ICSE.2012.6227187

[16] Robert B. Watson, Mark Stamnes, Jacob Jeannot-Schrieder, and Jan H. Spyridakis. 2013. API Documentation and Software Community Values: A Survey of Open-Source API Documentation. In: *Proceedings of the 31st ACM International Conference on Design of Communication (SIGDOC 13)*, ACM, 165-174. DOI: https://doi.org/10.1145/2507065.2507076

[17] Sebastian Baltes, Christoph Treude, and Martin P. Robillard. 2020. Contextual Documentation Referencing on Stack Overflow. IEEE Trans. Softw. Eng. Advance online publication. DOI: https://doi.org/10.1109/TSE.2020.2981898

[18] Steven Clarke. 2007. What is an end user software engineer? In: Dagstuhl Seminar Proceedings. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. Retrieved from http://drops.dagstuhl.de/opus/volltexte/2007/1080/.

[19] Hans van der Meij, and John M. Carroll. 1995. Principles and heuristics for designing minimalist instruction. Tech. Commun. 42, 2, 243–261.

[20] Jonathan Sillito and Andrew Begel, 2013. App-directed learning: An exploratory study. In *Proceedings of 6th International Workshop on Cooperative and Human Aspects of Software Engineering*.IEEE, 81–84. DOI: https://doi.org/10.1109/CHASE.2013.6614736

[21] Andrew J. Ko and Yann Riche. 2011. The Role of Conceptual Knowledge in API Usability. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*.IEEE, 173-176. DOI: https://doi.org/10.1109/VLHCC.2011.6070395

[22] Mary Beth Rosson and John M. Carroll. 1996. The reuse of uses in Smalltalk programming. ACM Trans. Comput.-Hum. Interact. 3, 3 (September 1996), 219-253. DOI: https://doi.org/10.1145/234526.234530

[23] Miryung Kim, Lawrence Bergman, Tessa Lau, and David Notkin. 2004. An ethnographic study of copy and paste programming practices in OOPL. In *Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE '04)*.IEEE, 83-92. DOI: https://doi.org/10.1109/ISESE.2004.1334896

[24] Kirk St.Amant and Lisa Meloncon. 2016. Reflections on research: Examining practitioner perspectives on the state of research in technical communication. Tech. Commun. 63, 4, 346-364.

[25] John T. Guthrie, Stan Bennett, and Shelley Weber. 1991. Processing procedural documents. A cognitive model for following written directions. Educ. Psychol. Rev. 3, 3 (September 1991), 249-265. DOI: https://doi.org/10.1007/BF01320078

[26] Tom Johnson, n.d. What is a REST API? [blog post]. Retrieved from https://idratherbewriting.com/learnapidoc/docapis_what_is_a_rest_api.html.

[27] R Core Team. 2019. R: A language and environment for statistical computing [Computer software manual]. Vienna, Austria. Retrieved from https://www.R-project.org/

[28] Harald Baayen, Doug J. Davidson, and Douglas M. Bates. 2008. Mixed-effects modeling with crossed random effects for subjects and items. J. Mem. Lang. 59, 4 (November 2008), 390-412. DOI: https://doi.org/10.1016/j.jml.2007.12.005

[29] Douglas M. Bates, Martin Mächler, Ben Bolker, and Steve Walker. 2015. Fitting linear mixed-effects models using lme4. J. Stat. Softw. 67, 1, 1-48. DOI: https://doi.org/10.18637/jss.v067.i01

[30] Florian T. Jaeger. 2008. Categorical data analysis. Away from ANOVAs (transformation or not) and towards logit mixed models. J. Mem. Lang. 59, 4 (November 2008), 434-446. DOI: https://doi.org/10.1016/j.jml.2007.11.007

[31] Laura Winther Balling. 2008. A brief introduction to regression designs and mixed-effects modelling by a recent convert. In: Susanne Göpferich, Arnt Lykke Jakobsen and Inger M. Mees (Ed.): Looking at eyes. Eye-tracking studies of reading and translation processing. Frederiksberg: Samfundslitteratur Press (Copenhagen Studies in Language, 36), 175-192.

[32] Laura Winther Balling. 2018. No effect of writing advice on reading comprehension. J. Tech. Writ. Commun. 48, 1 (January 2018), 104-122. DOI: https://doi.org/10.1177/0047281617696983

[33] Michael Meng. 2019. Effects of visual signaling in screenshots: an eye tracking study. Tech. Commun. 66, 4, 396-411.

[34] Hannes Matuschek, Reinhold Kliegl, Shravan Vasishth, Harald Baayen, and Douglas M. Bates. 2017. Balancing Type I error and power in linear mixed models. J. Mem. Lang. 94, (June 2017), 305-315. DOI: https://doi.org/10.1016/j.jml.2017.01.001

[35] Russell Lenth. 2019. Estimated marginal means, aka least-squares means. R package version 1.3.5.

[36] Andrew J, Ko, Thomas D. LaToza, and Margaret M. Burnett. 2015. A practical guide to controlled experiments of software engineering tools with human participants. Empir. Softw. Eng. 20, 1 (February 2015), 110-141. DOI: https://doi.org/10.1007/s10664-013-9279-3

[37] John Sweller, Paul Ayres, and Slava Kalyuga. 2011. The Expertise Reversal Effect. In: John Sweller, Paul Ayres, and Slava Kalyuga (Ed.): Cognitive Load Theory. New York, Dordrecht, Heidelberg, London: Springer, 155-170. DOI: https://doi.org/10.1007/978-1-4419-8126-4_12

[38] James R. Lewis. 2019. Measuring perceived usability: SUS, UMUX, and CSUQ ratings for four everyday products. Int. J. Hum-Comput. Int. 35, 15, 1404-1419. DOI: https://doi.org/10.1080/10447318.2018.1533152