BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF FOREIGN LANGUAGES
ÚSTAV JAZYKŮ

TECHNICAL WRITING AS A SPECIFIC USE OF LANGUAGE
TECHNICKÉ PSANÍ JAKO SPECIFICKÝ ZPŮSOB POUŽITÍ JAZYKA

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR
AUTOR PRÁCE
Sergei Petrosian

SUPERVISOR
VEDOUCÍ PRÁCE
PhDr. Milan Smutný, Ph.D.

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Angličtina v elektrotechnice a informatice**
Ústav jazyků

Student: Sergei Petrosian ***ID:*** 185905
Ročník: 3 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Technické psaní jako specifický způsob použití jazyka

POKYNY PRO VYPRACOVÁNÍ:

Popište charakteristické rysy různých druhů technické dokumentace, jejich jazykové a nejazykové prostředky. Na příkladech analyzujte shody a rozdíly v prezentaci specifických technických informací v různých oblastech a pro různé účely.

DOPORUČENÁ LITERATURA:

Krhutová, M. (2009) Parameters of professional discourse: English for electrical engineering. Brno: Tribun EU.

Lengálová, A. (1999) A brief guide to technical writing. Brno: Vysoké učení technické v Brně, Fakulta managementu a ekonomiky ve Zlíně.

Urbanová, L. (2008) Stylistika anglického jazyka. Brno: Barrister and Principal, Filozofická fakulta Masarykovy univerzity.

Termín zadání: 7.2.2019 ***Termín odevzdání:*** 28.5.2019

Vedoucí práce: PhDr. Milan Smutný, Ph.D.
Konzultant:

doc. PhDr. Milena Krhutová, Ph.D.
předseda oborové rady

ABSTRAKT

Rychlý rozvoj počítačových technologii ve dvacátém století vyústil v rostoucí potřebu technické dokumentace, která by popsala používání těchto nových technologií. Tato dokumentace se píše ve speciálním stylu,, lišícím se od stylu publicistického nebo akademického. Zatímco jiné styly psané pro širokou veřejnost používají metafory, alegorie, a jiné řečnické obraty, které mají na čtenáře zapůsobit, technické psaní používá jasný přímý jazyk, který jednoznačně popisuje fakta. Tento jednoduchý jazyk umožňuje aby byl obsah čitelný a pochopitelný i pro čtenáře kteří nejsou experty v oblasti, kterou text popisuje, a pro které angličtina není rodným jazykem. Tato bakalářská práce uvádí hlavní postupy používané v technickém psaní pro dosažení jednoduchosti a čitelnosti textu, a také různé druhy technické dokumentace a jejich popis. Jako ilustrace toho, jak jsou tyto postupy používané ve skutečné práci, tato bakalářská práce uvádí analýzu třech manuálů popisujících stejnou funkci: 1. Dokumentace konkrétní funkcionality open source projektu Foreman, která inspiruje obdobnou funkci Red Hat Satellite. 2. Přepracované instrukce, vytvořené jako návrh pro publikaci v oficiální dokumentaci Red Hat Satellite, ve stavu před kontrolou. 3. Zkontrolovaná a opravená verze, publikovaná v oficiální dokumentaci pro Red Hat Satellite.

KLÍČOVÁ SLOVA

technické psaní, odborná obec, technická komunikace, technická dokumentace, angličtina jako světový jazyk.

ABSTRACT

The rapid development of computer technologies in the 21st century has resulted in the growth of needs for a technical documentation describing how to use these new inventions. Documentation written in a style of technical writing differs enormously from texts in journalism, academic writing, or marketing styles. While other genres are written for the wide audience and they tend to be written in a lush language, using metaphors, allegories, and other tropes in order to impress the reader, technical writing uses a strict clear and straightforward language to express the idea unambiguously. Technical writing uses simple language in order to make the content readable and easily understandable even for those readers who study English as their second language and may not be experts in the field the text describes. This bachelor thesis describes the main tactics used in technical writing to write simply and clearly, as well as different types of technical writing and their description. As an illustration of how these tactics are used in a real working environment, the bachelor thesis outlines an analysis of three manuals describing the same feature: 1. A manual of the feature from a documentation of a product Foreman that was taken as a basis for the Red Hat Satellite feature. 2. A reworked manual that was written as a preparation to be published to official documentation of Red Hat Satellite. 3. The reviewed and approved document that now resides in Red Hat Satellite official documentation.

KEY WORDS

technical writing, technical communication, technical documentation, English as a global language.

PETROSIAN, Sergei. *Technické psaní jako specifický způsob použití jazyka.* Brno, 2019. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/119373. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav jazyků. Vedoucí práce Milan Smutný.

Prohlášení

Prohlašuji, že svou bakalářskou práci na téma Technické psaní jako specifický způsob použití jazyka jsem vypracoval samostatně pod vedením vedoucí/ho bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne …….. ……..

(Sergei Petrosian)

PODĚKOVÁNÍ

Děkuji vedoucímu bakalářské práce PhDr. Milanu Smutnému, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc, a další cenné rady při zpracování mé bakalářské práce.

V Brně dne ……..

…….

(Sergei Petrosian)

Table of contents

1 Introduction to Technical Communication 1
2 Principles 1
2.1 Principles of Simplicity 1
Tactic 1: Adopting Reduced Dictionary 2
Tactic 2: Using Words with only One or Few Meanings 3
Tactic 3: Using the Simplest Grammar Structures 3
Tactic 4: Avoiding Verbs with Two or Three Words in Them 5
Tactic 5: Avoiding Nominalizations 5
2.2 Principles of Clarity 6
Tactic 6: Avoiding Abbreviations, Acronyms and Contracted Forms 7
Tactic 7: Avoiding Anthropomorphic Verbs 7
Tactic 8: Indicate Location of Content in a Document Clearly 9
2.3 Principles of Proper Punctuation 9
Tactic 9: Using the Oxford Comma 10
Tactic 10: Separating Long Sentences 10
Tactic 11: Using Parentheses 11
2.4 Principles of Proper Document Layout 11
Tactic 12: Focus on Findability 12
Tactic 13: Use Clear Titles 12
Tactic 14: Keeping Paragraphs Short 13
Tactic 15: Text Highlighting 14
Tactic 16: Using Lists and Tables 15
3 Types of Technical Writing 16
3.1 Reports and communications 16
3.2 Books, magazines, technical papers for the purposes of education and the sharing of knowledge and information. 17
3.3 Patents 17
3.4 Operational manuals, instructions, and procedures 17
4 Technical Texts Analysis 18
4.1 Analysis of the Introductory Section 19
4.1.1 The Upstream Product Documentation 19
4.1.2 The Draft Document 21
4.1.3 The Official Document 21
4.2 Analysis of the Installation Section 22
4.2.1 The Upstream Product Documentation 22
4.2.2 The Draft Document 23

4.2.3 The Official Document....24
4.3 Analysis of the Configuration Section....25
4.3.1 The Upstream Product Documentation....25
4.3.2 The Draft Document....27
4.3.3 The Official Document....29
4.4 Analysis of the Usage Section....33
4.4.1 The Upstream Product Documentation....33
4.4.2 The Draft Document....36
4.4.3 The Official Document....38
5 Conclusion....42
6 Bibliography....43

1 Introduction to Technical Communication

Technical writing had existed long ago before the invention of the first computers. In 15th-16th centuries, Leonardo Da Vinci chronicled his inventions and discoveries as technical documentation. Owing to this, present day scientists are able to design their inventions and make sure that they actually work. However, in the last several decades the style of technical writing has grown up rapidly with the invention of first computers that triggered massive needs for user's manuals, technical reports and descriptions.

The style of technical writing differs greatly from the style of journalism, academic writing, or marketing. Though technical writing could be written for the narrow or wide audience, only a few enjoy the process of reading it. For the most time, instructions, for example, are read to fulfil some task or achieve some purpose. That is why the main principles of technical writing require writing in a clear and simple language, so that even a person who is not an expert in the field that the text describes and whose English is far from being perfect would understand the text and get the message conveyed correctly. The tactics a technical writer uses to meet this purpose and different types of technical writing, their main features and applications are described in this bachelor thesis. In addition, the analysis of technical documents is performed to present how these tactics are used in a real working environment.

2 Principles

This chapter describes the main principles that could help a technical writer to create good content and meet the requirements of proper technical writing:

1. Principle of simplicity
2. Principle of clarity
3. Principle of proper punctuation
4. Principle of proper document layout

Each principle leads to tactics that can be used to ensure that the principle is kept.

2.1 Principles of Simplicity

Every individual word can carry several different meanings. It is only possible to determine the meaning of a word by the context. It requires the reader to make guesses and to apply personal experience to understand a text. To ensure that the reader understands your writing, it is necessary

to use the right words in your text. The main approaches to make your writing simple are described in this section.

Tactic 1: Adopting Reduced Dictionary

The basic English vocabulary, or, in other words, the vocabulary non-native speakers learn in the very first year of studying language is a good choice for documentation purposes. The main idea of technical documentation is not to write using sophisticated language that sounds great but to write clearly. For example, use *long* instead of *protracted* or *extended, teacher* instead of *facilitator* or *instructor* or *lecturer, house* instead of *residence* or *domicile, start* instead of *initiate* or *implement*. That aspect is described widely in Ogden (1937). As he saw English as a global language, he decided to implement simplicity practices into the language. According to Ogden (1937), the two most significant disadvantages of English are its complicated spelling and an extremely large vocabulary. Weiss (2005) claims, that Ogden's spelling suggestions failed and did not spread widely, but vocabulary recommendations were more successful. Ogden (1937) decided to reduce the rich English vocabulary into 850 basic words. Such a little amount can be learnt in several weeks and then used to handle a conversation and discuss personal and professional topics. Listed below are all the permitted words beginning with J and K:

- J-words — jelly, jewel, join, journey, judge, jump.
- K-words — keep, kettle, key, kick, kind, kiss, knee, knife, knot, knowledge

(Ogden 1937)

Guided by these principles of Basic English, one will not be able to use words *just, justice, July,* or sing a *Jingle bells* song, as well as use *knock, karate, kick.* According to Weiss (2005, p. 17) Ogden says that there is an equivalent word or phrase in the list of 850 words to stand for the forbidden word, for example using *dog doctor* instead of *veterinarian.*

Weiss (2005) claims that many firms develop their own restricted vocabulary, as a way to improve their documentation and make it more understandable for the international audience. They are inspired by the Ogden's work and limit the number of permitted words by 1000-3000, in addition to a number of product-specific terms and use only those words.

Pringle and O'Keefe (2009) claim that texts in technical writing style must be written at an eighth-grade level to ensure that readers will understand the content. The writer must not make assumptions about the audience of his writing to avoid being dismissive.

Krhutová (2009), on the other hand, claims that in technical communication, it is required to know who is the audience and to understand their degree of understanding in a given topic.

> Thus, producers need not and cannot rely on negotiation of meaning present in spoken utterances but they count on professional knowledge as a reliable background for the appropriate

> comprehension. As they know, they will not receive immediate feedback; writers of professional texts have to presuppose their readers´ degree of professional knowledge to eliminate misunderstanding.
> (Krhutová, 2009, p. 42)

Based on this knowledge, the writer must be as specific as it is required for the readers. Some technical texts might be written for experienced IT specialists and system administrators only. Such texts may use complicated terminology without explanations expecting readers to know them. For other texts, for example, for vacuum cleaner manuals, the audience may know nothing about technology. The writer must always carry in mind who his audience is and write in as simple language as required.

Tactic 2: Using Words with only One or Few Meanings

Words the meaning of which vary depending on the context could confuse the reader. For example, there are words that have different meanings in British and American English, such as *block* (BrE: a building; AmE: the buildings next to each other between crossing streets, or the distance from one street to the next in a city or town), *bureau* (BrE: an organization or a business that collects or provides information; AmE: a chest of drawers), *coach* (BrE: bus; AmE: someone whose job is to teach people to improve at a sport, skill, or school subject) (Cambridge Dictionary, 2017).

Also, it is recommended to avoid words which could have metaphorical meanings. For example, the word *head*, in simplified English could be defined as "the top of something". While in general English, it has several different meanings, including definitions as an adjective as a reference to the user of illegal drugs.

Tactic 3: Using the Simplest Grammar Structures

Tactic 1: Adopting Reduced Dictionary recommends using vocabulary that non-native speakers learn in the very first year of language studying in technical writing. The same approach applies to grammar. It often takes a long time to learn the more complicated forms of verbs. That is why difficult tenses and moods must be avoided in technical documentation and substituted, where possible, with simpler structures.

In the documentation, the use of simple forms of present and past tenses in the documentation. DeRespinis (2012) denotes that a writer must use past or simple tense only when using present tense does not make sense in a given context. The first sentence in the example below

is correct with present tense as well as with future tense, but present tense must be used for simplicity. However, the second sentence is incorrect when it uses present tense only:

Before:

When you open the latch, the panel will slide forward.

Cancel a broker deployment only if you are sure that the broker never responds to the deployment request.

After:

When you open the latch, the panel slides forward.

Cancel a broker deployment only if you are sure that the broker will never respond to the deployment request.

(DeRespinis, 2012, p. 35)

In technical documentation, active voice is used instead of passive, indicative mood instead of subjunctive. Considering indicative and subjunctive moods, the first one is simpler to understand. Even though it is missing from nearly all American business writing, it still appears in such expressions as: *should it prove to be the case that* (*if*), or *should you decide to* (*if you decide to*) (Weiss, 2005).

Before:

Should access to the file be denied, grant it the proper permissions.

After:

If you do not have access to the file, grant it the proper permissions.

(Weiss, 2005, p. 26)

Gerson and Gerson (2010) claim that passive voice causes the following problems for understanding:

1. Passive constructions make sentences unclear because they use pronouns instead of saying precisely who performs the action
2. Passive constructions use more words because they always require helping words.

The following examples illustrate how these two problems can be resolved. The **After** sentence explicitly outlines that the subject is *Kin Norman* and eliminates helping words *has been*.

Before:

It has been decided that Joan Smith will head our Metrology Department.

After:

Kin Norman decided that Joan Smith will head our Metrology Department.

(Gerson and Gerson, 2010, p. 57)

However, DeRespinis (2012) denotes that passive voice is acceptable sometimes and lists the following conditions that must be met to use passive voice in technical writing:

- The system performs the action.
- It is more appropriate to focus on the receiver of the action.
- You want to avoid blaming the user for an error, such as in an error message.
- The information is clearer in passive voice.
- Specific information types, such as glossary definitions, require passive voice.

(DeRespinis, 2012, p. 36)

The following examples represent situations where all conditions are met and using of passive voice is acceptable:

Examples of appropriate use of passive voice

The file is saved when you press Enter.

When you use the **recover** command, any lost data is recovered.

Error: An incorrect value was entered.

(DeRespinis, 2012, p. 36)

Tactic 4: Avoiding Verbs with Two or Three Words in Them

Using simple forms of verbs makes the documentation easier to read and to understand. Phrasal verbs carry context-dependent meaning, which makes the documentation ambiguous. That is why it is necessary to replace phrasal verbs like *carry on* and *open up* with *continue* and *expand.*

By avoiding two- and three-word verbs with prepositional tails, another additional problem is solved. For example, the structure of the **Before** sentence may confuse the reader because it overlooks the fact that the verb there is *attend (to).* While replacing the two-word structure *attend to* with a one-word verb *discuss* excludes the problem:

Before:

The report outlines three problems to which we should attend.

After:

The report outlines three problems we should discuss.

(Weiss, 2005, p. 25)

Tactic 5: Avoiding Nominalizations

According to the Cambridge Dictionary (2017), nominalization the process of making a noun from a verb or adjective. The use of such a noun phrase instead of a verb or an adjective makes the style more formal. For their ability to convert a powerful direct word like *explain* into a weak pseudotechnical *furnish an explanation for,* Weiss (2005, p. 33) calls nominalizations

"smothered verbs". This strategy is often used in school and business writing, where writers are trying hard to make ordinary comment sound more like scientific or technical reporting. Nominalization is also used to break bad news. In the following example, the **Before** sentence sounds less disappointing or incompetent than the **After** sentence**.**

> **Before:**
> We have not yet made a selection regarding a project leader.
> **After:**
> We have not selected a project leader yet.
> (Weiss, 2005, p. 25)

For non-native English speakers, nominalizations could sound confusing, especially those in the passive voice, similar to multi-word phrasal verbs mentioned previously.

Weiss (2005, p. 36) demonstrates sentences with the most common examples of using nominalizations and how they could be fixed:

> **Before**:
> The agency will conduct an investigation on the incident.
> Comparison of the actual and forecast return is performed within the software.
> **After**:
> The agency will investigate the incident.
> The software compares the actual return with the forecast.
> Weiss (2005, p. 36)

Even though these nominal phrases certainly impress readers, they must not be used in technical writing, because the portion of complexity and difficulty they add to the sentence is unnecessary in this specific style.

2.2 Principles of Clarity

The goal of technical writing is to produce a document that conveys a single meaning that will be understood by a reader easily and clearly.

According to Markel (2012), technical communication must be clear for two reasons:

- *Unclear technical communication can be dangerous.* A carelessly drafted building code, for example, could tempt contractors to use inferior materials or techniques.
- *Unclear technical communication is expensive.* The average cost of a telephone call to a customer-support center is more than $32 (About.com, 2008). Clear technical communication in the product's documentation — its instructions — can greatly reduce the number and length of such calls.

(Markel 2012, p.13)

Krhutová (2009) also claims that clear and unambiguous meaning is of vital importance in technical written communication as opposed to spoken communication, because readers never have a possibility to ask questions if something in a text is unclear to them:

> It is necessary in explication, which is a prevalent form of science and technology texts, to be clear and keep up to the logical order in describing the phenomena concerned. Compared with conversation, a written text need not be brief which depends on the phenomenon in question. Some theories might be rather sophisticated and thus deserve a long explication or description. Nevertheless, the producer has to be logical and chronological in his elaboration.
>
> (Krhutová, 2009, p. 112)

This chapter covers the main approaches to avoid confusing and misleading writing.

Tactic 6: Avoiding Abbreviations, Acronyms and Contracted Forms

Using contractions in technical documentation must be avoided, because of possible ambiguity. It is necessary to use full forms, using *can't*, *don't* is prohibited, as well as using *e.g.* or *i.e.*, it is necessary to replace them with *for example* and *that is*.

Using abbreviations is not recommended either, because they could mean different things in different context. We cannot expect the reader to know what is meant under an abbreviation. For example, there are 70 different meanings for the IRA abbreviation (AcronymFinder, 2017). We cannot expect the reader to know the specific meaning in the particular content.

Of course, there are exceptions to this rule. Gerson and Gerson (2010, p. 56) lists common acronyms such as *IBM* that is widely known as *International Business Machines, radar - radio detecting and ranging* - and *modulator-demodulator* that could not be recognized to be a *modem*.

Another exception is using contracted forms, introduced by the company. In this case, the abbreviation must be clarified at the beginning of the section where this abbreviation appears for the first time. Later in the section mentioning a key to the abbreviation is not necessary. For example, a text about a Software-as-a-Service defines what Software-as-a-Service is in the beginning and gives the acronym in the form *Software-as-a-Service (SaaS)*.

Tactic 7: Avoiding Anthropomorphic Verbs

Verbs that attribute characteristics of human beings to inanimate objects could lead to misunderstanding and difficulties in translation, especially for machine translation. DeRespinis (2012) recommends avoiding the use of verbs such as *ask, want, expect, decide, allow* with inanimate objects. Instead, focus on actions that the reader must perform to achieve his goal.

Examples (anthropomorphic verbs)

The application *asks* you for a value.

The control partition *decides* to switch roles.

If the system *sees* a new device, it changes the setting automatically.

The Merge Certificate window *tells* you about the new certificate information.

If a connector is disabled, the system *thinks* that the entire channel is disabled.

When the controller *wants* to create a resource, it constructs a list of storage locations.

Examples (appropriate verbs)

The application prompts you for a value.

The control partition switches roles.

If the system detects a new device, it changes the settings automatically.

The Merge Certificate window displays the new certificate information.

If a connector is disabled, the system operates as if the entire channel is disabled.

Before the controller creates a resource, it constructs a list of valid storage locations.

(DeRespinis, 2012, p. 9)

However, some programs perform actions and it is required to inform a user that a program *searches* for a text string or *detects* a new device. DeRespinis (2012) provides examples of how to avoid a product focus and focus on a user instead:

Examples (avoid a product focus)

The Delivery Information window allows you to specify the name of the sender.

This menu enables you to create diagrams.

The configuration expects the disk to have parity protection.

The Replicator page lets you synchronize your local database with replica databases.

If you double-click a file, the system offers to download it to your local directory.

The website permits you to save your credit card information.

Examples (prefer a user focus)

In the Delivery Information window, specify the name of the sender.

Use this menu to create diagrams.

Before you configure the disk, ensure that it has parity protection.

On the Replicator page, you can synchronize your local database with replica databases.

Double-click a file to download it to your local directory.

You can save your credit card information on the website.

(DeRespinis. 2012, p. 8)

Tactic 8: Indicate Location of Content in a Document Clearly

Documentation often refers to the information that was mentioned in the above paragraph, above chapter, or in an example that goes after. These descriptions are clear for readers who have the document opened on their PC. To see the above paragraph, they scroll up the document intuitively. However, for those readers who have printed the document, this could become a problem. Because an example, that locates in the above section, could be situated on the previous page for them. To avoid ambiguity in such cases, it is recommended to use clear marks of location, for example, *previous* instead of *above, following* instead of *below.*

To make technical documentation usable for people who are blind or have low vision, it is necessary to refer to other parts of a document not only by a location, but also by text information. DeRespinis (2012) recommends using prefixes *upper* and *lower* instead of *top* and *bottom* and provides the following examples to conform to accessibility best practices:

> **Before:**
> **T**he list of tables is displayed in the upper-right corner of the window.
> **After:**
> The tables are displayed in the Table List pane, which is in the upper-right corner of the window.
> (DeRespinis, 2012, p. 206)

2.3 Principles of Proper Punctuation

Correct punctuation helps to organize the correlation between words in a sentence. When a sentence lacks a single punctuation mark, it is likely to be misunderstood, or at least it will make the reader read the sentence several times to understand it correctly. The following example illustrates a situation when a missing comma creates problems in understanding the meaning:

> By the time I finished my supper had already got cold.
> *(*Weiss, 2005, p. 71)

One missing comma makes the sentence unclear from the first reading, which forces the reader to read it through once again. In such a simple case with one missing comma, the sentence stays more or less clear for the experienced reader. However, the inexperienced language student may misunderstand the sentence. The approach to using as few commas as possible prevails in journalism, while in technical environment clarity plays a crucial role, which makes correct punctuation extremely important. This section describes using punctuation marks in a more detailed way.

Tactic 9: Using the Oxford Comma

The *Oxford comma*, the *Harvard comma* or the *serial comma*, is a comma used before the words *and*, *or*, and *nor* at the end of a listing. It is considered optional in many styles: in newspapers, magazines, school articles, but it is highly required in technical writing. At first glance, Oxford comma could seem to be unnecessary, however, lack of such a comma in the following example might lead to misunderstandings:

> The location study covered labor, tax, freight, and communications costs, all in terms of 1972 prices.
> (Garner, 2016, p. 748)

In this example, the missing Oxford comma after *freight* could make a reader think, that *freight* and *communication costs* belong to one category, though it is not meant to be. Therefore, to ensure that the text will not cause any misunderstandings, using the Oxford comma is mandatory in technical writing.

Tactic 10: Separating Long Sentences

Besides using the simplest grammar structures in order to meet the requirement of simplicity, there is another way to make a text easy to read — write in short sentences and separate long complicated clauses by full stops. It is easier to read two independent sentences with full stops than a single sentence with two clauses linked by a comma. DeRespinis (2012) recommends keeping sentences 25 or fewer words. As can be seen from the following example presented by Weiss (2005), the first sentence separating long sentences makes the text easier to comprehend:

> **Before:**
> Unless these terms are defined in a glossary, international documents should be free of "buzzwords," overworked words, or any words uniquely associated with a particular management theory or fashionable management consultant.
> **After:**
> In international documents, do not use overworked or fashionable words. Avoid words that are part of a particular management theory; also avoid words made popular by a particular management consultant. If you must use these words, however, define them in a glossary.
> (Weiss, 2005, p. 66)

Tactic 11: Using Parentheses

Text enclosed in parentheses is considered to be less important. Therefore, it is recommended to use parentheses only with abbreviations or symbols. For example, using parentheses to introduce abbreviation in the following sentence:

> Brno University of Technology (BUT) is one of the leading teaching and research universities in central Europe. Its programmes cover not only engineering but also fields of natural science, arts, IT and economics, with a strong interdisciplinary emphasis and close connection with the industrial sphere.
>
> ("Brno University of Technology - Study in the Czech Republic," n.d.)

However, in running text, a reader can intuitively pay less attention to the text in parentheses. Therefore, it is recommended to either remove the parentheses, if the text within is important. Alternatively, remove a text in parentheses completely, if it does not carry any meaning. For example, the following sentence contains a text in parenthesis that can be missed or considered as less important. To avoid this, the text in parenthesis can be converted to a new sentence or removed if it is not important in the given context.

> **Before:**
>
> Red Hat Satellite 6 also contains some predefined kickstart templates (and the ability to create your own), which are used to provision new systems and customize the installation.
>
> **After:**
>
> Red Hat Satellite 6 also contains some predefined kickstart templates, which are used to provision new systems and customize the installation. Additionally, you can create and use your own kickstart templates.
>
> **Or:**
>
> Red Hat Satellite 6 also contains some predefined kickstart templates, which are used to provision new systems and customize the installation.
>
> (Red Hat Customer Portal)

Kirkman (2006, p.87) outlines that parenthesis can also be used to refer to tables and figures:

> ... our plot of energy migration (Figure 2) shows the ...
>
> (Kirkman, 2006, p.87)

2.4 Principles of Proper Document Layout

In order to write simply and clearly, different methods and instruments are used in technical writing. A technical writer creates a readable, clear easy to search through documentation. In order to achieve these points, the document should be formatted correctly. Modern text editors offer a

wide range of useful text formatting. Tactics on how to make a text readable, where to include tables, lists, diagrams and figures into your writing are described in this section.

Tactic 12: Focus on Findability

The content of technical documents must be structured logically. so that a reader will intuitively find the required information. Tebeaux, E., and Dragga denote several aspects that are of great importance for document design:

> Effective document design is built on principles of visual perception—on how human beings perceive and interpret visual information. For example,
>
> • Contrast: Different items must be visibly different, especially more important and less important items.
>
> • Alignment: Related items must be aligned with each other, and every item must be aligned with some other item (or risk looking misaligned).
>
> • Proximity: Related items must be positioned close together.
>
> • Size: Greater size implies greater importance.
>
> • Repetition: Repetition of design creates unity and builds familiarity.
>
> (Tebeaux, E., & Dragga, 2015, p. 86)

As per the Proximity principle, related topics must be placed next to each other. The concluding steps, such as retiring a product, cleaning up after retirement must be placed closer to an end of a section. The verification steps must go after the procedure, the success of which is going to be verified.

The repetition principle states that the structure must remain the same throughout the documentation. Therefore, after reading and working with one section or chapter, readers will already know where to find the information they need in another section.

Tactic 13: Use Clear Titles

To find the required information, the first thing readers usually do, is searching through titles in the table of contents in the beginning of a document and determining what title is related to their need. If readers find the required information by a title in the table of contents, it saves them a large amount of time, in comparison to scanning through the whole document. Consequently, it is essential to pay special attention to titles.

Make titles clear and accurate. A good title should be descriptive, but not very long. It should inform a reader, but be short enough to allow a reader to read it quickly. Every word should relate to the content and be meaningful.

A good title must be consistent. If one title uses a gerund form of a verb instead of an infinitive, for example, the gerund form "Making" in the title "Making a text bold in Microsoft Office Word", then all the other titles in the document must be consistent. It makes the titles easier to scan through while reading quickly.

The table of contents of the *Managing Errata* section in the Red Hat Satellite Content Management guide (n.d.) is an example of a correct titles structure:

9. Managing Errata

9.1. Managing Errata with Content Views

9.2. Inspecting Available Errata

9.3. Applying Errata to Individual Systems

9.4. Applying Errata to Multiple Systems

9.5. Subscribing to Errata Notifications

9.6. Chapter Summary

("Chapter 9. Managing Errata - Red Hat Customer Portal," n.d.)

These titles are descriptive, they accurately describe what procedures are explained in each section. The titles are consistent in using verbs in gerund form.

Tactic 14: Keeping Paragraphs Short

Long paragraphs discourage readers from reading them. In technical writing, even if the long paragraph is on the same topic, it is recommended to separate it into smaller ones. Short paragraphs are easier to read, to work with, and to search through. Weiss (2005) claims that even if a long paragraph is logically cohesive, its length may discourage the reader, therefore it is beneficial to split it into two:

Before:

Remember that nearly all readers subvocalize, saying words mentally to themselves as they read silently. So, words that are hard to pronounce will slow the reader. This advice is particularly germane in *naming* products, systems, or companies. Nearly every E2 and E3 has trouble with the *th* sound (especially unvoiced) and many Asian languages struggle with *l* and *r*. When General Instrument Corporation of Horsham, Pennsylvania, changed their image in 1996, they also adopted the more high-tech sounding name of NextLevel. In 1998, they will restore the original name, mainly because most of their Asian customers for cable-TV converter boxes have trouble with say ing NextLevel. (Similarly, the spokesclown for McDonalds restaurants in Japan is called *Donald McDonald* not *Ronald*.)

After:

Remember that nearly all readers subvocalize, saying words mentally to themselves as they read silently. So, words that are hard to pronounce will slow the reader. This advice is particularly

germane in *naming* products, systems, or companies. Nearly every E2 and E3 has trouble with the *th* sound (especially unvoiced) and many Asian languages struggle with /and *r*.

When General Instrument Corporation of Horsham, Pennsylvania, changed their image in 1996, they also adopted the more high-tech sounding name of NextLevel. In 1998, they will restore the original name, mainly because most of their Asian customers for cable-TV converter boxes have trouble with saying NextLevel. (Similarly, the spokesclown for McDonalds restaurants in Japan is called *Donald McDonald* not *Ronald*.)

(Weiss, 2005, p. 85)

Tactic 15: Text Highlighting

In comparison to times of typewriters, when the quotation marks where the only possibility to highlight words in a sentence, nowadays modern software provides more fancy ways to accomplish this task. Different companies use different styles, but it is necessary to be consistent throughout the documentation.

Kirkman (2006) outlines two jobs that punctuation marks do in our writing:

- *grammatical*: they show where the boundaries are meant to be between segments of larger statements, and how segments of text are meant to relate to one another;
- *rhetorical*: they show the emphasis or tone we want to give to a word or word-group.

(Kirkman, 2006, p. 5)

Text highlighting matches the rhetorical job done by punctuation marks. He gives 3 example of how it is possible to signal the emphasize or a tone to a word or word-group:

1. This is known as 'exact' replacement of ...
2. ... and the line must be active at this moment ...
3. Do not touch the connectors. Frequently, the lines remain charged AFTER the power has been switched off!

(Kirkman, 2006, p. 6)

DeRespinis (2012) lists more possibilities for text highlighting:

- Bold – for column heading in the table, glossary terms, interface controls.
- Bold Monospace – for command names, keywords, parameter names. process names.
- Monospace – for directory and file names, code examples.
- Italics – for citations, letters as letters, variables.
- Double quotation marks – for hardware label names.
- Angle brackets – tags in markup languages and XML.

(DeRespinis, 2012, pp. 106 - 110)

One method is the emphasizing of key phrases with boldface. Boldface text grabs the attention of a reader. That way it is easier for the reader to notice the term he is interested in while searching through the text and read the needed paragraphs selectively.

Another use of emphasis is highlighting examples with italics. Italics are commonly used to highlight examples and values that a user must replace. DeRespinis (2012, p. 109) provides a very illustrative example of using several highlighting methods:

> Use the **OPEN** command with the ***-f*** parameter. For example:
>
> OPEN -f *file_name*
>
> where *file_name* is the name of the file you want to open.
>
> (DeRespinis, 2012, p. 109)

Tactic 16: Using Lists and Tables

Pringle and O'Keefe (2009) suggest using numbered lists to denote steps that must be performed in a particular order. Additionally, use bulleted lists for items that do not have a particular order. They provide an example of how a monster chunk of text could converted to a bulleted list that communicates the same information in a much clearer way.

An instruction that contains items or steps is far more understandable in numbered or bulleted list form:

> **Before:**
>
> To make your cake and the icing, use the oven, peeler, food processor (or cheese grater), measuring cups and spoons, wooden spoons, egg whisk, two large mixing bowls, 9 by 13 pan, wire rack, oven mitts, mixer, and bowl scrapers.
>
> **After:**
>
> Use the following kitchen appliances and hardware while baking your cake and making the icing:
>
> - Oven
> - Peeler
> - Food processor (or cheese grater)
> - Measuring cups and spoons
> - Wooden spoons for stirring
> - Egg whisk
> - Two large mixing bowls
> - 9 by 13 pan
> - Wire rack
> - Oven mitts

- Mixer
- Bowl scrapers

(Pringle and O'Keefe, 2009, pp. 117 - 118)

Difficult paragraphs that describe decision logic: options, alternatives and multiple paths must be converted into tables. DeRespinis (2012) claims that it is necessary to place a table as close as possible to the content it references to.

3 Types of Technical Writing

According to Kenneth G. Budinski (2005), the four main groups of documents in a style of technical writing are the following:

- Reports and communications in technical professions, and day-to-day business
- Books, magazines, technical papers for the purposes of education and the sharing of knowledge and information.
- Patents
- Operational manuals, instructions, and procedures

3.1 Reports and communications

In many technical professions preparation of various reports is required. Very few companies pay the salary without reports on what had been worked on or implemented, and what the results are. A report usually includes a written document and engineering drawings related.

According to Kenneth G. Budinski (2001), in day-to-day business writing letters, emails, memos surveys are common. Making a mistake in an email violates the status of a sender. Even considering that the sender spent less time on writing and reviewing the email, a mistake in it creates a negative opinion about him. In the business world, very little time is allocated for the internal emails. No one has time to read long letters written in a sophisticated lush language. On the contrary, a short simple clear readable format of technical writing is required. These formats are mostly intended for the narrow audience and they require a strict and clear language.

Gerson and Gerson (2010) identify two types of reports: s*hort, informal reports,* and *long, formal reports.* In addition, Gerson and Gerson outline the following needs that reports satisfy

- Supply a record of work accomplished
- Record and clarify complex information for future reference
- Present information to a large number of people
- Record problems encountered
- Document schedules, timetables, and milestones

- Recommend future action
- Document current status
- Records procedure

(Gerson & Gerson, 2010, p. 456)

3.2 Books, magazines, technical papers for the purposes of education and the sharing of knowledge and information.

Another category represents documents intended for educational purposes, in the form of books, magazines, and papers. These works are usually produced by experts in the fields. Definitely, writing a book requires much more discipline than writing a report, because books are written for a wide audience, but the clarity of a presentation is still required.

3.3 Patents

The third category of technical writing is Patents. These are written usually by lawyers, these pieces should be in a highly organized manner. Patents have a very restricted format, which is prohibited to break. The patent should clearly describe its subject and include precise figures and descriptions.

3.4 Operational manuals, instructions, and procedures

Finally, operational manuals, instructions and procedures have legal implications and are usually written by trained technical writers. They include all the basic principles, which are described in the previous chapter. It should be concise, clear, simple, correct, readable, easy to understand for non-native English speakers, and easy to translate to other languages. To meet these requirements, a professional technical writer should be experienced in the field of the product he writes about and in stylistics. Therefore, a technical writer becomes more and more valuable with every year of work. Readers of this documentation are not willing to study the subject to understand how it works. They are willing to fulfil the task, e.g. to fix some error or to configure some service.

Gerson and Gerson (2010) emphasized two criteria for writing instructions. The first one is audience recognition. This means that a writer must no assume that his readers have high-tech knowledge. Instead of assuming, it is necessary to spell information out clearly and thoroughly. The second criteria is ethical instructions. This means that the potential harm must be thoroughly identified in an instruction.

4 Technical Texts Analysis

This chapter outlines the process of creating a piece of technical document. It describes what is usually done to gather the required information, and how this information is processed. To achieve this, the analysis of 3 different documents that describe the same feature is performed.

The analysis is performed for the document in three different phases:

1. The document in the upstream documentation of a Foreman product that was taken as a basis for the Red Hat Satellite feature - Foreman Templates 5.0 Manual.
2. A reworked manual that was written as a preparation to be published to the official documentation of Red Hat Satellite. This document was created in the Google Docs tool and sent to a professional in the field of this program to check the technical correctness of the manual, as well as other writers to verify the style.
3. The reviewed and approved document that is now a part of the Red Hat Satellite official documentation - Appendix F. Synchronizing Templates with Git on Red Hat Customer Portal.

One of the technical writer's main tasks is to create a precise and accurate text. Initially, it must be correct from the technical view. Then, after the text is technically correct, the style can be improved, to make the text clear and simple. In this case, the goal is to describe how to use the TemplateSync plug-in with Red Hat Satellite Server. According to the official Red Hat Satellite documentation: "Red Hat Satellite is a system management solution that enables you to deploy, configure, and maintain your systems across physical, virtual, and cloud environments. Satellite provides provisioning, remote management and monitoring of multiple Red Hat Enterprise Linux deployments with a single, centralized tool" (Chapter 1. Introduction to Red Hat Satellite 6 - Red Hat Customer Portal, n.d.). The TemplateSync plug-in enables a user to synchronize provisioning templates between Satellite Server and a remote Git repository. Provisioning templates – are text files that define provisioning settings for Satellite hosts. For example, after applying changes to a provisioning template on one machine, a user can synchronize those changes to a remote Git repository. Then, from the Git repository, pull the new changes to the template file on a remote set of machines. To create accurate documentation, firstly, it is required to understand how the function works and to ensure that the function works correctly. Therefore, it is necessary to test it, to understand what information is essential for a user to know and what can be omitted. In addition, the actions a user must perform to accomplish a task, the prerequisites, the settings that had to be configured before starting the procedure. As a reference, the documentation for the upstream product "Foreman Templates" was used (Foreman Templates 5.0 Manual, n.d.). Engineers who developed the product most commonly write the documentation of an upstream product. Usually,

the documentation written by engineers does not meet the current technical writing standards, as they tend to describe all features of their product, and do not focus on a specific task.

4.1 Analysis of the Introductory Section

The introductory section must mention what the document is about and what goals will a reader be able to fulfil after reading it. It can also include some notes and warnings a reader needs to know before using a feature.

4.1.1 The Upstream Product Documentation

1. Foreman Templates 5.0 Manual

> This plugin enables synchronization of provisioning templates, partition tables and job templates from external git repository and/or file system. It can be used both with the community templates repository or with any other source of templates that has the same directory structure. Please note that Foreman core provides tracking of changes, browsing the history and reverting natively, but if you need more advanced tools such as branching, code reviews etc. you might be interested in this plugin.
>
> You can use the native git support for simple synchronization with remote git repositories, which is usually useful for syncing new template versions from community repository or for exporting local changes to contribute back. Or if you have more complicated workflow, you can use this plugin to sync the directory when required. This way, you can use any VCS (including git) to control the versioning of that directory and use this plugin to sync whenever the directory is at desired version.
>
> (Foreman Templates 5.0 Manual, n.d.)

In this case, the introduction consists of two long paragraphs. They are difficult to read through. They include several complicated constructions, such as a conditional sentence: "Or if you have more complicated workflow, you can use this plugin to sync the directory when required." That violate the Tactic 3: Using the Simplest Grammar Structures. The problem with this, is that it carries information, which can be useful for the specific user scenario where a workflow is complicated, but an ordinary user does not need to read this to perform his tasks. This information is redundant for a user, but still, he will need to read this through. This section lists several possibilities that this feature enables a user to do:

- This plugin enables synchronization of provisioning templates, partition tables and job templates from external git repository and/or file system.

- It can be used both with the community templates repository or with any other source of templates that has the same directory structure.
- You can use the native git support for simple synchronization with remote git repositories.
- You can use this plugin to sync the directory when required.
- This way, you can use any VCS (including git) to control the versioning of that directory and use this plugin to sync whenever the directory is at desired version.
- if you need more advanced tools such as branching, code reviews etc. you might be interested in this plugin.

These functions are hidden in the text, and, therefore, difficult to find and to scan through.

This section is not consistent. The text uses different structures for describing the same thing - what the plug-in enables a user to do:

- the active structure "you can use ..." where the subject is a reader,
- the active structure "This plug-in enables …", where the subject is the feature itself,
- the passive structure "it can be used ... ",
- the conditional clause with might: "you might be interested in this plugin."

This text uses the word plugin written as one word. DeRespinis (2012) suggests writing this word hyphenated - plug-in. However, the hyphens often get dropped when a word becomes more commonly used. On the web, the form of *plug-in* written as one word is met more often than the appropriate one.

The articles are dropped in several places; for example, the definite article must be used before the collocation *external git repository* in the following sentence: *"This plugin enables synchronization of provisioning templates, partition tables and job templates from external git repository and/or file system."*

This introduction uses the shortened version of the verb to synchronize - "sync" twice. This violates Tactic 6: Avoiding Abbreviations, Acronyms, Contractions.

A lowercase letter is used with the name of the Git product four times. The Git official documentation (Git) capitalizes its product name.

The abbreviation VCS is used without providing its meaning - version control system. This can be explained by the fact that this text is intended for the IT experts and expects its reader to know this abbreviation. Still, this does not correspond with the current technical writing style. This violates Tactic 6: Avoiding Abbreviations, Acronyms, Contractions.

In this section, grammar, and spelling mistakes can be explained by the fact that the upstream documentation is written by developers, who are often not English native speakers and not language experts. They do not aim to create perfectly stylized documentation; they are describing how their product on works. They often reflect changes made in the source code by adding bits in different places of documentation. This upstream product is free, so readers do not expect high standards from its documentation.

4.1.2 The Draft Document

> Synchronizing Templates with Git
>
> Red Hat Satellite 6 allows synchronization of Job Templates, Provisioning Templates, and Partition Table Templates between Satellite Server and a Git repository. This section details the workflow for synchronizing new template versions and exporting local changes to contribute back.
>
> (Synchronizing Templates with Git. Draft document. Author: S. Petrosian, 2018)

The draft introduction consists of one paragraph. The first sentences describes what this feature does. The second sentence describes what information can be found in this section.

This introduction is much shorter, only the information that is necessary for the reader to know is listed.

4.1.3 The Official Document

> APPENDIX F. SYNCHRONIZING TEMPLATES WITH GIT
>
> Red Hat Satellite 6 enables synchronization of Job Templates, Provisioning Templates, and Partition Table Templates between Satellite Server and a Git repository or a local directory.
>
> **NOTE**
>
> Synchronizing templates between Satellite Server and a Git repository or a local directory is a Technology Preview feature. Technology Preview features are not fully supported under Red Hat Subscription Service Level Agreements (SLAs), may not be functionally complete, and are not intended for production use. However, these features provide early access to upcoming product innovations, enabling customers to test functionality and provide feedback during the development process.
>
> This section details the workflow for:
>
> - installing and configuring the TemplateSync plug-in
> - performing exporting and importing tasks
>
> (“Appendix F. Synchronizing Templates with Git - Red Hat Customer Portal,” n.d.)

The introduction in the official documents is an improved version of the draft introduction. The first sentence describes what this feature is about, the same as in the draft document. After the first sentence, there is a note informing a reader that this section is about a "Technology Preview" feature, informing that this feature is not fully functional and is provided to test functionality and gather feedback.

This note is added to all documents written about technology preview features. As it was added after the full documentation of this feature was created and published, this note is missing from the draft introduction. The last sentence of the draft document was separated using bullet points to outline what specific procedures are described in the section. This makes the text easier to read and to comprehend as per Tactic 16: Using Lists and Tables.

The official document uses the word "enables" instead of "allows", which was used in the draft document. It is not correct to use the word "allow" according to the Tactic 7: Avoiding Anthropomorphic Verbs.

4.2 Analysis of the Installation Section

The installation section describes what actions must be performed before using the feature. If the feature is pre-installed in the product and no further actions are required to enable it, this section can be omitted.

4.2.1 The Upstream Product Documentation

> 2. Installation
>
> You can install the plugin using the foreman-installer (recommended way). To do that, run the installer with following argument
>
> ```
> foreman-installer --enable-foreman-plugin-templates
> ```
>
> If you installed the plugin using package directly, you should run the migrations and seed (just to be sure) and restart the Foreman. If installed foreman from git, you can simply add foreman-templates gem as one of the dependencies. Don't forget to run migrations and seed afterwards and obviously restart the Foreman.
> ("Foreman Templates 5.0 Manual," n.d.)

The installation section of the upstream product documentation outlines three different methods of installing the plug-in:

- Using the foreman-installer
- Installing using the package directly

- Installing from gems

Although several possible methods are given, the reasons for why each method must be used, for example, their advantages and disadvantages are not described.

The description of methods is not structured and given in a plain text. Only the first method lists a command that must be entered, This makes the text difficult to comprehend.

The text uses vague phrases that do not carry any special meaning and can be easily omitted, such as *don't forget to, just to be sure, obviously.*

The sentence "If installed foreman from git, you can simply add foreman-templates gem as one of the dependencies." is not grammatically correct, it omits the pronoun *you* in the beginning.

The text is inconsistent in using the product name Foreman. It is spelt with the uppercase F twice, and with the lowercase once, which is inconsistent.

The last sentence includes a complicated structure, it lists several actions inline. This is grammatically incorrect and could lead to ambiguity. This can be fixed by using the Oxford comma as per Tactic 9: Using the Oxford Comma. In this case, the corrected version of the sentence would use 3 commas: "After adding the gem, run migrations, seed, and restart the Foreman."
Furthermore, the word *seed,* in this case, is an IT jargon and has multiple meanings. Therefore, a reader can simply misinterpret it. This violates Tactic 1: Adopting Reduced Dictionary and Tactic 2: Using Words with only One or Few Meanings.

4.2.2 The Draft Document

Enabling the plug-in

The plug-in providing this feature is already included into Red Hat Satellite. Follow these steps to enable it in your Satellite Server:

1. Install the plug-in on your Satellite Server using the following command:
   # satellite-installer --enable-foreman-plugin-templates
2. Restart katello services to enable the plub-in:
   # katello-service restart

If the plug-in is installed correctly, you will see the *TemplateSync* menu in *Administer* → *Settings*.

(Synchronizing Templates with Git. Draft document. Author: S. Petrosian, 2018)

The draft installation procedure is called "Enabling the plug-in". The term *enabling* was chosen because the plug-in package is already pre-installed with Satellite. However, the first step

uses the term *install*, which makes this section not consistent and ambiguous, the terms in it do not correspond.

This section uses the following passive structures, which violates the Tactic 3: Using the Simplest Grammar Structures:

- *is already included*
- *If the plug-in is installed correctly*

Though the plug-in name **TemplateSync** uses the contracted form of the word synchronization, this cannot be changed in the documentation, because buttons in the user interface have these labels on them. The same text as on buttons labels is used to avoid ambiguity and ensure that a user finds the necessary button easily.

4.2.3 The Official Document

F.1. Enabling the TemplateSync plug-in

1. To enable the plug-in on your Satellite Server:
   # satellite-installer --enable-foreman-plugin-templates
2. To verify that the plug-in is installed correctly, ensure **Administer** > **Settings** includes the **TemplateSync** menu.

("Appendix F. Synchronizing Templates with Git - Red Hat Customer Portal," n.d.)

After engineers reviewed the draft document, they told that it is not required to restart Katello services to enable the plug-in. Therefore, this step was removed from the procedure. The introduction to the procedure that was included in the draft document was removed completely, it was decided that the procedure's title is self-explanatory itself and no introduction is needed.

The problem with inconsistency in using terms *install* and *enable* remained. The first step *enables* the plug-in; the second step verifies if the plug-in is *installed*.

The problem with passive structures also remained, while the first occurrence of the passive structure was removed, the second one "To verify that the plug-in is installed correctly" remains. This happened because in the time the section was written passive structures were not paid much attention in the product documentation.

The text uses boldface to highlight the labels **Administer, Settings, TemplateSync**. In addition, the highlighting is used to distinguish the command that must be entered into the command line interface. This highlighting is consistent throughout the Satellite documentation. As the reader is expected to know that the black code blocks with white typeface inside starting

with the hash sign implies that this text is a command and it must be entered to the command line interface, this information is omitted.

4.3 Analysis of the Configuration Section

The configuration section outlines the process of configuring the plug-in. It describes how to configure the plug-in, and what are the available options for configuration.

4.3.1 The Upstream Product Documentation

3. Configuration

After the installation there are new settings available. These are the default setting for both importing and exporting tasks. They can be overridden on each synchronization run and serve only as default so users don't have to specify the same value on every run. You will find them under Administer -> Settings -> TemplateSync. Only administrator can modify their values. Following table explains the attribute behavior. Note that some are only used on import or export actions.

Setting name	**Meaning on importing**	**Meaning on exporting**
associate	associate OS, Organization, Location based on metadata	N/A
branch	git branch to read or write from/to (only for git-based repositories), default value is based on current Foreman version - e.g. develop or 1.15-stable	
dirname	subdirectory under $repo to read from	subdirectory under $repo to export to
filter	import only templates with name matching this regular expression, after $prefix was applied	export only templates with name matching this regular expression

metadata_export_mode	N/A	one of refresh/remove/keep, refresh generates new metadata on export based on current associations and attributes, remove strips all metadata from template, keep keeps the same metadata that are part of template code
negate	negates the $filter condition	
prefix	adds specified string to beginning of the template on import, but only if the template name does not start with the prefix already	N/A
repo	target path, specifies also the protocol, e.g. /tmp/dir or git://example.com, https://example.com, ssh://example.com	
verbose	adds extra verbose messages to the action output	

Settings above are preconfigured for importing from upstream community-templates repository.

("Foreman Templates 5.0 Manual," n.d.)

The configuration section in the upstream product documentation consists of an introductory paragraph with details related to configuring the plug-in, a table of configuration settings available with their definition, and a sentence with some additional information in the end. The last piece of text is short and is not located with all other text, and, therefore, could be lost by a reader.

The introductory paragraph is badly structured, all different information is not separated into units.

The last sentence in this section uses the word "above" to refer to the previous table. That violates Tactic 8: Indicate Location of Content in a Document Clearly.

There are several typos and grammar mistakes in this text. The word *setting* must be in plural in this sentence: "These are the default setting for both importing and exporting tasks."

There is an indefinite article missing before the word *administrator in* the following sentence: "Only administrator can modify their values".

The word *setting* is omitted after the word *some* in the following sentence:"Note that some are only used on import or export actions."

The labels names use the same typeface and font as the casual text for Administer, Settings, TemplateSync. This violates Tactic 15: Text Highlighting. The reader would need to read through the whole paragraph to see where these configurations are located. In addition, the highlighting is dropped in the table with replaceable values and directories names, such as "$repo", "/tmp/dir/".

The table uses the contraction "e.g." to outline examples. This violates Tactic 6: Avoiding Abbreviations, Acronyms and Contracted Forms. In addition, the table uses contractions in the first column, e.g. *dirname* instead of Directory Name, and *repo* instead of repository. Here, this is correct because the first column contains labels from the web user interface and cannot be changed.

This section uses the passive constructions, e.g. "some are only used", which violates the Tactic 3: Using the Simplest Grammar Structures.

This section uses a lowercase letter with the name of the Git product again. The Git official documentation (Git) capitalizes its product name.

The text in cells in the table is not written in full sentences. Text in the table includes several grammar mistakes, for example missing articles. There must be the definite article after the word *beginning* in the following sentence: "adds specified string to beginning of the template on import, but only if the template name does not start with the prefix already." It makes text difficult to read and to process.

The following sentence is long; it can be split to two separate sentences in order to meet the requirements outlined in Tactic 10: Separating Long Sentences. Furthermore, it uses parentheses, which violates Tactic 11: Using Parentheses. The text in the parentheses is important, so to make the sentence correct, it is required to include the text in the parentheses into the sentence inline:

> When using git-based repository for import or export, this setting specifies git branch to read from or write to. The default value is based on current Foreman version - e.g. develop or 1.15-stable

4.3.2 The Draft Document

> Configuring the plug-in
>
> Navigate to *Administer* → *Settings* → *TemplateSync* to configure settings of the plug-in. The following table explains the attributes behavior. Note that some attributes are only used on import or export actions.

Setting name	Meaning on importing.	Meaning on exporting.
Associate	Associates templates with OS, Organization, and Location based on metadata.	N/A
Branch	Specifies the default branch in Git repository to read from.	Specifies the default branch in Git repository to write to.
Dirname	Specifies the subdirectory under the repository to read from.	Specifies the subdirectory under the repository to write to.
Filter	Imports only templates, with names that match this regular expression.	Export only templates with names that match this regular expression.
Force import	If selected, imported templates will overwrite locked templates with the same name.	N/A
Metadata export mode	N/A	Defines how metadata is handled when exporting: *Refresh* generates new metadata based on current assignments and attributes. *Keep* retains the existing metadata. *Remove* exports template without metadata, useful if you want to add metadata manually.
Negate	If selected, negates the filter attribute.	
Prefix	Adds specified string to the beginning of the template, if the template name does not start with the prefix already.	N/A
Repo	Defines the path to the repository to sync from.	Defines the path to a repository to export to.
Verbosity	Defines extra verbose messages to the action output for importing templates through rake.	N/A

(Synchronizing Templates with Git. Draft document. Author: S. Petrosian, 2018)

The draft configuration section consists of an introductory part and a table. There is no additional text below the table; all the necessary information is located above the table. The introductory part is shortened to contain only the information that is required for a user.

The introductory part contains a sentence that uses an anthropomorphic verb in the collocation *the table explains.* That violates Tactic 7: Avoiding Anthropomorphic Verbs.

The text in the table cells is changed to use full sentences, every sentence starts with capital letters and ends with dots.

This draft document kept the passive constructions, e.g. “attributes are only used”, which violates the Tactic 3: Using the Simplest Grammar Structures.

The draft section fixed the issue with the incorrect capitalization of the Git product, here it uses a capital letter with Git: “Git repository”.

4.3.3 The Official Document

F.2. Configuring the TemplateSync plug-in

Navigate to **Administer** > **Settings** > **TemplateSync** to configure the plug-in. The following table explains the attributes behavior. Note that some attributes are only used on importing or exporting tasks.

Table F.1. Synchronizing Templates Plug-in configuration

Parameter	API parameter name	Meaning on importing	Meaning on exporting
Associate	associate Accepted values: always, new, never	Associates templates with OS, Organization, and Location based on metadata.	N/A

Branch	branch	Specifies the default branch in Git repository to read from.	Specifies the default branch in Git repository to write to.
Dirname	dirname	Specifies the subdirectory under the repository to read from.	Specifies the subdirectory under the repository to write to.
Filter	filter	Imports only templates with names that match this regular expression.	Exports only templates with names that match this regular expression.
Force import	force	Imported templates overwrite locked templates with the same name.	N/A

Metadata export mode	`metadata_export_mode` Accepted values: `refresh`, `keep`, `remove`	N/A	Defines how metadata is handled when exporting: • **Refresh** — remove existing metadata from the template content and generate new metadata based on current assignments and attributes. • **Keep** — retain the existing metadata. • **Remove** — export template without metadata. Useful if you want to add metadata manually.
Negate	`negate`	Imports templates ignoring the filter attribute.	Exports templates ignoring the filter attribute.

	Accepted values: `true`, `false`		
Prefix	`prefix`	Adds specified string to the beginning of the template if the template name does not start with the prefix already.	N/A
Repo	`repo`	Defines the path to the repository to synchronize from.	Defines the path to a repository to export to.
Verbosity	`verbose` Accepted values: `true`, `false`	Enables writing verbose messages to the logs for this action.	N/A

("Appendix F. Synchronizing Templates with Git - Red Hat Customer Portal," n.d.)

The introductory part in the official documentation is improved for the sake of minimalism. The redundant noun *settings* is removed as follows:

Draft:

to configure settings of the plug-in.

Official:

to configure the plug-in.

The table here contains one more column. API parameter name resided in the table makes it easier to construct API calls for users.

4.4 Analysis of the Usage Section

The usage section for the TemplateSync plug-in outlines how to import and export templates. Users would come here after installing and configuring the plug-in. A reader would expect clear and simple procedures that describe the actions required to export and import templates.

4.4.1 The Upstream Product Documentation

4. Usage (features description)

Both importing and exporting is available as rake tasks. Rake tasks are triggered from terminal. If you installed the plugin using foreman-installer, or simply it's package based installation), you should be able to run rake using foreman-rake command. Source-based installs of Foreman will need to use Bundler to call Rake as normal, e.g. cloned from git, you should likely use RAILS_ENV=production bundle exec rake ran from your Foreman directory.

There are other ways to run the action discussed under API chapter.

4.1 Importing

If you have all attributes configured via Settings you can simply run following command

```
foreman-rake templates:import
```

This task based on $repo setting either uses git or scans local directory for templates to import. If you want to customize any setting for a particular run, just specify it as extra argument.

```
foreman-rake templates:import repo=https://github.com/theforeman/community-templates
```

The example above would clone the repository to some temp directory. If branch was specified, it switches to this branch. Then if dirname was provided, it will "cd" into it. Finally it searches for all files in this directory and subdirectories that has '.erb suffix. Then the template is parsed and imported (see below for more details)

```
foreman-rake templates:import repo=/var/lib/my_templates
```

Compared to previous example, this uses local directory. Therefore not git clone or branch checkout is done. If dirname is specified, the task goes into this subdirectory and searches for all files with `.erb' extension. Then the same parsing and importing process follows,

The parsing process reads the content of the file. It expects the content of the template including some template metadata. The metadata are an ERB comment on top of the file, for illustration see some template in community-templates repository. It's in yaml format and specifies template name, kind, compatible operating systems, organizations, locations and possibly other. The template gets it's name based on information from metadata. Before the template is saved in database, prefix setting is applied to it's name. Then the filter regular expression is matched against the prefixed name. If the filter matches, templates is saved. If negate is set to true, the template would be saved only if filter regular expression didn't match the template name. If the template with a given name exists in database already, it's content is updated.

The result of the whole import is printed on the STDOUT. The output contains list of created/updated templates as well as applied diffs.

A few more useful examples follow

```
# prefix all templates with "[comunity] " string to avoid overriding existing templates
foreman-rake templates:import prefix='[community] '

# get the latest templates (might be compatible with only Foreman nightly)
foreman-rake templates:import repo=https://github.com/theforeman/community-templates
branch=develop

tree /tmp/my-repo/
/tmp/my-repo/
└── large-ptable.erb

0 directories, 1 file

# this will create a template named "[community] large-ptable'
foreman-rake templates:import repo=/tmp/my-repo/ prefix='[community]
```

4.2 Exporting

If you want to edit templates directly in Foreman and then sync them back to external repositories, you need the exporting task. This plugin allows to export directly to git repository or local directory. For exporting to git repository, the user that's Foreman running under needs to have write (commit) access to this repository. If your settings is correctly configure, exporting can be triggered by running following command.

```
foreman-rake templates:export
```

If the repo is git repository, we clone it to a temp directory. Then we apply changes to template files and automatically create a commit with a generated message and push it back to origin. We do not delete files that are not present in Foreman database, so this should not be destructive. On the other hand if you want more control over the process or you want to use another SCM, local directory export is advised. To specify a local directory, just set the repo accordingly

```
oreman-rake templates:export repo=/var/lib/my_templates
```

When repo is set to local directory, exported templates are written into it. We don't touch any other files, so the directory can be under git (or any other SCM) control. You can then investigate changes by running git status and do the commit yourself when you're happy about the change.
For both git and plain directory targets, following applies. If dirname is specified, the export is written to this subdirectory of repo. Only templates with name matching to filter are exported. If filter is blank, all templates are exported. In case negate was used, the filter matching result is negated. This can be used to ignore some templates, e.g. following would not export templates starting with [private] and [wip] prefixes

```
foreman-rake templates:export filter='\[wip\]|\[private\] '
```

When a template is exported, there are 3 ways how to handle metadata.

refresh - remove any metadata from template content and generate new based on current template attributes and assignment

remove - remove any metadata from template content, useful if you want to add metadata manually or externally

keep - keep any metadata found in template content despite current attributes and assignments in database, useful when you want to send a patch to community-templates but you don't have all operating systems configured in your Foreman

Suppose you'd like to change something in default Foreman templates. You can use this plugin to make this process easy. First you need to update the template in you database and test it works. Then go to https://github.com/theforeman/community-templates and fork the repository. Clone your fork to the host where you have Foreman installed. Then run foreman-rake templates:export repo=/path/to/your/clone. Review the change, commit the change by git commit -a and push back to your github fork by git push. Then on github, open a new PR against theforeman/community-templates repositories.
("Foreman Templates 5.0 Manual," n.d.)

The upstream product documentation outlines two methods of using the plug-in of importing and exporting:

1. Using rake tasks
2. Using the API

In the first paragraph, this section informs a user about the first method only. However, in the second paragraph, the second method is mentioned as an alternative. This paragraph refers to the API chapter for the other ways of using the plug-in. In the table of contents, the API chapter is on the same level as the Usage chapter. Even though it outlines the usage of the plug-in as well. This chapter is short; however, it states that using the API is the recommended method to initiate import and export. In fact, the recommended way is hidden outside of the usage chapter and is not documented properly. These structural inconsistencies could lead to a situation, when a user uses a feature incorrectly, not as it was intended to use by engineers. This could lead to losing time, and result in a financial loss.

This section includes information about how the plug-in works, what tasks it performs when entering each command. This is helpful for developers to track the history of the feature, to see differences between different versions of the plug-in. However, ordinary users would like to use the plug-in. They do not need information about how their code works. For the upstream documentation, this kind of description is good. Because it is written and read mostly by developers.

The Importing section uses nominalization in the following form: “If you have all attributes configured”. This violates Tactic 5: Avoiding Nominalizations and can be replaced by: “Configure all attributes”.

The phrase “you should be able to run” uses phrasal verbs and too many words. This violates the Tactic 4: Avoiding Verbs with Two or Three Words in Them and, therefore, makes this phrase complicated for translating and understanding. This complicated structure can be replaced by a simple phrase “you can run”.

This section includes several long paragraphs. They must be separated into shorter once as per Tactic 14: Keeping Paragraphs Short.

4.4.2 The Draft Document

Importing and Exporting Templates

Both importing and exporting functions are available through a series of API calls. API calls use the Role-Based Access Control (RBAC) system, which means that they can be run as any user.

Prerequisites:

1. Configured Git server which allows connecting through SSH Keys.
2. The plug-in is configured in the TemplateSync tab in *Administer* → *Settings*.

3. There is an established SSH connection between your Satellite and the target Git server. For more information on establishing SSH connection, see To Configure SSH Access to Libvirt.

Using the API

The following procedure describes how to use API for exporting and importing.

Exporting

To edit templates directly in Satellite and synchronize them back to external repositories, you need the exporting task. This plugin allows to export directly to a Git repository or a local directory.

1. Execute the following command to export templates from your Satellite Server to the Git repository specified in settings:

```
$ curl -H "Accept:application/json,version=2" \
    -H "Content-Type:application/json" \
    -u admin:_password_ \
    -k https://_satellite.example_.com/api/v2/templates/export \
    -X POST

    {"message":"Success"}
```

2. You can override default settings by specifying them in the request. The following example exports templates to a stated local directory:

```
$ curl -H "Accept:application/json,version=2" \
-H "Content-Type:application/json" \
-u admin:_password_ \
-k https://_satellite.example_.com/api/v2/templates/export \
-X POST \
-d "{\"repo\":\"/another/repo\"}"
```

Importing

To Import templates back to the Satellite Server after their content was changed, use the following command:

```
```

```
$ curl -H "Accept:application/json,version=2" \
-H "Content-Type:application/json" \
-u admin:_password_ \
-k https://_satellite.example_.com/api/v2/templates/import \
-X POST

{"message":"Success"}
----
```

Note:

Templates provided by Satellite are locked. By default locked templates will not be imported. To overwrite this behaviour,, change the `Force import` attribute in `TemplateSync` menu into `yes` or supply `force` parameter `-d '{ "force": "true" }'` to the end of the import command.

(Synchronizing Templates with Git. Draft document. Author: S. Petrosian, 2018)

The draft section is shorter and better structured. It outlines one method of using the plug-in - through the API. However, its titles can be improved. As per Tactic 13: Use Clear Titles, every title and every word in a title must be meaningful. In this section, there is the title *Using the API*, after reading this title, a reader can misunderstand what it means and is likely to ask about the purpose of the API and why they should read the information again. To avoid this ambiguity, the title can be changed to be more explicit, for example *Using the API to Export and Import Templates.* However, this section has two subsections under it - *Exporting* and *Importing.* This nesting of titles makes the structure complicated and long. This can be improved by reducing the number of titles and making two wide titles: *Exporting Templates* and *Importing Templates.* This would make the structure easy to navigate and to scan through.

4.4.3 The Official Document

F.3. Importing and Exporting Templates

Importing and exporting tasks are available through a series of API calls. API calls use the role-based access control system, which enables the tasks to be executed as any user. The TemplateSync plug-in allows synchronizing with a Git repository or a local directory.

Prerequisites

For imported templates to appear in the Satellite web UI, each template must contain the location and organization that the template belongs to. This applies to all template types. Before you import a template, ensure that you add the following section to the template:

```
<%#
kind: provision
name: My Kickstart File
oses:
- RedHat 7
- RedHat 6
locations:
- First Location
- Second Location
organizations:
- Default Organization
- Extra Organization
%>
```

F.3.1. Synchronizing Templates with a Git repository

1. Configure a Git server that uses SSH authorization, for example gitosis, gitolite, or git daemon.
2. Configure the TemplateSync plug-in settings on a **TemplateSync** tab.
   a. Change the **Branch** setting to match the target branch on a Git server.
   b. Change the **Repo** setting to match the Git repository. For example, for the repository located in `git@git.example.com/templates.git` set the setting into `ssh://git@git.example.com/templates.git`.
3. Accept Git SSH host key as the foreman user
   ```
   # sudo -u foreman ssh git.example.com
   ```
4. You can see the `Permission denied, please try again.` message in the output, which is expected, because the SSH connection cannot succeed yet.
5. Create an SSH key pair if you do not already have it. Do not specify any passphrase.
   ```
   # sudo -u foreman ssh-keygen
   ```
6. Configure your Git server with the public key from your Satellite, which resides in `/usr/share/foreman/.ssh/id_rsa.pub`.
7. Export templates from your Satellite Server to the Git repository specified in the **TemplateSync** menu:
   ```
   $ curl -H "Accept:application/json,version=2" \
   -H "Content-Type:application/json" \
   -u login:password \
   -k https://satellite.example.com/api/v2/templates/export \
   -X POST
   ```

```
{"message":"Success"}
```

8. Import templates to Satellite Server after their content was changed:

```
$ curl -H "Accept:application/json,version=2" \
-H "Content-Type:application/json" \
-u login:password \
-k https://satellite.example.com/api/v2/templates/import \
-X POST
```

```
{"message":"Success"}
```

Note that templates provided by Satellite are locked and you cannot import them by default. To overwrite this behavior, change the Force import setting in the **TemplateSync** menu to yes or add the force parameter -d '{ "force": "true" }' to the import command.

F.3.2. Synchronizing templates with a local directory

Synchronizing templates with a local directory is useful if you have configured any revision control system repository in the local directory. That way, you can edit templates and track the history of edits in the directory. You can also synchronize changes to Satellite Server after editing the templates.

1. Create the directory where templates are stored and apply appropriate permissions and SELinux context:

```
# mkdir -p /usr/share/templates_dir/
# chown foreman /usr/share/templates_dir/
# chcon -t httpd_sys_rw_content_t /usr/share/templates_dir/ -R
```

2. Change the **Repo** setting on the **TemplateSync** tab to match the export directory /usr/share/templates_dir/.
3. Export templates from your Satellite Server to a local directory:

```
$ curl -H "Accept:application/json,version=2" \
-H "Content-Type:application/json" \
-u login:password \
-k https://satellite.example.com/api/v2/templates/export \
-X POST \
```

```
{"message":"Success"}
```

4. Import templates to Satellite Server after their content was changed:

```
$ curl -H "Accept:application/json,version=2" \
```

```
-H "Content-Type:application/json" \
-u login:password \
-k https://satellite.example.com/api/v2/templates/import \
-X POST

{"message":"Success"}
```

Note that templates provided by Satellite are locked and you cannot import them by default. To overwrite this behavior, change the Force import setting in the **TemplateSync** menu to yes or add the force parameter -d '{ "force": "true" }' to the import command.

NOTE

You can override default API settings by specifying them in the request with the -d parameter. The following example exports templates to the git.example.com/templates repository:

```
$ curl -H "Accept:application/json,version=2" \
-H "Content-Type:application/json" \
-u login:password \
-k https://satellite.example.com/api/v2/templates/export \
-X POST \
-d "{\"repo\":\"git.example.com/templates\"}"
```

("Appendix F. Synchronizing Templates with Git - Red Hat Customer Portal," n.d.)

The structure in the official documentation is clear and straightforward. As per Tactic 13: Use Clear Titles, the titles are meaningful and easy to navigate. As per Tactic 15: Text Highlighting, all labels are marked as bold, and all commands are kept in code blocks with special formatting. The values that a user must replace use italics, for example, *login:password* in the API calls. This is consistent throughout the guide and makes the text easier to comprehend.

In comparison with the draft document, this document lists only one prerequisite to the procedures. In addition, the prerequisites that are listed in the draft document, are implemented here into the procedures with a thorough description on how to perform the required steps. This decreases the time that a user would spend on searching how to meet the prerequisites and makes the procedure easier to follow.

5 Conclusion

In conclusion, it should be mentioned, that this specific style requires the technical writer to always keep in mind several main principles. Firstly, writing a concise clear document, that will be easy to read for the person who is not an expert in the field the document describes and could have weak language skills. This is where the second principle comes into play, the document must be easily understood in only one meaning even by not experienced non-native English speaker. That means that the language must be simple, the structure must not be aggravating, the sentences must be short, the punctuation must support the reader. In addition, the writer should always remember that the text he writes is likely to be translated into other languages, so the content must be unambiguous. The second chapter describes main principles that ensures that the writing is simple, clear, and well-structured.

Though technical writing is produced in a simple language using simple grammar structures and simple vocabulary, this is far from being the simplest task to fulfil. The comparison of technical documentation shows that there are many aspects to keep in mind while writing, for example:

- punctuation
- text highlighting
- text structure
- choosing correct words
- minimalism
- findability

Furthermore, except for these aspects, there are goals that one must fulfil - to make the text clear and simple. However, to make it usable and correct.

6 Bibliography

Appendix F. Synchronizing Templates with Git - Red Hat Customer Portal. (n.d.). Retrieved December 1, 2018, from https://access.redhat.com/documentation/en-us/red_hat_satellite/6.3/html/content_management_guide/synchronizing_templates_with_git

Brno University of Technology - Study in the Czech Republic. (n.d.). Retrieved May 1, 2019, from https://portal.studyin.cz/institution/vutbr

Cambridge Dictionary. (n.d.). Retrieved December 1, 2018, from https://dictionary.cambridge.org/

Chapter 1. Introduction to Red Hat Satellite 6 - Red Hat Customer Portal. (n.d.). Retrieved May 11, 2019, from https://access.redhat.com/documentation/en-us/red_hat_satellite/6.4/html/planning_for_red_hat_satellite_6/chap-red_hat_satellite-architecture_guide-introduction_to_red_hat_satellite

Chapter 9. Managing Errata - Red Hat Customer Portal. (n.d.). Retrieved May 11, 2019, from https://access.redhat.com/documentation/en-us/red_hat_satellite/6.3/html/content_management_guide/managing_errata

DeRespinis, F., International Business Machines Corporation, Jenkins, J., Hayward, P., & Laird, A. (2012). *The IBM Style Guide: Conventions for Writers and Editors*. Upper Saddle River, NJ, United States: IBM Press.

Foreman Templates 5.0 Manual (n.d.). Retrieved December 1, 2018, from https://theforeman.org/plugins/foreman_templates/5.0/index.html

Garner, B. A. (2016). *Garner's Modern English Usage*. New York, NY, United Stated: Oxford University Press.

Gerson, S. J., & Gerson, S. M. (2010). *Technical Communication: Process and Product* (3rd ed.). Upper Saddle River, NJ, United States: Prentice Hall.

Git. (n.d.). Retrieved December 1, 2018, from https://git-scm.com/

IRA - Definition by AcronymFinder. (n.d.). Retrieved December 1, 2018, from https://www.acronymfinder.com/IRA.html

Kenneth G. Budinski. (2001). *Engineers' Guide to Technical Writing*. Materials Park, OH, United States: IBM Press.

Kirkman, J. (2006). *Punctuation Matters: Advice on Punctuation for Scientific and Technical Writing* (3rd ed.). New York, NY, United States: Routledge.

Krhutová, M. (2009). *Parameters of Professional Discourse*. Brno, Czech Republic: Tribun EU.

Markel, M. (2015). *Technical Communication* (10th ed.). Boston, NY, United States: Bedford/St. Martin's.

Ogden, C. K. (1937). *Basic English: A General Introduction with Rules and Grammar* (6th ed.). London, United Kingdom: K. Paul, Trench, Trubner & co.

Petrosian S. D. (2018). *Synchronizing Templates with Git. Draft document.*

Pringle, A. S., & O'Keefe, S. S. (2009). *Technical Writing 101: A Real-World Guide to Planning and Writing Technical Content* (3rd ed.). Research Triangle Part, NC, United States: Scriptorium Publishing Services, Incorporated.

Tebeaux, E., & Dragga, S. (2015). *The Essentials of Technical Communication* (3rd ed.). New York, NY, United States: Oxford University Press.

Weiss, E. H. (2005). *The elements of international English style: a guide to writing correspondence, reports, technical documents, and internet pages for a global audience*. Armonk, NY, United States: M.E. Sharpe.