MAGDALENA KRAWIEC

ORCID: 0000-0001-9541-2515

Uniwersytet Rzeszowski, Poland

Terminological discrepancies in the field of software testing: A case of mistake, error, bug, defect, fault, and failure in the specialist language of IT

Introduction

Hatim and Mason (1990:32) and Bybee-Lonsdale (1996:91) delineated translation competence, in appreciably simplistic terms, as knowledge necessary to translate well. In consonance with that, one may venture a standpoint that terminological competence is knowledge requisite to adequately select terms in a specialist communication. An uninitiated specialist language user may contend with copious terminological hindrances, even at a level of seemingly elementary terms such as *mistake*, *error*, *bug*, *defect*, *fault* and *failure*.

The specialist language

The approach adapted in this paper concurs in the recognition presented by Grygiel (2017:3), who views specialist languages as “three dimensional multimodal forms of communication where specialist knowledge, professional practices and modes of linguistic expression are mixed together”. That wording is in consonance with Gotti’s (2003:24) elucidation, who delineates “specialized discourse as reflecting more clearly the specialist use of language in contexts which are typical of a specialized community stretching the academic, the professional, the technical and the occupational areas of knowledge and practice”.

Furthermore, the term specialist, rather than specialized or special language(s) is to be used in this article, on the grounds that, as pondered by Grygiel (2015:8),

“it emphasizes the fact that their users, in the first place, are experts in special work-related fields of human activity”.

Towards building a terminological competence

In recent years, new frontiers of knowledge were opened as a consequence of a multitude of technological advances in the specialist field of IT. That, in turn, has been conducive to the emergence of new technological semantic fields and lexical structures. As elaborated on by Ahmad (2009:7), “[o]n account of such profound and dramatic improvement new terminologies are coined to name the scientific and technological equipment and are to assign terms in new ways to activities, processes or subjects resulting from research and investigation”.

Most conspicuously, such terminologies, due to their inextricable connection to their contexts, should not be accounted for in isolation. Furthermore, scientific and technological discourses make use of them, so that information may be structured and the nature of more common sense meaning may be altered. Therefore, the most vital feature of professional language users lies in their ability to readily locate terms in corresponding conceptual systems. Faber (2004) presupposes that in order to adequately and successfully translate source texts, professional language users ought to arrive at a more specialist knowledge in particular specialist fields. Therefore, attempts should be made to extend and develop language users’ awareness as regards available terminological resources. That seems particularly important, on the grounds that uninitiated language users are particularly prone to take what is written in a dictionary at face value, which is underlain by a fact that they are heedless of meanings diversification across discourses.

Faber (1998:11), avers that “students naively regard dictionaries with exaggerated veneration” and perceive them as “invariably reliable”. This is considerably explicit in their peculiar inclination to treat even the most unsound entries as “sacred scripture”.

Building terminological awareness within any specialist language entails establishing inter-linguistic correspondence, using information available in monolingual dictionaries. In this case, dictionary entries may be used to “construct parallel representations of part of a semantic field”.

As accounted for by House (2000:150), scrutinising the connection between translation and terminology needs to be underpinned by understanding that both of them are substantially governed by semantic, pragmatic, contextual, and cultural factors that function at the level of source and target language. Cabré (2000:73–74) further elaborates and deems that that both function as “convergence points” for other knowledge areas, viz., e.g., linguistic, cognitive or communication sciences.

Most importantly, terminology germane to Information Technology or any other specialist field, is not established as a speech act in itself, but constitutes

rather a catalogue of terms. Ergo, one may infer that it is an instrument used to communicate in a specialist domain.

Mistake and error

A *mistake* is understood as an incorrect human action, misconception or misunderstanding and arises on the part of field specialists, e.g., software developers, analysts, and testers. Therefore, when a developer *makes a mistake*, for instance, types a variable name incorrectly, it results in an *error*. The wording formulated by the Online Collins Dictionary (2018) also highlights the human component: ‘if you make a mistake, you do something which you did not intend to do, or which produces a result that you do not want’, and may be found functional in the process of terminological research. In simplistic terms:

- (1) I *made a mistake*, because I confused the requirements. It caused an *error*.
- (2) I *made a mistake*, because I confused the syntax. It caused an *error*.
- (3) I *made a mistake*, because I miscalculated the values. It caused an *error*.
- (4) I *made a mistake*, because I misinterpreted the values. It caused an *error*.

That seems to find confirmation in Galin’s (2004) words, who presupposes that an *error* may be either a minor syntax or grammatical mistake in the code, or a logical error, such as, e.g., multiplying instead of adding two operands. Panko (1998:15–21) adduces akin taxonomy, identifying, e.g., omission errors (facts omitted in code, which ‘lead to an incorrect, bottom line value’), logic errors (an algorithm that is incorrect or was implemented incorrectly), or interestingly, strong but wrong errors, such as functional fixedness, i.e., a fixed mindset.

Despite the fact that the International Software Testing Qualifications Board (ISTQB) Glossary (2006) delineates *mistake* as a synonymous expression for *error*, a *mistake* is to be understood as a general rather than specialist language unit, accounting solely for an incorrect human action resulting in further “specialist” consequences, i.e., deviation from actual and expected value and unexpected system behaviour. That seems to find confirmation in the wordings furnished by Webster’s New Dictionary of Synonyms (1984) for the entry *error*, which “etymologically implies deviation; it suggests culpability but not necessarily carelessness or intention, for it implies a guide to be followed such as a record or manuscript, or a rule or set of rules, or a principle, law, accepted code, or the like’, whereas *mistake* is accounted for as implying ‘misconception, misunderstanding, a wrong but not always blameworthy judgement, or inadvertence; it expresses less severe criticism than error”.

That said, in IT-related terminology, an *error* is a *mistake*, or a result of a mistake, caused by a software developer, or any other field specialist. Therefore:

Fundamentals of Software Testing (2013) define *error* as human action underpinning the root of a *defect*. Ergo:

As presupposed by Naik and Tripathy (2008), an *error* is a state of the system that can result in a failure. Thus:

Moreover, Galin (2004) suggests that an error may lead to software malfunction, however in some of the cases sometimes ‘the fault may be corrected or neutralized by subsequent code lines’. Therefore:

Galin’s vantage point was further elaborated on by Myers and Ko (2005:44), who advanced a three-partite taxonomy of software errors, runtime faults, and runtime failures to elucidate the complex nature of software anomalies. Their wording reads as follows: software error is fragment of code that may cause a runtime fault during program execution. For example, software errors in loops include a missing increment statement, a leftover “break” command from a debugging session, or a conditional expression that always evaluates to true. It is important to note that while a runtime failure guarantees that one or more runtime faults have occurred, and a runtime fault guarantees that one or more software errors exist, software errors do not always cause runtime faults, and runtime faults do not always cause runtime failures. Also note that under our definition, a single change to the design specifications can introduce an arbitrary number of software errors (Myers and Ko 2005:44).

In simplistic terms, *errors* occur in code, *faults* during execution, and *failures* in program behaviour:

Interestingly enough, despite the apparent comprehensiveness in English, in Polish both *error* and *mistake* are translated jointly as *błąd*, hence in the mind of

a Polish-speaking end user there is no semantical distinction between those two words. Consequently, the cause and effect component is lost.

On the other hand, while translating from Polish into English, some form of terminological dissonance may occur especially when the translator lacks specialist knowledge, and does not discriminate between the two words. That constitutes a frequent example, and as postulated by Montero and Faber (2009:1), “polysemy and synonymy occur quite frequently in specialized communication, and [may] generate translation problems“. Therefore, it is instructive and prudent to draw language users’ attention to semantic dissimilarities across discourses and language varieties.

Defect, bug, fault

IT-oriented definitions account for *defect* as a deviation from the requirements resulting in system malfunction. Thus, the said deviation may be any function or functionality that was specified by the customer but was not developed or implemented by the software engineer. That said, a *defect* occurs when software quality departs from the specified value and does not meet end user’s expectations. This approach concurs in the recognition with the delineation furnished by Online Oxford Dictionary (2018) which defines *defect* as ‘a shortcoming, imperfection, or lack of something’.

The ISTQB Glossary (2006) advances the following elucidation for the term *defect*: ‘an imperfection or deficiency in a work product where it does not meet its requirements or specifications’. Interestingly enough, it also provides *bug* and *fault* as synonymous terms. Furthermore, as juxtaposed with other entries such as, e.g., *fault seeding*, *fault injection* or *fault tolerance* it is conspicuous that the authors apply those terms interchangeably:

- (1) *fault seeding* – The process of intentionally adding defects to those already in the component or system.
- (2) *fault injection* – the process of intentionally adding defects to a system [...].
- (3) *fault tolerance* – The capability of the software product to maintain a specified level of performance in cases of software faults (defects) or of infringement of its specified interface.

Foundations of Software Testing: ISTQB Certification propounds the following formulation: *defect (fault, bug)* ‘[is] a flaw in a component or system that can cause the component or system to fail to perform its required function, e.g., an incorrect statement or data definition. A defect, if encountered during execution, may cause a failure of the component or system’. Conversely, Gomes (2009),

adduces the same elucidation, yet for the term *fault*, and provides *defect* as a synonym. Interestingly, in the IEEE Standard Glossary of Software Engineering Terminology there is no entry for the term *defect*.

In largely simplified terms, from the end user's perspective, one can account for *defect* using the following case scenarios:

- (4) End user wants to pay using online payment methods.
The functionality of selecting an online payment method is missing.
- (5) End user wants a functionality that checks spelling.
The functionality of the spelling check is missing.

Hence:

and

and

Interestingly, in the IEEE Standard Glossary of Software Engineering Terminology, neither an entry for *defect* nor *bug* was provided. As regards the latter, a reference to *error* and *fault* was adduced, which confusingly points out to synonymy between the three terms. In consonance with that:

ReQtest Online (2018) elaborates on *bug* in the following way: 'a slang term for fault, defect, or error. Originally used to describe actual insects causing malfunctions in mechanical devices that predate computers'. That seems to concur with a generally held vantage point that the term is most commonly used in everyday programming parlance. That said, *defects* (or *bugs*, or *faults*) occur because of an error in logic or in coding which results into the failure or unpredicted or unanticipated results. Divergent vantage point is held by McDowell and Helmbold (1989:593–622) and Krawczyk, Wiszniewski and Mork (1994:91), who deem the term *bug* to be most often applied 'to address all kinds of damages caused by faults. The further elaborate on by saying that the term is not precise, but it reflects very well the uncertainty surrounding the

origins of faults: bugs are really creeping into programs when programmers are wrong but confident’.

As adduced in Myers and Ko (2005:43), a *fault* ‘is a machine state that may cause a runtime failure (e.g., a wrong value in a CPU register, branching to an invalid memory address, or a hardware interrupt that should not have been activated)’, therefore:

Contrary to that view, as averred by Naik and Tripathy (2008), as well as Roggio, Gordon, and Comer (2013:2), *errors* are underlain by faults:

The IEEE Standard Glossary of Software Engineering Terminology, on the other hand, furnishes the following wording: *fault* ‘is an incorrect step, process, or data definition in computer programs’.

Failure

Failure appears as the least debatable term under analysis. It is understood ‘as departure of the operational software system behaviour from user expected requirements. A particular failure may be caused by several faults and some faults may never cause a failure’. This wording is congruous with Lanubile and Visaggio (1997:39–57), the IEEE Standard Glossary of Software Engineering Terminology as well as Myers and Ko (2005:43), who delineate it as ‘an event that occurs when a program’s behaviour – often some form of visual or numerical program output – does not comply with the program’s design specifications’.

As presupposed by Krawczyk, Wiszniewski, and Mork (1994:91) jointly account for this term as a ‘[t]he inability of a functional software unit to perform its required functions is called a failure. A failure severity can be categorized in different ways, for example, as a software unit being inoperable, its major function being inoperable, or just a cosmetic failure. A failure is caused by a defect encountered during code execution’. That remains in consonance with the wording provided, e.g., by Selby (1990:444–454). Consequently:

and

Having provided the above quoted definitions, one may arrive at a conclusion that a *failure* signals the occurrence of a *fault* or a *defect*, i.e., a *failure* cannot be physically fixed or removed, without fixing or removing the *fault* or *defect*. In further elaboration, much as a *failure* may occur in a software or one of its modules, it is not viable for it to be found in a code or documentation. In consonance with that, a *failure* is a method of searching for defects.

In simplistic terms, from the end user's perspective, one can account for *failure* using the following approach:

fault or defect or bug + CIRCUMSTANCES = failure

Let us consider the following case scenario:

- (6) you want to sign in with Google in an online shop, and having a Google account constitutes the CIRCUMSTANCES.

If clicking on the Google button triggers no action, i.e., the end user is not directed to the Google sign-in page, a system *failure* may be observed. From the software tester perspective, they may either report the failure, or look for a defect, fault, or bug, triggering it.

Software malfunctions – possible systematization

Having arrived at a comprehensive, yet sometimes contradictory account of software testing terms, one may venture to illustrate the relations graphically. The International Software Testing Qualifications Board (ISTQB) Glossary explains that 'a human being can make an error (mistake), which produces a defect (fault, bug) in the program code, or in a document. If a defect in code is executed, the system may fail to do what it should do (or do something it shouldn't), causing a failure'. This account is indeed far too general to encompass all technicalities and discrepancies between the terms in question, yet, as juxtaposed with other elucidations, vaguely classifies software anomalies into two categories, i.e., malfunctions occurring on the part of a software engineer (human component) constituting the cause, and those on the part of software, constituting the effect:

This two-partite classification still remains a little vague in its entirety, due to the occurrence of three elements on the part of software.

Over a decade ago, the ISEB Glossary postulated a distinction between *errors* and *defects*. *Defects* ensue from *errors*, which are in turn made by humans. Much as Polish practitioners in the field of software testing and programming are well-acquainted with ISTQB, and all terms under research are perfectly translatable into Polish, one may discern an inclination and certain willingness towards the use of *mistake*, *error*, and *bug* only. Interestingly, the first two have the same equivalent in the target language, i.e., *blad*, whereas *bug* permeated into Polish both phonologically, semantically, and graphically and is declined in accordance with applicable grammar rules, as in the case *We’ve got a bug*, *Mamy buga* [PL], or *There are five bugs*, *Jest kilka bugów* [PL]. The rest of the terminology did not get naturalized in this parlance.

Many sources, especially those that eschew following the ISTQB trend, encompassing both software testing practitioners and amateur adepts, incline to define software anomalies and malfunctions as *errors*, or *bugs*, therefore, one usually comes across a software *error* rather than *defect* or *fault*. Theoreticians endorsing the ISTQB certification claim that an *error* has to be ‘made’. It is a human, who makes *errors*, which resultantly generate consequent *defects*, which in turn may lead to *failures*.

Practitioners largely simplify this issue. Everything may be accounted for as an *error*, or *bug*. One may talk about *human error*, *software error/bug*, *interface error/bug* etc. Most frequently, there is no differentiation between the cause, and effect. Conspicuously enough, in English the distinction appears more useful and feasible in theoretical considerations, rather than practical application. Therefore in English, a four-partite distinction appears the most feasible:

The graphical representation may be further simplified:

Conclusion

Conspicuously enough, semantic research on the most prominent knowledge sources for software testing practitioners is far from straightforward. The extent to which the perception differs in Polish and English is indeed substantial.

Both language environments seem to concur in the recognition that there are two groups of anomalies, with the first one embracing solely the human, and the second one – the software component. A person (software developer, analyst, architect, tester, QA specialist etc.) makes a mistake which results in an error. Subsequently, the error may result in a defect/fault/bug, which while executed may cause a failure. In Polish, ‘fault’ (*usterka*), and ‘defect’ (*defekt*) are rather infrequent choices, and ‘bug’ quite frequently appears as an umbrella term for all instances of software anomalies. On the other hand, explicit willingness and inclination towards the use of English borrowings in Polish (*mamy buga*), enables to segregate the two categories.

As already mentioned, *mistake*, *error*, and *bug* are all translated into Polish as *bląd*, which may at first result in a cognitive chaos as regards the differentiation between these three terms, one the grounds the juxtaposition of the cause, and the effect is lost, i.e., no distinction is made between the human and the computer component. Furthermore, while translating from Polish into English, the uninitiated user may encounter certain hindrances for they have to select from at least three term, which usually, without specialist knowledge and practices, seems to be a formidable task.

On the whole, confusing, and to some extent contradictory synonymy in specialist and general language dictionaries constitutes another challenging issue. In great majority of them, the terms in question are mingled and entries provide them as synonymous words. One may therefore assume that in English there is a much bigger room for terminological flexibility which, simultaneously, has certain hindrances. The paper may indeed serve as a basis for further research on specialist terms in the field of software testing, in a sense that language tendencies alter along with the development of modern technologies, even in a seemingly homogenous specialist field such as software testing.

References

- AHMAD Jameel, 2009, Teaching of English, New Delhi.
 BYBEE-LONSDALE Allison, 1996, Teaching Translation from Spanish to English, Ottawa.
 CABRÉ Maria Teresa, 2000, La enseñanza de la terminología en España: Problemas y propuestas, in: *Hermèneus* 2, pp. 41–94.
 FABER Pamela, 1998, Translation competence and language awareness, in: *Language Awareness* 7, pp. 9–21.

- FABER Pamela, 2004, Terminological competence and enhanced knowledge acquisition, in: Fleischmann E./Schmitt P./Wotjak G. (eds.), *Translationskompetenz*, Tübingen, pp. 479–499.
- FUSARO Pierfrancesco / LANUBILE Filippo / VISAGGIO Giuseppe, 1997, A Replicated Experiment to Assess Requirements Inspection Techniques, in: *Empirical Software Engineering* 2(1), pp. 39–57.
- GALIN Daniel, 2004, *Software Quality Assurance: From Theory to Implementation*, Cambridge.
- GOMES Luís / FERNANDES João M., 2009, *Behavioral Modeling for Embedded Systems and Technologies: Applications for Design and Implementation*, Hershey/New York.
- GOTTI Maurizio, 2003, *Specialised Discourse: Linguistic Features and Changing Conventions*, Bern.
- GRYGIEL Marcin, 2015, Business English from a linguistic perspective, in: *English for Specific Purposes World*, Special Issue 16, pp. 1–12.
- GRYGIEL Marcin, 2017, Specialist languages and cognitive linguistics: A marriage of convenience or irreconcilable opposites?, in: Grygiel M. (ed.), *Cognitive Approaches to Specialist Languages*, Cambridge Scholars Publishing, Cambridge, pp. 2–13.
- HATIM Basil / MASON Ian, 1990, *Discourse and the Translator*, London.
- HOUSE Juliane, 2000, Consciousness and the strategic use of aids in translation, in: Tirkkonen-Condit S./Jääskeläinen R. (eds.), *Tapping and Mapping the Processes of Translation and Interpreting*, *Outlooks on Empirical Research*, Amsterdam/Philadelphia, pp. 149–162.
- KO Andrew / MYERS A. Brad, 2005, A Framework and Methodology for Studying the Causes of Software Errors in Programming Systems, in: *Journal of Visual Languages & Computing* 16(1–2), pp. 41–84.
- KRAWCZYK Henryk / WISZNIEWSKI Bogdan / MORK Peter, 1994, Classification of Software Defects in Parallel Programs, in: *Technical Report 2*, Gdańsk, pp. 89–111.
- MCDOWELL Charles E./ HELMBOLD David P., 1989, Debugging Concurrent Programs, in: *ACM Computing Surveys* 2, pp. 593–622.
- MONTERO MARTÍNEZ Silvia / FABER Pamela, 2009, Terminological competence in translation, in: Alcina A. (ed.), *Teaching and Learning Terminology: New Strategies and Methods*, pp. 88–104.
- NAIK Kshirasagar / TRIPATHY Priyadarshi, 2008, *Software Testing and Quality Assurance: Theory and Practice*, Hoboken (New Jersey).
- PANKO Raymond, 1998, What we know about spreadsheet errors, in: *Journal of End User Computing* 10, pp. 15–21.
- ROGGIO Robert / GORDON Jamie / COMER James, 2013, Taxonomy of Common Software Testing Terminology: Framework for Key Software Engineering Testing Concept, in: *Proceedings of the Conference for Information Systems Applied Research* 6, pp. 1–9.
- SELBY Richard, 1990, Empirically based analysis of failures in software systems, in: *IEEE Transactions on Reliability* 39, pp. 444–454.
- Webster's New Dictionary of Synonyms, 1984, New York/Springfield.

Online sources

<https://glossary.istqb.org/>.
<https://www.collinsdictionary.com/>.
<https://en.oxforddictionaries.com/>.
<https://www.ieee.org/publications/rights/rights-glossary.html>.
<https://reqtest.com/blog/glossary-of-testing-terms/>.