Software chrestomathies

Ralf Lämmel

Universität Koblenz-Landau, Germany

Abstract

A software chrestomathy is a collection of software systems ('contributions') meant to be useful in learning about or gaining insight into programming and software engineering. We describe the essential and potential characteristics of such collections. Eventually, we compile a research agenda on software chrestomathies.

Keywords: Chrestomathy, Program chrestomathy, Software chrestomathy, Philology, Linguistics, Programming languages, Software engineering, Understanding software

Contents

1. Prologue

There is little doubt that *examples* are generally useful for teaching and learning and understanding. For instance, well-chosen program samples could help those learning programming (languages). Obviously, a content provider (a teacher, a textbook author, or a wiki editor) with commitment to examples should follow some *principles* of collecting and organizing examples as well as integrating examples with other forms of content. Research on such principles is essentially research on chrestomathies, which is the topic of this paper.

A *software chrestomathy* is a collection of software systems ('contributions') meant to be useful in learning about or gaining insight into software languages, software technologies, software concepts, programming, and software engineering. For instance, a chrestomathy could contain a number of systems implementing the same requirements on different platforms, thereby allowing someone with knowledge of one platform to understand another platform essentially by comparing the two systems for the two platforms.

The notion of software chrestomathy was introduced recently in a software language engineering context [1]; it is very close to the notion of *program or programming chrestomathy* which has been in potent and pragmatic use in the broad programming community for several years now in the wild.[1,2]

The chrestomathy notion has its origin in philology and linguistics, where the term is in use since at least the 1830ies [2]. 'Chrestomathy' is, in fact, formed from the Greek terms 'chresto' (Engl.: 'useful') and 'mathein' (Engl.: 'to learn').

In the present paper, we describe the essential and potential characteristics of chrestomathies in programming and software (language) engineering. To this end, we also look at various actual chrestomathies. Ultimately, we present a research agenda on software chrestomathies.

2. The chrestomathy notion

Let us first understand the origin of the chrestomathy notion proper. Based on diverse dictionary entries,[3] we synthesize the following definition of chrestomathy: *a collection of literary passages in typically one language from one or more authors compiled by one or more chrestomathy authors as an aid in learning a language*.

This short definition suggests some obvious questions, which are eventually also interesting in a programming or software engineering context. How exactly could a chrestomathy be expected to be *chrestomathic*, i.e., conductive to useful learning? In particular, what would be the more specific *learning objective* and the means of realizing the objective? Also, what *selection criteria* would be applied for including literary passages? Further, what *structuring principles* and forms of *content enrichment* could be possibly used to meet the chrestomathic principle? Let us look at a few chrestomathies for inspiration.

3. Examples of chrestomathies

First, we consider examples of chrestomathies, as they are used in linguistics. A short analysis of these examples prepares us for a profound understanding of chrestomathies in a programming or software engineering context.

Assyrian Grammar with Chrestomathy and Glossary. [10] A chrestomathy of the Assyrian language which is meant to illustrate the description of the language's grammar.

Coptic Gnostic Chrestomathy. [11] A chrestomathy of the Coptic language which is systematically edited to include annotations for grammatical analysis such as relationships between prepositions, verbs, and nouns.

Chrestomathy of Classical Arabic Prose Literature. [12] A chrestomathy of the Arabic language (in fact, a selection of classical Arabic prose) which is accompanied with grammatical and lexical commentaries as well as notes pointing to historical, cultural, and religious background information.

These illustrations support the following claimed characteristics of chrestomathies. First, chrestomathies often support those who want to learn a language *grammar*, but *additional knowledge dimensions* such as culture or history may be served as well. Second, chrestomathies are typically *more than just plain collections of literary passages*; they tend to include comments, annotations, translations, and links.

Now let us also mention an example of a chrestomathy that is closer to philology; it happens to be a classical piece of literature in itself:

[1] `http://en.wikipedia.org/wiki/Chrestomathy` – accessed 5 Aug 2013.

[2] `http://c2.com/cgi/wiki?ProgrammingChrestomathy` – accessed 5 Aug 2013.

[3] *Selected dictionary entries*: a selection of passages used to help learn a language [3]; a volume of selected passages or stories of an author [3]; a selection of literary passages, usually by one author [4]; an anthology used in studying a language [4]; a collection of literary passages, used in the study of language [5]; a collection of selected literary passages, often by one author and esp. from a foreign language [6]; a collection of literary selections, especially in a foreign language, as an aid to learning [7]; a collection of literary selections from one author [7]; a selection of choice literary passages from one or more authors [8]; a selection of passages from different authors that is compiled as an aid in learning a language [9].

A Mencken Chrestomathy. [2] A collection of commentary and criticism published by H.L. Mencken over many years in various newspapers. Each passage carries a short headline and there are various groups (themes) such as 'man', 'women', 'religion', 'morals', and 'death'.

A chrestomathy like the Mencken one is arguably little more than an *anthology*—a collection of literary works chosen by the compiler (which is, in this case, Mencken himself). Still such collections are useful ('chresto') to learn ('mathein') the language—either in general or in a specific domain.

4. Characteristics of program chrestomathies

Let us start with some trivial (and essential) characteristics. A program (or programming) chrestomathy collects programs rather than literary works. A program chrestomathy is meant to be useful to learn about programming and programming languages rather than natural languages.

Other less basic characteristics follow; not all of them are essential. Most of these characteristics further set apart program chrestomathies from the philological or linguistic ones.

Community effort. In philology or linguistics, a chrestomathy is typically compiled by a single author or an author team, and the authors of the collected literary works do not need to take part in the chrestomathy effort. A program chrestomathy relies on programs ('contributions') to be authored specifically for the chrestomathy to meet the requirement specification. Therefore, in practice, program chrestomathies often turn into community efforts to combine expertise and subdivide work. One-man chrestomathies are the exception.

Requirement specification. The programs of a program chrestomathy are expected to meet a certain requirement specification, typically expressed in terms of *tasks*. These may be fine-grained tasks (e.g., 'write a function to send an email') or coarse-grained tasks (e.g., 'build a human resources management system'). In philology and linguistics, there is no proper counterpart for the requirement specification of a program chrestomathy. Instead, collected works are drawn from existing literary works driven by the objective to compile a coherent and representative corpus.

Multiplicity of languages. A program chrestomathy may very well collect programs of just one specific language, but it is quite common that a program chrestomathy is actually designed to serve comparison across languages, e.g., comparison of expressiveness, style, or performance. In philology and linguistics, chrestomathies are language-specific, even though exceptions may exist.

Variety of objectives. A program chrestomathy will almost certainly serve learning or understanding in one way or another, e.g., understanding of programming style and techniques across languages based on comparison. There are also objectives that are not primarily linked to learning and understanding, e.g., demonstration (evidence) of expressiveness or measurement of performance across languages. Of course, a secondary link to learning and understanding may exist. For instance, the demonstration of expressiveness conveys idioms.

Evolution. In philology or linguistics, chrestomathies tend to be relatively definite artifacts, even though revisions may be conceivable in principle. A program chrestomathy will typically grow and otherwise evolve over some time in a fine-grained manner, perhaps also in terms of the underlying requirement specification. Program chrestomathies may eventually cease to evolve because, for example, of decreasing interest in a particular chrestomathy. In this manner, chrestomathies may potentially end up in a somewhat outdated state as programming languages and technologies continue to evolve.

Infrastructural support. In a simple case, a program chrestomathy is little more than a web page. In a more advanced case, a program chrestomathy relies perhaps on a repository, some automated sanity checks, a wiki for collaborative editing of code, documentation, and attachment of metadata. As the complexity of infrastructural support increases, as the collected items are increasingly subjected to a software engineering discipline, we may be moving closer to software chrestomathies; see §6.

5. Examples of program chrestomathies

The Evolution of a Haskell Programmer. [13] This is a Haskell-specific chrestomathy; all other entries in this section are multi-language chrestomathies. The programs implement the *Factorial* function in many different ways; thus dealing with a single fine-grained task. The chrestomathy supports the learning objective of illustrating diverse programming techniques and styles in Haskell. Infrastructurally, the chrestomathy is essentially just a website maintained by a single person; each solution is sparsely commented to hint at the style or technique at hand.

99 Bottles of Beer. [14] Each program is supposed to print (generate) the lyrics of the songs '99 Bottles of Beer'. This may count as a humorous objective, but basic programming techniques are illustrated across languages, as the use of iteration or recursion is stipulated. Clearly, the chrestomathy is concerned with a specific functional requirement to produce some prescribed output. Infrastructurally, the chrestomathy offers a submission form; the team behind the chrestomathy processes submissions and organizes the programs on web pages.

OO shapes. [15] Each program is supposed to implement some given tennets of object-oriented programming (specifically, polymorphism) based on shape objects for circles and rectangles. The objective is mainly concerned with expressiveness and OO style across languages specifically also including non-object oriented programming languages. There is a single coarse-grained task which breaks down into the implementation (or encoding) of certain interfaces or classes and behaviors for constructing and manipulating shapes. Infrastructurally, submissions are emailed to the maintainer who organizes them on a web site. In fact, other parties also maintain some sub-chrestomathies of *OO shapes* independently.

Rosetta Code. [16] The chrestomathy covers programming broadly in that it offers hundreds of tasks organized in several groups such as networking and algorithms. The learning objective is indeed to show how certain recurring (fine-grained) programming problems are expressed in a language, also making good use of libraries. A learner may systematically compare solutions of tasks between a familiar language and one that is to be learnt. Infrastructurally, the chrestomathy leverages a relatively advanced wiki with forms, categories, and other features supporting organization, submission, and association of metadata.

Beautiful Code. [17] This is actually a textbook with chapters from several leading computer scientists. Each chapter covers a different development project as chosen by the chapter's author. Some of the source code is available online. Different programming languages and programming domains are at play. There is no overarching requirement specification that underlies these chapters, but the implicit assumption is certainly that the chapters complement each other in some useful manner. The reader is supposed to learn from the reflections and insights of the contributing scientists.

Keyword Pattern Matching. [18] This is actually a journal paper which describes a taxonomy of a certain class of algorithms, namely many sublinear (multiple) keyword pattern matching algorithms. The actual algorithms are shown in a dialect of Dijkstra's guarded command language. We quote from the paper: "The taxonomy is based on deriving the algorithms from a common starting point by adding algorithm and problem details, to arrive at efficient or well-known algorithms." Obviously, this program (algorithm) collection is useful in learning about algorithmic options and associated complexity within the domain. New developments in the domain should be preferably characterized by positioning them within the taxonomy.

6. Characteristics of software chrestomathies

The software chrestomathy notion generalizes the program chrestomathy notion: **a software chrestomathy collects software systems**; often these are tiny systems, but systems nevertheless—as opposed to programs (as in 'source code' or fragments thereof). That is, a system may break down into models, source code, modules, packages, generated code, data files, database images, build scripts, etc.; a system should be buildable and runnable and testable.

Such a step from programs to systems naturally expands the scope in which chrestomathies may be useful for learning and gaining insight. That is, the scope may include now software development or software engineering in addition to just programming. Further, software engineering ideas may be applied to the development (the assembly), the documentation, the maintenance, and the use of software chrestomathies. This is also expressed by the following

characteristics. We admit that the decision on when a given chrestomathy is still a program chrestomathy or when it is already a software chrestomathy may be subject to discussion.

Revision and access control. A collection of systems cannot reside on just a web site. Rather the systems with all the involved artifacts and all related documentation should be managed through a revision control system, thereby supporting collaborative development and evolution of systems. In fact, access control is also needed to maintain and enforce user profiles on the grounds of the different kinds of persons ('roles') involved: individual contributors, maintainers of infrastructure, developers with code access, authors with documentation access, and authors of task descriptions. This may already be true, to some extent, for some program chrestomathies.

Quality assurance. The systems should be treated as regular software systems in that they should be checked to meet reasonable quality parameters, e.g., related to test coverage or code smells. Additional aspects of quality assurance may relate to the specifics of the chrestomathy. For instance, conformance checks may be needed to validate that systems implement tasks as claimed by the metadata and they comply with a specific style regarding code, design, or documentation. Arguably, a chrestomathy could also contain intentionally 'bad' examples from which to learn. In this case, quality assurance means that one has to check for non-qualities rather than qualities.

Richer metadata. Trivial metadata is already needed for a program chrestomathy: the language of a collected program, the task implemented (or additional details for coarse-grained tasks), and the contributor. A software chrestomathy may require richer, software engineering-related metadata, e.g., the software technologies leveraged in development and at runtime, the involved software languages in addition to the primary programming language, the software concepts (e.g., patterns) demonstrated by the system, or the relationships between different systems.

Process management. In the interest of scalability and quality, the process of collecting and maintaining contributions should be properly defined and executed. For instance, a contribution may go through the following lifecycle: submission of a proposed contribution, review, acceptance or rejection or conditional acceptance subject to required improvements, documentation, attachment of metadata, online presence also involving online feedback, repeated revision, possibly removal. Process management relies on tool support and human involvement.

7. Examples of software chrestomathies

Java Pet Store. [19] Originally, the Pet Store was a reference application by Sun Microsystems to convey best practices of Java Enterprise Development. However, platform providers (e.g., for Spring or .NET) also developed the Pet Store application—for comparison. This software chrestomathy is virtual in the sense that there is no central infrastructure or repository to maintain the different systems, but they are obtainable independently. These systems have been discussed by the community and they influenced each other.

The Computer Language Benchmarks Game. [20] This chrestomathy addresses the objective of performance measurements across languages and language implementations. Thus, the collected systems are essentially implementations of benchmark problems. Contributions are highly constrained to be 'regular', readily executable, and amenable to useful measurements. Thus, some degree of process management and quality assurance can be attested for the Benchmarks Game. Learners benefit from this chrestomathy, as it conveys efficient implementation techniques for the different programming languages.

101companies. [1] This chrestomathy relies on a feature model (say, a requirement specification) for a simple information system to touch upon various fundamental techniques of programming, design, data modeling, and non-functional requirements. The chrestomathy is meant to be useful to learn about not just programming languages, but software languages more generally as well as software technologies, and technological spaces. 101companies leverages a semantic wiki and a distributed repository for maintaining the chrestomathy. *Linked Data* principles are adopted to surface primary chrestomathy artifacts as well as various derived resources for programmatic use and human users. This chrestomathy is readily used in programming courses as providing the running example and ontological support for those courses.

8. A research agenda

Overall, we observe the following challenges regarding software chrestomathies. Foremost, we should be able to establish that a given chrestomathy is indeed useful for learning or understanding; we may even want to quantify such usefulness. Thus, research on empirical *validation of usefulness*, e.g., the *educational value*, is needed. Another area of research concerns the utilization of the usefulness of chrestomathies in the sense of *integration with teaching*. Further, a range of challenges has to do with *productivity of development and maintenance* for chrestomathies, thereby calling for involved technical improvements such as polyglot analyses and transformations of chrestomathy artifacts. Finally, chrestomathies are becoming so complex knowledge bases that we need to work out means of *knowledge management*, for example, to guarantee quality of the content, to integrate resources, and to enable effective evolution. We provide more details on these areas of research.

Validation of usefulness. Some chrestomathies appear to be potentially useful in an obvious manner, e.g., by assessing expressiveness or performance of given software technologies (e.g., [21, 22, 20]). Other chrestomathies may lack such an 'obvious' argument for potential usefulness (e.g., [13, 17, 1]); nevertheless, they are assumed to provide some educational value. Research is needed to effectively model potential usefulness and measure actual usefulness.

Chrestomathies versus fora. Program and software chrestomathies are obviously not the only example-based tools that are meant to be useful in learning about and gaining insight into programming and software engineering. In particular, these days, fora such as *Stack Overflow*[4] or *MSDN*[5] are popular productivity tools for developers. Hence, research on the validation of educational value of chrestomathies also should determine the specific pros and cons of chrestomathies versus fora to ultimately suggest some form of synergy based on an integration of concepts.

Crowdsourcing. The collection of data (feedback) from chrestomathy users may provide a means of quality assurance as well as a support utility for validation, as discussed above. For instance, such crowdsourcing may help with assessing idiomatic use of programming languages or software libraries across different ('competing') contributions. Further, crowdsourcing may provide feedback on the learning experience of chrestomathy users. Rich and reliable feedback is needed—as opposed to anonymous likes or dislikes. Research is needed here to transpose existing techniques for crowdsourcing and corresponding data assessment (e.g., [23]) to the specific situation of software chrestomathies. We mention in passing that research is also needed on appropriate elements of gamification, thereby stimulating chrestomathy adoption and community contributions.

Integration with teaching. Chrestomathies should be useful for learning, by definition, but they may be of limited use, practically, when their underlying ontology and usage model is not aligned with the relevant profiles of learners. Research is needed to turn chrestomathies into effective teaching environments on the grounds of teaching concepts such as techniques of eLearning (e.g., [24]) or massive open online courses (MOOCs [25]). For example, this may involve capabilities such as course scope for discussion, content management for exams and homework as well as user-definable settings for prioritization of content and resources.

Polyglot analyses and transformations. We mentioned quality assurance and evolution as essential characteristics of software (if not program) chrestomathies. When a chrestomathy involves possibly many different programming languages, other software languages, and software technologies, then automated software analyses (for quality assurance) and automated software transformations (to support evolution) need to be polyglot in a scalable manner: adding yet another language (or technology) needs to be relatively simple. For instance, imagine a revision of a task of the chrestomathy, which may be as simple as renaming an involved function or type. Given the pervasive impact of such a revision, a semi-automatic, polyglot transformation would be needed in the interest of productivity. On top of the polyglot challenge, there is the need to deal with coupling or co-evolution as studied elsewhere in software engineering [26].

[4] `http://stackoverflow.com/` – accessed 25 Aug 2013.

[5] `http://msdn.microsoft.com/` – accessed 25 Aug 2013.

Similarity management. Another aspect of productivity of development and maintenance, also touching upon the area of knowledge management, concerns the similarity between systems, which is to be determined, to be evaluated, to be made transparent, and possibly to be preserved along evolution, or to be removed, when this is more appropriate. That is, the systems collected by a chrestomathy may be incidentally or intentionally similar in terms of file-system layout, source code, and other software artifacts also including metadata, and documentation. Learners benefit from understanding similarity between systems as they may focus in this manner on unique aspects when studying a similar system. Developers benefit from support for maintaining intentional similarity; without such management, the similar systems may diverge accidentally along loosely coupled evolution of individual systems. An approach to similarity management may leverage elements of clone detection, a reactive product line strategy [27], and version control, without though requiring learners and developers to intimately interact with such elements.

Knowledge management. Ideally, software chrestomathies end up as integrated, ontology-based, model-based knowledge bases—with knowledge about conceptual entities (e.g., software languages and technologies) and actual software artifacts (i.e., the systems of the chrestomathy). The *status quo* is to organize chrestomathies with the help of wiki systems, sometimes with some limited ontological idioms, e.g., for categories of pages, but without though proper ontologies. Research is needed to integrate ontology engineering [28] into chrestomathy development and maintenance. Further, documentation should also make use of appropriate modeling languages, e.g., megamodels for the linguistic architecture of collected systems and leveraged software technologies [29]. Also, documentation needs to be subjected to designated forms of quality assurance [30]. Finally, authors should also be supported in carrying out knowledge integration [31] within the scope of the chrestomathy while accessing diverse external resources such as textbooks, wikis, and alternative chrestomathies.

9. Epilogue

Depending on where we draw the line, there are dozens or hundreds of program and software chrestomathies in use and under more or less continuous development in the wild, with thousands of active contributors combined, with much potential for being useful to learn about and gain insight into programming and software engineering. Next-generation software chrestomathies are highly complex, highly structured, highly collaborative conglomerates of artifacts. Thus, research on software chrestomathies challenges software engineering and computer science in various respects. Advances on computer science teaching and knowledge management are likely to result from such research.

Acknowledgement. My understanding of the chrestomathy notion has enormously benefited from joint work with Jean-Marie Favre and Andrei Varanovich. Three anonymous reviewers provided insightful and helpful feedback on an earlier version of this paper; I addressed all concerns as good as I could.

References

[1] J.-M. Favre, R. Lämmel, T. Schmorleiz, A. Varanovich, *101companies*: a community project on software technologies and software languages, in: Proc. of TOOLS 2012, Vol. 7304 of LNCS, Springer, 2012, pp. 59–74.

[2] H. Mencken, A Mencken Chrestomathy, Knopf, 1949.

[3] Merriam-Webster Inc., Merriam-Webster's Collegiate Dictionary, Eleventh Edition (2011).

[4] Houghton Mifflin Harcourt, The American Heritage Dictionary of the English Language, Fourth Edition. Accessed via [32] (2009).

[5] Collins, Collins English Dictionary – Complete and Unabridged, Accessed via [32] (2003).

[6] K Dictionaries Ltd., Random House Kernerman Webster's College Dictionary, Accessed via [32] (2005).

[7] The Gale Group, Inc., Ologies & -Isms, Accessed via [32] (2008).

[8] The Gale Group, Inc., Dictionary of Collective Nouns and Group Terms, Accessed via [32] (2008).

[9] Farlex Inc., Farlex clipart collection, Based on WordNet 3.0. Accessed via [32] (2012).

[10] S. Mercer, Assyrian Grammar with Chrestomathy and Glossary, Luzac & Co., 1921.

[11] B. Layton, Coptic Gnostic Chrestomathy: A Selection of Coptic Texts with Grammatical Analysis and Glossary, Peeters Pub, 2004.

[12] R.-E. Brünnow, A. Fischer, L. Edzard, A. Bjorsnos, Chrestomathy of Classical Arabic Prose Literature, Harrassowitz, 2008.

[13] F. Ruehr, The Evolution of a Haskell Programmer, Website: `http://www.willamette.edu/~fruehr/haskell/evolution.html` — Accessed on 5 Aug 2013 (2001).

[14] O. Schade, G. Scheithauer, et al., 99 Bottles of Beer, Website: `http://www.99-bottles-of-beer.net` — Accessed on 5 Aug 2013 (2013).

[15] J. Weirich, OO example code, Website: `http://onestepback.org/articles/poly` — Accessed on 5 Aug 2013 (2013).

[16] Many contributors, Rosetta Code, Wiki: `http://rosettacode.org` — Accessed on 5 Aug 2013 (2013).

[17] G. Wilson, A. Oram, Beautiful Code – Leading Programmers Explain How They Think, O'Reilly Media, 2009.

[18] L. G. Cleophas, B. W. Watson, G. Zwaan, A new taxonomy of sublinear right-to-left scanning keyword pattern matching algorithms, Sci. Comput. Program. 75 (11) (2010) 1095–1112.

[19] Oracle Corporation, Java BluePrints – Guidelines, Patterns, and code for end-to-end Java applications, Website: `http://www.oracle.com/technetwork/java/index-jsp-136701.html` — Accessed on 5 Aug 2013. See also `http://en.wikipedia.org/wiki/Java_BluePrints` (2013).

[20] Many contributors, The Computer Language Benchmarks Game, Website: `http://benchmarksgame.alioth.debian.org` — Accessed on 5 Aug 2013 (2013).

[21] A. Rodriguez, J. Jeuring, P. Jansson, A. Gerdes, O. Kiselyov, B. C. d. S. Oliveira, Comparing libraries for generic programming in Haskell, in: Proc. of Haskell 2008, ACM, 2008, pp. 111–122.

[22] B. Alexe, W.-C. Tan, Y. Velegrakis, STBenchmark: towards a benchmark for mapping systems, Proc. VLDB Endow. 1 (2008) 230–244.

[23] G. Kazai, J. Kamps, N. Milic-Frayling, An analysis of human factors and label accuracy in crowdsourcing relevance judgments, Inf. Retr. (2013) 138–178.

[24] Y. Li, M. Dong, R. Huang, Designing Collaborative E-Learning Environments based upon Semantic Wiki: From Design Models to Application Scenarios, Educational Technology & Society 14 (4) (2011) 49–63.

[25] F. G. Martin, Will massive open online courses change how we teach?, Commun. ACM 55 (8) (2012) 26–28.

[26] D. D. Ruscio, L. Iovino, A. Pierantonio, A Methodological Approach for the Coupled Evolution of Metamodels and ATL Transformations, in: Proc. of ICMT 2013, Vol. 7909 of LNCS, Springer, 2013, pp. 60–75.

[27] T. Mende, F. Beckwermert, R. Koschke, G. Meier, Supporting the Grow-and-Prune Model in Software Product Lines Evolution Using Clone Detection, in: Proc. of CSMR 2008, IEEE, 2008, pp. 163–172.

[28] P. Spyns, Y. Tang, R. Meersman, An ontology engineering methodology for DOGMA, Applied Ontology 3 (1-2) (2008) 13–39.

[29] J.-M. Favre, R. Lämmel, A. Varanovich, Modeling the Linguistic Architecture of Software Products, in: Proc. of MODELS 2012, Vol. 7590 of LNCS, Springer, 2012, pp. 151–167.

[30] E. H. Weiss, Egoless writing: improving quality by replacing artistic impulse with engineering discipline, ACM Journal of Computer Documentation 26 (1) (2002) 3–10.

[31] P. N. Robillard, The role of knowledge in software development, Commun. ACM 42 (1) (1999) 87–92.

[32] Farlex Inc., The Free Dictionary, Website: `http://www.thefreedictionary.com/chrestomathy` — Accessed on 5 Aug 2013 (2013).