Guidelines for Collaboration

Zhiyang Ong *

December 1, 2020

Abstract

This is a set of guidelines for development/design processes and practices, and conduct while collaborating on open source projects. It also includes guidelines for creating a shared BibTeX database.

Contents

Revision History 2

1 Guidelines for Conduct 3

2 Guidelines for Creating a Shared BibTeX Database 3
- 2.1 Recommended Fields for BibTeX Entries . . . 6
- 2.2 Template Information for the "Annote" BibTeX Field . . . 8
- 2.3 Template Information for the "Howpublished" BibTeX Field . . . 8
- 2.4 Suggested Prefixes for Labels of Parts of a LaTeX Document . . . 9

3 Coding Standard 9

4 Exception Safety 11

5 Suggested Software Architecture 11

6 Adoption of Best Practices 12
- 6.1 Practice of Automated Regression Testing . . . 13
- 6.2 Cloud-based Data Science and Machine Learning . . . 13

7 Additional Guidelines 14

Acknowledgments 15

*Email correspondence to: ongz@acm.org

Revision History

Revision history:

1. Version 1, October 2, 2014. Initial version of the guideline (for another project).
2. Version 1.1, December 23, 2014. Version ported for this boilerplate code project.
3. Version 2, October 20, 2015. Added guidelines for `Doxygen`-supported, `Javadoc`-based coding standard. This coding standard is also known as coding style, coding style guide, coding guideline, coding scheme, code convention, code documentation guideline, programming guideline, or programming style.
4. Version 2.1, October 21, 2015. Finished guidelines for `Doxygen`-supported, `Javadoc`-based coding standard for *C++*.
5. Version 2.2, June 4, 2016. Finished section for additional guidelines: to include documentation using *Markdown*, and tools for software development, integrated circuit and cyber-physical system design, and documentation.
6. Version 3, November 3, 2016. Added guidelines for: documenting `GNU Octave` and MATLAB code, in order to facilitate documentation generation using *Texinfo* [315–317,356]; sharing of source code, design files, sets of benchmarks, data sets, and documentation on online repositories [99,120]; and added section on exception safety.
7. Version 3.1, November 4, 2016. Fixed references for indent style conventions.
8. Version 3.2, December 20, 2016. Update guidelines for conduct.
9. Version 3.3, February 3, 2017. Update information about usage of *GitHub*'s services.
10. Version 3.4, March 11, 2017. Update information on naming convention.
11. Version 3.5, October 9, 2017. Update guidelines on commenting/writing code.
12. Version 3.6, December 24, 2017. Fix grammatical error in a sentence.
13. Version 3.7, January 25, 2018. Added suggestions for software architecture of my computer programs.
14. Version 3.8, January 31, 2018. Added information about coding style guideline for different computer languages, and also about online repositories that facilitate research reproducibility, replicability, and repeatability.
15. Version 3.9, February 1, 2018. Added information about developing software in a *Python*ic style.
16. Version 4.0, June 8, 2018. Added information about specifying (co-)authors' full name, and research reproducibility, and other best practices from software development, and embedded/cyber-physical system and integrated circuit design.
17. Version 4.1, September 19, 2018. Updated ACM Code of Ethics and Professional Conduct; added The Joint ACM/IEEE-CS Software Engineering Code of Ethics and Professional Practice; and updated guidelines on exception handling.
18. Version 4.2, September 21, 2018. Added acknowledgements, shout outs, for people who helped me with automated regression testing. And, refactored document.
19. Version 4.3, September 22-25, 2018. Added references on agile SoC design, and hardware/VLSI/RTL/HDL refactoring.
20. Version 4.4, December 29-30, 2018. Added template information for the "Annote" and "Howpublished" BibTeX fields to help me identify particular information about the document/publication.
21. Version 4.5, February 11, 2019. Extend recommendations and suggestions to projects in data science and (applied) machine learning.
22. Version 4.6, February 14, 2019. Add references for *Git* and its substitute, *Mercurial SCM*, for distributed version/revision control (or software configuration management). Also, add "Acknowledgments" and "Revision History" sections to the "Table of Contents".

23. Version 4.7, March 13, 2019. Add comment on how to include mathematical expressions in *Markdown* documents. In addition, fix minor errors.
24. Version 4.8, January 22, 2020. Fix BibTeX key error (renamed for a publication), and added references regarding design by contract (or programming by contract, or contract programming) and Hoare logic.

1 Guidelines for Conduct

Collaborators of open source software and/or hardware projects that we are involved in should follow the *Code of Conduct* of the *Institute of Electrical and Electronics Engineers* (IEEE) [152–154] and the *ACM Code of Ethics and Professional Practice* of the *Association for Computing Machinery* (ACM) [5,18,45–47,129–131,363], including the "The Joint ACM/IEEE-CS Software Engineering Code of Ethics and Professional Practice" [132,133]. Also, actions of discrimination are not acceptable [155]; we should intentionally commit to inclusive diversity. An additional guideline is "Dave Packard's 11 simple rules" [66].

In addition, when there is a dispute about which technology, algorithm, design paradigm/style/pattern, process, or methodology to use, follow the "Code Wins Arguments" philosophy [186,369]. Also, when considerable effort has been invested in an automated regression testing/verification infrastructure, do not be afraid to "move fast and break things" [91,101].

Lastly, we should adopt a mission-focused and value-based approach to participate in meetings and discussions for the project(s). We should be flexible/liberal enough to consider and explore viable alternate approaches to do things and solve problems [31,32]. Where disputes occur, a data-driven, fact-based approach based on the "Code Wins Arguments" philosophy should be used to resolve conflicts.

2 Guidelines for Creating a Shared BibTeX Database

Guidelines for creating BibTeX entries and the BibTeX database, which is used for writing the paper, are given as follows:

1. Each BibTeX key should be unique:
   (a) Check if your desired BibTeX key already exists in the BibTeX database:
      i. If it does, do not add it to the BibTeX database.
      ii. Else, add it to the BibTeX database.
   (b) Use the following format for creating BibTeX keys: [first] author's last name, appended by the year of publication. E.g., my first conference paper would have the BibTeX key Ong2014. If the year of publication is not known, use an approximate year, with XY for the last 2 digits in the year (e.g., 20XY). Alternatively, if you cannot determine if it was published this millennium or the previous millennium, use 20XY (or UNKNOWN). For example, use Smith20XY (preferred), or KleinbergUNKNOWN.
   (c) Remove duplicate entries in the BibTeX database. **WARNING! Before doing this, perform a union operation on the fields of the BibTeX entries. For example, if a BibTeX entry has information that the other BibTeX entry does not have, and vice versa, merge the information to a BibTeX entry.**
   (d) **Rationale: Duplicate BibTeX entries will cause problems in typesetting.**

   (e) Regarding hash collision of BibTeX keys, such as multiple instances of Gratz2014, distinguish them by appending a letter to them. E.g., use Gratz2014, Gratz2014a, Gratz2014b, Gratz2014c, and so on. If we run out of letters, append it with "a" followed by a number. The use of the letter "a" separates the year from the instance of BibTeX key. That is, Gratz2014a2 tells me that it is the 29^{th} instance of Gratz2014, as opposed to Gratz201429.
   (f) If possible, restrict the characters of each BibTeX key to be alphanumeric. The year is always numeric, and is appended to the (first) author's last name.
      i. If the (first) author's last name has characters with diacritical marks, accents, or diacritics, trim the characters used to typeset the diacritical marks (or accents) from the (first) author's last name, and append the year of publication to it. E.g., *Sŏménzi* (year 2000) becomes *Somenzi2000*.
      ii. If the (first) author's last name has characters that are not letters in English, anglicize those characters. We should avoid using the transliteration for a given non-English language, since such transliteration may not be standardized (for non-commonly spoken/used languages). Also, supporting letters from other languages is a tedious task. Hence, we can use the anglicized version of their last names instead.

2. If possible, use the full name for each author. When writing research publications, if we need to reduce the authors' first name to just their initial, we can use a script to transform their names. If we need to use their full names in the reference list and if we do not include their full names, we have to look up these references again in the future to include their full names.
3. For terms that should be typeset as is, place them in between braces (i.e., curly brackets). That is, put curly braces around acronyms and mixed-case names.
   (a) For example, terms in upper or mixed cases (upper and lower cases), such as names (e.g., McMullen) and acronyms (e.g., SIGDA), place them in between braces (i.e., {McMullen} and {SIGDA}). This prevents the titles (or another BibTeX field) from changing the term into lower case, with exception for the first term/word. E.g., "ICCAD Update: A Report from SIGDA" may typeset into "ICCAD Update: A report from sigda".
4. For special symbols that are typeset with LaTeX in the `math mode`, such as α, place them in between a pair of dollar signs (i.e., α).
5. For each BibTeX entry, check if all required fields are complete. See pages 8 and 9 in §3.1 of [258] for a list of BibTeX entry types; alternatively, refer to the *Wikipedia* entry for , or [177, §12.2.1, pp. 230–231]. In this/these list(s), the required fields are listed for each BibTeX entry.
6. For the `pages` field, ensure that all page ranges are indicated with double hyphens. E.g., "page = {11--34},". This makes the page range looks more pretty.
7. For the `pages` field, ensure that multiple pages and/or page ranges are separated by commas. E.g., "page = {11-34, 57, 88, 109--187},".
8. For books and journal articles that have an associated digital object identifier (DOI) [157], ensure that the `doi` field is included in the BibTeX entry with the DOI of the publication. This makes it easier for people to access the web page for the book or journal/conference paper.
9. Stylistic validation of the references can be carried out as follows:
   (a) Include all BibTeX keys in one citation in your LaTeX document.
   (b) Typeset the LaTeX document.
   (c) Check that the font and style of the reference list is correct.
   (d) If there are errors, correct the errors as appropriate.
   (e) Finally, the BibTeX database should be correct.

10. Information that I would include when citing common sources of information, such as *Wikipedia*, using the Harvard Referencing Style:
    (a) Wikipedia contributors, "TITLE_OF_THE_ARTICLE," in {\it Wikipedia, The Free Encyclopedia: CATEGORY}, Wikimedia Foundation, San Francisco, CA, MONTH DATE, YEAR. Available online at: \url{URL}; last accessed on August 26, 2014.
    (b) Wikibooks contributors, "CHAPTER_NAME," in {\it TITLE_OF_THE_BOOK}, Wikibooks: Open books for an open world, Wikimedia Foundation, San Francisco, CA, MONTH DATE, YEAR. Available online at: \url{URL}; last accessed on August 26, 2014.
    (c) Wikibooks contributors, "SECTION," in {\it CHAPTER} of {\it TITLE OF THE BOOK}, Wikibooks: Open books for an open world, Wikimedia Foundation, San Francisco, CA, MONTH DATE, YEAR. Available online at: \url{URL}; last accessed on August 26, 2014.
    (d) Wikibooks contributors, "TITLE_OF_THE_BOOK," Wikibooks: Open books for an open world, Wikimedia Foundation, San Francisco, CA, MONTH DATE, YEAR. Available online at: \url{URL}; last accessed on August 26, 2014.
    (e) Wikiquote contributors, "TITLE," Wikiquote, Wikimedia Foundation, San Francisco, CA, MONTH DATE, YEAR. Available online at: \url{URL}; last accessed on August 26, 2014.
    (f) Wiktionary contributors, "TITLE," Wiktionary, Wikimedia Foundation, San Francisco, CA, MONTH DATE, YEAR. Available online at: \url{URL}; last accessed on August 26, 2014.
    (g) Dictionary.com, "WORD," IAC, Oakland, CA, MONTH DATE, YEAR. Available online at: \url{URL}; last accessed on August 26, 2014.
    (h) AUTHOR, "TITLE," in {\it The New York Times: The Opinion Pages: Op-Ed Contributor}, The New York Times Company, New York, NY, MONTH DATE, YEAR. Available online at: \url{URL}; last accessed on August 26, 2014.
    (i) AUTHOR, "QUESTION", in {\it CATEGORY}, Quora, Inc., Mountain View, CA, MONTH DATE, YEAR. Available online at: \url{URL}; last accessed on August 26, 2014.
    (j) AUTHOR, Answer to "QUESTION", in {\it CATEGORY: QUESTION}, Quora, Inc., Mountain View, CA, MONTH DATE, YEAR. Available online at: \url{URL}; last accessed on August 26, 2014.
    (k) AUTHOR, "TITLE OF POST", in {\it BLOG TITLE}, Quora, Inc., Mountain View, CA, MONTH DATE, YEAR. Available online at: \url{URL}; last accessed on August 26, 2014.
    (l) AUTHOR, "TITLE," Stack Exchange Inc., New York, NY, MONTH DAY, YEAR. Available online from {\it Stack Exchange Inc.: Stack Overflow: Questions} at: \url{URL}; March 16, 2016 was the last accessed date.
    (m) AUTHOR, "TITLE OF REPOSITORY," GitHub, Inc., San Francisco, CA, MONTH DAY, YEAR. Available online from {\it {GitHub: GitHub USERNAME (or NAME OF ORGANIZATION)}: at: \url{URL}; March 16, 2016 was the last accessed date.
    (n) AUTHOR, "TITLE OF PAPER," Cornell University, Ithaca, NY, MONTH DAY, YEAR. Available online (as Version XYZ) from {\it {arXiv: FIELD(s)}: at: \url{URL}; March 16, 2016 was the last accessed date.
    (o) When BibTeX entries are created for the aforementioned sources of information, populate the appropriate fields so that each information in the aforementioned sources are included in the BibTeX entries.
    (p) For other organizations, communities, and groups, use the term "contributors" instead of "members," unless otherwise specified.

11. Refer to the file "bibtex-template.txt" for templates for selected BibTeX entry types. The more information that you can put in, the easier you can protect yourself from accusations of plagiarism and to make it easier for people (including yourself) to find the reference again. This is especially true for web-based references/resources.

2.1 Recommended Fields for BibTeX Entries

The recommended fields for BibTeX entries are:

1. techreport:
   (a) Address
   (b) Author
   (c) Howpublished
   (d) Institution
   (e) Keywords
   (f) Month
   (g) Number
   (h) Title
   (i) Url
   (j) Year
2. proceedings:
   (a) Address
   (b) DOI
   (c) Editor
   (d) Keywords
   (e) Month
   (f) Organization
   (g) Publisher
   (h) Series
   (i) Title
   (j) Volume
   (k) Year
3. manual:
   (a) Address
   (b) Author
   (c) Howpublished
   (d) Keywords
   (e) Month
   (f) Organization
   (g) Title
   (h) Url
   (i) Year
4. incollection:
   (a) Address
   (b) Author
   (c) Booktitle
   (d) Chapter
   (e) DOI
   (f) Edition
   (g) Howpublished
   (h) Keywords
   (i) Pages

   (j) Publisher
   (k) Series
   (l) Title
   (m) Url
   (n) Volume
   (o) Year

5. inproceedings:
   (a) Address
   (b) Author
   (c) Booktitle
   (d) DOI
   (e) Keywords
   (f) Month
   (g) Organization
   (h) Pages
   (i) Publisher
   (j) Series
   (k) Title
   (l) Volume
   (m) Year

6. article:
   (a) Address
   (b) Author
   (c) DOI
   (d) Journal
   (e) Keywords
   (f) Month
   (g) Number
   (h) Pages
   (i) Publisher
   (j) Title
   (k) Volume
   (l) Year

7. phdthesis (or mastersthesis):
   (a) Address
   (b) Author
   (c) Howpublished
   (d) Keywords
   (e) Month
   (f) Number
   (g) School
   (h) Title
   (i) Url
   (j) Year

8. misc:

   (a) Address
   (b) Author
   (c) Howpublished
   (d) Keywords
   (e) Month
   (f) Publisher or School
   (g) Title
   (h) Url
   (i) Year
9. book:
   (a) Address
   (b) Author
   (c) DOI
   (d) Edition
   (e) Keywords
   (f) Month
   (g) Pages
   (h) Publisher
   (i) Series
   (j) Title
   (k) Volume
   (l) Year

2.2 Template Information for the "Annote" BibTeX Field

This is a list of template information for the "Annote" BibTeX field to help me identify particular information about the document/publication:

1. When citing from this entry, shift the names of the authors from the author field to the editor field.
2. Editors: BLAH ... BLAH ... BLAH
3. Originally published in: 20XY
4. Also, published in: 20XY
5. Originally published by BLAH in BLAH–BLAH
6. Also, published by BLAH in BLAH–BLAH
7. Also, published as BLAH (old series title and old volume number)
8. Reprinted in 20XY by BLAH
9. Copyright renewed in 20XY.
10. Authors' names (alternate): blah, blah blah, blah blah blah, ...
11. Received a copy of this *report/document/publication* by email.
12. Alternate title: *Another title for the document/publication.*

2.3 Template Information for the "Howpublished" BibTeX Field

For the "Howpublished" BibTeX field, the suggested template information is:

- Available online at: \url{}; self-published; MONTH DAY, YEAR was the last accessed date

When the date is not known so that it can be provided in the "MONTH DAY, YEAR" format, use the following phrase instead of "MONTH DAY, YEAR was the last accessed date".

- the last accessed date is unknown

2.4 Suggested Prefixes for Labels of Parts of a LaTeX Document

Add one of the following prefixes to labels in my LaTeX documents for parts, such as chapters, sections, subsections, subsubsections, figures, tables, equations, code listings, definitions, theorems, lemmas, corollaries, propositions, proofs, examples, and remarks:

1. "chp:" for chapter
2. "sec:" for section
3. "ssec:" for subsection
4. "sssec:" for subsubsection
5. "fig:" for figure
6. "tab:" for table
7. "eqn:" for equation
8. "lst:" for code listing
9. "defn:" for definition
10. "thrm:" for theorem
11. "lem:" for lemma
12. "crly:" for corollary
13. "prop:" for proposition
14. "prf:" for proof
15. "eg:" for example
16. "rem:" for remark

This practice makes it easier for collaborators and paper/reviewers to determine if the label for a part of the LaTeX document, belongs to a label or refers to something else.

3 Coding Standard

This is a guideline for `Doxygen`-supported [340], `Javadoc`-based [205] coding standard that shall be used for this boilerplate code project and other projects. The term "coding standard" is used interchangeably/synonymously with coding style, coding style guide, coding guideline, coding scheme, code convention, code documentation guideline, programming guideline, or programming style. Our coding style/standard shall be self-documenting. The documentation generator that shall be supported is: `Doxygen`. Since we are using `Doxygen` for generating documentation, we can use LaTeX to provide richer markup.

Document the known bugs for each function/method.

Our indent style would be the *1TBS* variant of the *K&R* style, which is an abbreviation of "*The One True Brace Style*". It is also equivalent to the *Kernel Normal Form style* (or *BSD KNF style*) [353].

Classes, functions/methods, constants, macros, and static and instance variables shall be named using complete words or well-known abbreviations that are concatenated with an underscore in *C++*; this is a deviation from the *Hungarian notation* that uses an upper case letter to distinguish words/abbreviations in the name (i.e., the *Start case style of writing*; see letter case). That is, the naming convention followed is using multiple-word identifiers, via delimiter-separated words rather than letter-case separated words (e.g., *Hungarian notation*) [357].

For *C++* programs, the following tags shall be used in the comments:

1. @author *Author's_Name*: indicate the author (*Author's_Name*) of the file/function
   (a) @modified by NAME, DATE in "Month Day, Year" format.
2. @version *X.Y*: indicate the version (*X.Y*) of the file
3. @section *SECTION_NAME*: indicate the section (*SECTION_NAME*) of the file, which can be: *LICENSE* or *DESCRIPTION*
4. @param *x*: indicate the parameter (*x*) of the constructor or function
5. @exception *Exception_Name*, or @throws *Exception_Name*: an exception that a function/method can throw
6. @return *Return_Statement*: indicate the return (type and) action of the function
7. @see *reference*: a link to another element in the documentation; e.g., @see *Class_Name*, or @see *Class_Name#member_function_name*
8. @since *X.Y: Month-Day-Year*: This functionality has been added since version *X.Y* (and on the date *Month-Day-Year*)
9. @deprecated *description*: Describe an outdated function/method, and indicate when the function/method has deprecated
10. "@link ... *URL...* @endlink" is used to include hyperlinks in the generated documentation for Doxygen
11. #### IMPORTANT NOTES: Notes that are critical for helping the reader understanding assumptions and decisions made while developing the software
12. @todo(<message>, <version>) (or #### TO BE COMPLETED): Task to be finished at a later time. If it is busywork (or, busy work), indicate that it is busywork.
13. #### TO BE FIXED: Task to be debugged at a later time
14. @migration(<message>, <version>): Code is being migrated to another function/method, or class.
15. See http://www.stack.nl/~dimitri/doxygen/commands.html for more information of tags that are recognized by `Doxygen`.
16. @pre (or @precondition): Precondition(s) of the function.
17. @assert (or @assertion): Assertion(s) of the function.
18. @post (or @postcondition): Postcondition(s) of the function.

The order of tags in different sections of the *C++* code is given as follows:

1. Headers/Interfaces and Classes: @version, @author, @since, @link, @todo, @deprecated, @migration, and @see
2. Constructors: @param, @throws, @since, @link, @todo, @deprecated, @migration, and @see. For collaborators modifying or extending my code, they should include the @version and @author tags before the @param tag(s).
3. Functions/Methods: @param, @pre, @assert, @post, @return, @throws, @since, @link, @todo, @deprecated, @migration, and @see. For collaborators modifying or extending my code, they should include the @version and @author tags before the @param tag(s).
4. Variables can use the @see tags.
5. The @deprecated tag can be used for headers/interfaces, classes, constructors, functions/methods, and variables.

Additional coding style guidelines can be found in [28, 56, 71, 146, 231, 272, 306, 331].

For a suggested coding style for *Python* and *Ruby* scripts, see [252, 341] and [208], respectively. Regarding coding style guidelines for embedded *C*, see [27, 188]. In addition, there exists coding style

guidelines for *Java* [57, 205, 245–248, 311] and *LabVIEW* [41, 68]. Coding style guidelines for *Verilog* can be found at: [35, 36]. Likewise, the coding style guide for *SystemVerilog* can be found at [230]. For other coding style guidelines, see [54, 95, 100, 156, 171, 176, 191, 218, 301, 304, 309, 339, 346, 362]. *Google* style guides [128] has provided documentation about best practices [254] for coding standards and the philosophy [253] of Google's coding standards.

While well-documented source code is desired, natural language programming [354] is usually infeasible due to the choices of programming/computer languages used. Also, while literate programming [174, 175, 218, 238, 250, 301, 324] is encouraged, we are currently not following it due to the tedious process of developing software using literate programming. Hence, a short development time for well-commented, functionally correct, and efficient source code is prioritized over code written according to the literate programming approach.

4 Exception Safety

When developing software using programming/scripting languages that enable exceptions or errors to be thrown and caught, adopt "a set of contractual guidelines" [352] to support exception/error management. This "set of contractual guidelines" is based on exception safety guarantees in *C++* [3, 4, 352] [351, Subsection §4.4 on "Writing exception safe code"].

The levels of exception/error safety listed in descending order of safety guarantees are [3, 4, 351, 352]:

1. no throw guarantee, or failure transparency: "Best level of exception safety."
2. strong exception safety, commit/rollback semantics, or no-change guarantee
3. basic exception safety
4. minimal exception safety, no-leak guarantee
5. no exception safety: "No guarantees are made. (Worst level of exception safety)"

These aforementioned levels of exception/error safety can be partially handled. Also, the use of guards is strongly recommended for making the software and library (or, circuit or system) exception safe.

These guidelines about exceptions help software developers know what to do about fatal exceptions, boneheaded exceptions, vexing exceptions (due to unfortunate design decisions). Vexing exceptions and boneheaded exceptions, to a lesser extent, are preventable exceptions [204]. Hence, we should develop software that avoids triggering preventable exceptions.

Please judiciously consider what to do with the semipredicate problem [355].

5 Suggested Software Architecture

At the software system level, the software architecture can be described by the following modules/components:

1. parser(s):
   (a) For input benchmarks
2. utilities:

   (a) output generator(s)
   (b) flag/switch -based printing information to standard output/error:
      i. Print statements only when debugging mode is on.
      ii. Else, squelch print/trace statements to speed up computation/performance.
3. solvers:
   (a) ODE solver(s) for ordinary differential equations (ODEs):
      i. ODE solver(s) for nonlinear ODEs.
   (b) PDE solver(s) for partial differential equations (PDEs):
      i. PDE solver(s) for nonlinear PDEs.
   (c) satisfiability modulo theories (SMT) solver(s)
   (d) boolean/proposition satisfiability (SAT) solver(s)
   (e) maximum satisfiability modulo theories (Max-SMT) solver(s)
   (f) maximum satisfiability (Max-SAT) solver(s)
   (g) pseudo-boolean optimization (PBO) solver(s)
   (h) quadratic unconstrained binary optimization (QUBO) solver(s)
   (i) weighted boolean optimization (WBO) solver(s)
   (j) framework for algorithmic portfolio optimization
4. data structures:
   (a) directed graphs:
      i. directed acyclic graphs (DAGs)
      ii. binary decision diagrams (BDDs)
      iii. AND-inverter graphs (AIGs)
   (b) undirected graphs:
      i. heaps
      ii. trees
   (c) maps, dictionaries, and hash tables
5. graphical user interface (GUI), if required.

Lastly, suggestions are not available for digital and mixed-signal integrated circuits (ICs) and VLSI systems, such as system-on-chips (SoCs). More work needs to be done in terms of looking at hardware refactoring, and hardware design patterns.

6 Adoption of Best Practices

Where possible, we shall try to adopt multiple best practices from leading product teams (i.e., R&D teams) in the semiconductor and IT industries, and also good researchers spanning electrical engineering and computer science. These practices include: research reproducibility and reproducible research [26, 29, 40, 64, 73, 76, 109–111, 173, 181, 203, 210, 290, 303, 323], build automation [65, 74, 88, 100, 136, 147, 166, 249, 277, 284, 300–302, 325], distributed version/revision control (or software configuration management) [10, 20, 54, 55, 58, 59, 65, 74, 89, 103, 116–119, 121–123, 138, 147, 216, 217, 250, 255, 256, 270, 277, 299, 301, 309, 327, 329, 350], design by contract (DbC; or, contract programming, programming by contract, or design-by-contract programming) [49, 137, 151, 228, 294, 321, 328, 365] when using the procedural/imperative programming paradigm [23, 33, 199, 206, 211, 233, 259, 282, 308, 334, 345, 358] and Hoare logic [25, 139, 151, 178, 185, 190, 232, 236, 260, 294, 305, 361, 365, 368], regression testing [9, 273, 300, 302, 342], automated software testing [37, 54, 72, 75, 97, 147, 158, 170, 187, 229, 237, 244, 257, 273, 274, 285, 292, 300, 302, 312, 321, 324, 336, 347, 365], and automated regression testing [273, 283]. An analogy of regression testing

for software for VLSI design is regression verification [6,7,9,23,43,51,124,202,262–266,310,332]. Similarly, endeavor to use the concepts of abstraction [61, 67, 106, 115, 140, 143, 150, 183, 183, 198, 219, 318, 343, 344] and encapsulation [25, 48, 68, 96, 143, 182, 199, 211, 220, 267, 277, 279, 307, 313] with hierarchical design methodologies [8, 53, 100, 135, 179, 180, 242, 291, 337], hierarchical design space exploration [326], top-down hierarchical approach for design steps [62, 85, 105, 108, 145, 163, 207, 295, 348] and bottom-up hierarchical approach for verification steps [61, 105, 219, 275, 344], or rather top-down approach of incremental verification (also known as the modified V approach) [23], and platform-based design [23, 24, 52, 60, 81, 108, 172, 209, 212, 293, 295–298] in our projects involving VLSI design.

In addition, try to use agile (software development and VLSI design) methodologies [1, 12, 14, 15, 19, 22, 30–32, 42, 54, 63, 65, 69, 70, 74, 77, 79, 83, 90, 103, 114, 126, 149, 159, 160, 167, 168, 190, 192–197, 200, 213–215, 243, 261, 271, 273, 276, 286–289, 301, 309, 319, 322, 324, 338, 364] to develop software as well as design electronic circuits and systems [39, 125, 144, 161, 221, 360] [113, Chapter 6, §6.2.2.3, pp. 243], and cyber-physical systems (or embedded systems) [234, 235]. A strong motivation for using these methodologies and their associated practices is to reduce technical debt [1, 44, 65, 134, 184, 239, 278, 286, 320, 333].

Also, carry out refactoring [98, 102] on an ad-hoc basis to improve the software [1, 22, 34, 50, 74, 84, 86, 92, 95, 103, 159, 169, 176, 218, 227, 309, 314, 321], hardware [93, 94, 141–143, 164, 165, 359, 366, 367], and/or system [65, 189, 286] architecture as well as databases [16]. In terms of personal and professional development, collaborators are strongly encouraged to refactor their wetware [148] and reduce their personal technical debt [2, 78, 107, 201, 268], too.

We shall also use project portfolio management [287, 288] to help us manage projects that we are involved in.

Moreover, for projects involving integrated circuit design (and embedded hardware or cyber-physical system design), morph your process for VLSI CAD engineering into hardware DevOps (hardware/IC development and information-technology operations) [240].

6.1 Practice of Automated Regression Testing

Regarding the practice of automated regression testing, Mr. Heiko Maurer (then a lecturer at the University of Adelaide) and Dr./Mr. Tishampati Dhar (a former classmate at the University of Adelaide) suggests printing information regarding passed test cases to a file (or to standard output) and printing information regarding failed test cases to another file (or to standard error). During build automation of software, such as `gem5` [38, 112], carry out automated (regression) testing during the last stage of the build/installation process to ensure that the build/installation was done correctly. When performing automated software testing (or software test automation), list the the test cases and their test results (i.e., "OK"/"Fail"), just like `gem5` during the testing phase of build automation. At the end of each automated software testing run (or round/run of automated software testing), indicate the total number of test cases used, the total number of test cases passed, and the percentage of test cases passed (with respect to the total number of test cases used).

[330, §Testing Guidelines] provides a set of testing guidelines for *Python* libraries and packages, or software in general.

6.2 Cloud-based Data Science and Machine Learning

Regarding data science projects, we shall use a lot of the aforementioned software development practices and methodologies [54]. Also, we can use cloud-based machine learning (and deep learning) platforms, such as Google's *Colaboratory* [127], Anaconda's *Anaconda Cloud* [17], and Amazon's *Amazon Web Services* (AWS) [13]. These cloud-based software services, also known as software as a service (SaaS), helps to bring data science, machine learning, and deep learning capabilities to more people, since they do not need expensive, modern hardware to run computationally intensive tasks for data analytics and machine learning.

7 Additional Guidelines

Please kindly use the *Markdown* language for writing text documents. This is because *Bitbucket* will treat my text file as a file written in the *Markdown* syntax. That said, the raw file looks a lot better than the represented *Markdown* files. Their (*Bitbucket*) formatting for *Markdown* is messed up. *GitHub*'s formatting for *Markdown* works as expected. To insert mathematical expressions into *Markdown* documents, use *TexPaste* [241] to typeset the mathematical expressions via LaTeX and insert snapshots of these mathematical expressions as pictures in the *Markdown* documents. A recommended style guide for *Markdown* is from *Google* [335].

In addition, tools for working with source code and LaTeX source files include:

1. `git` [89, 138]:
   (a) `Mercurial SCM` [21, 80, 222–226, 255] can be used as a substitute.
2. `latexdiff`: "determine and markup differences between two latex files"
   (a) Evan Driscoll, "Latexdiff notes," from *Evan Driscoll's web page: Writings on Software: LaTeX*, the Department of Computer Sciences, University of Wisconsin-Madison College of Engineering, University of Wisconsin-Madison, Madison, WI. Available online at: `http://pages.cs.wisc.edu/~driscoll/software/latex/latexdiff.html`; last accessed on February 15, 2016 [87].
3. documentation generators:
   (a) `Doxygen` [340]
   (b) `Texinfo`-based generators [315–317, 356]
4. Build automation:
   (a) `SCons` [88]

Data sets and sets of benchmarks for experiments shall be publicly published using an online repository, via *figshare LLP* [99] and/or *DataHub* [76]. For each data set, or each set of benchmarks, create a unique Digital Object Identifier (DOI) [157] to identify it.

Repositories for software as well as designs of integrated circuits and cyber-physical systems shall be stored online, using online repositories such as *GitHub* [120]. Each repository shall have a unique DOI to identify it, and include all source code, documentation, and design files. There also exists cloud-based repositories for the source code of software/hardware projects that allow me to execute my software (or simulate my hardware). E.g., see [64, 290] as examples to facilitate research reproducibility, replicability, and repeatability. This supports research reproducibility and reproducible research [26, 29, 40, 64, 73, 76, 109–111, 173, 181, 203, 210, 290, 303, 323].

Please kindly note that *GitHub* [120]:

1. Does not allow a *GitHub*-based page to be refreshed/reloaded many times in a few seconds. Else, it would report the following:
   (a) "Whoa there!"
   (b) "You have triggered an abuse detection mechanism."
   (c) "Please wait a few minutes before you try again."

If possible, develop software in a *Python*ic style [11, Chapter 1, pp. 1–12,12–17] [104, 162, 269, 280, 281, 341].

Acknowledgments

Mr. David Knight (then a lecturer at the University of Adelaide) and Dr. Charles Lakos (then a senior lecturer at the University of Adelaide) introduced me to regression testing and automated software testing during their introductory course on software engineering. During programming assignments and projects for this course, Dr. Nikolay Stoimenov helped me honed my skills in regression testing and automated software testing, via the practice of pair programming [82, 103, 158, 251, 309, 349]. Subsequently, Mr. Heiko Maurer (then a lecturer at the University of Adelaide) planted the seeds of automated regression testing with his suggestion of separating the results of test cases that passed from the results of test cases that failed. Shortly after, Dr./Mr. Tishampati Dhar (a former classmate at the University of Adelaide) suggests printing information regarding passed test cases to a file (or to standard output) and printing information regarding failed test cases to another file (or to standard error). Months later, Dr. Francis Vaughan (then a senior lecturer at the University of Adelaide), Mr. Kevin J. Maciunas (then a lecturer at the University of Adelaide), and Dr. Robert Esser (then a senior lecturer at the University of Adelaide) helped me develop a sound methodology towards automated regression testing. In addition, Dr. Lakos and Dr. Esser introduced me to using formal methods and software formal verification in the software development process.

References

[1] 37signals staff. Getting Real: The smarter, faster, easier way to build a successful web application. 37signals, Chicago, IL, 2006.

[2] Sara Aboulafia. 5 ways to kill your credibility as a product manager. Available online from *UserVoice: UserVoice Blog, The Product Management Blog* at: https://community.uservoice.com/blog/5-ways-to-kill-your-credibility-as-a-product-manager/; September 26, 2018 was the last accessed date, July 14 2016.

[3] David Abrahams. Exception-safety in generic components: Lessons learned from specifying exception-safety for the C++ standard library. In Proceedings of the International Seminar on Generic Programming, volume 1766 of Lecture Notes in Computer Science, pages 69–79, Wadern, Merzig-Wadern, Saarland, Germany, April 27 – May 1 1998. Springer-Verlag Berlin Heidelberg.

[4] David Abrahams. Exception-safety in generic components: Lessons learned from specifying exception-safety for the C++ standard library. Available online from *Boost C++ Libraries: Community* at: http://www.boost.org/community/exception_safety.html; self-published; October 27, 2016 was the last accessed date, 2001.

[5] ACM Council. ACM code of ethics and professional conduct. Available online at: http://www.acm.org/about/code-of-ethics; September 27, 2014 was the last accessed date, October 16 1992.

[6] John Adler. Novel approaches to automated digital design debugging in a modern development cycle. Master's thesis, University of Toronto, Toronto, Canada, November 2017.

[7] John Adler and Andreas Veneris. Leveraging software configuration management in automated RTL design debug. IEEE Design & Test, 34(5):47–53, September/October 2017.

[8] Saurabh N. Adya. Unification of VLSI placement and floorplanning. PhD thesis, University of Michigan, Ann Arbor, MI, 2004.

[9] David J. Agans. Debugging: The 9 Indispensable Rules for Finding Even the Most Elusive Software and Hardware Problems. AMACOM, New York, NY, 2006.

[10] Carl Albing, J. P. Vossen, and Cameron Newham. bash Cookbook: Solutions and Examples for bash Users. O'Reilly Media, Sebastopol, CA, 2007.

[11] Marty Alchin. Pro Python: Advanced Coding Techniques and Tools. The Expert's Voice$^{\text{o}}$ in Open Source. Apress, Berkeley, CA, 2010.

[12] Rachel Alt-Simmons. Agile by Design: An Implementation Guide to Analytic Lifecycle Management. Wiley & SAS Business. John Wiley & Sons, Hoboken, NJ, 2016.

[13] Amazon Web Services staff. Amazon Web Services (AWS): Cloud computing services. Available online from *Amazon* at: https://aws.amazon.com/; February 11, 2019 was the last accessed date, 2019.

[14] Scott W. Ambler. Agile Database Techniques: Effective Strategies for the Agile Software Developer. Wiley Publishing, Indianapolis, IN, 2003.

[15] Scott W. Ambler and Mark Lines. Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise. IBM Press and Pearson PLC, Armonk, NY and Boston, MA, 2012.

[16] Scott W. Ambler and Pramod J. Sadalage. Refactoring Databases: Evolutionary Database Design. Addison-Wesley Signature Series. Addison-Wesley, Boston, MA, 2006.

[17] Anaconda staff. Anaconda cloud. Available online at: https://anaconda.org/; February 11, 2019 was the last accessed date, 2019.

[18] Ronald E. Anderson, Deborah G. Johnson, Donald Gotterbarn, and Judith Perrolle. Using the new ACM code of ethics in decision making. Communications of the ACM, 36(2):98–107, February 1993.

[19] Jurgen Appelo. Management 3.0: Leading Agile Developers, Developing Agile Leaders. Addison-Wesley Signature Series. Addison-Wesley, Boston, MA, 2011.

[20] Atlassian staff. Tutorial: Learn Bitbucket with Git. Available online from *Atlassian: Atlassian Documentation: Bitbucket Cloud documentation: Tutorials* at: https://confluence.atlassian.com/bitbucket/tutorial-learn-bitbucket-with-git-759857287.html; February 15, 2019 was the last accessed date, October 24 2018.

[21] Atlassian staff. Tutorial: Learn Bitbucket with Mercurial. Available online from *Atlassian: Atlassian Documentation: Bitbucket Cloud documentation: Tutorials* at: https://confluence.atlassian.com/bitbucket/tutorial-learn-bitbucket-with-mercurial-759857394.html; February 15, 2019 was the last accessed date, October 24 2018.

[22] Muhammad Ali Babar, Alan W. Brown, and Ivan Mistrik. Agile Software Architecture: Aligning Agile Processes and Software Architectures. Morgan Kaufmann, Waltham, MA, 2014.

[23] Brian Bailey and Grant Martin. ESL Models and their Application: Electronic System Level Design and Verification in Practice. Embedded Systems. Springer Science+Business Media, LLC, New York, NY, 2010.

[24] Brian Bailey, Grant Martin, and Thomas Anderson. Taxonomies for the Development and Verification of Digital Systems. Springer Science+Business Media, Inc., New York, NY, 2005.

[25] Douglas Baldwin and Greg W. Scragg. Algorithms and Data Structures: The Science of Computing. Computer Engineering Series. Charles River Media, Hingham, MA, 2004.

[26] Lorena A. Barba. The hard road to reproducibility. Science, 354(6308):142, October 7 2016.

[27] Michael Barr. Embedded C coding standard. Self-published, Gaithersburg, MD, 2013.

[28] Wojciech Basalaj and Richard Corden. High integrity C++ coding standard V4.0: An overview. Technical report, Programming Research Ltd., Hersham, Surrey, England, U.K., November 2013.

[29] Benjamin S. Baumer, Daniel T. Kaplan, and Nicholas J. Horton. Modern Data Science with R. Texts in Statistical Science. CRC Press, Boca Raton, FL, 2017.

[30] Kent Beck and Cynthia Andres. Extreme Programming Explained: Embrace Change. The XP Series. Pearson Education, Upper Saddle River, NJ, second edition, 2005.

[31] Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for Agile software development. Available online at: http://www.agilemanifesto.org/; February 7, 2016 was the last accessed date, 2001.

[32] Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Principles behind the agile manifesto. Available online from *Manifesto for Agile Software Development* at: http://agilemanifesto.org/principles.html; June 9, 2018 was the last accessed date, 2001.

[33] Mordechai Ben-Ari. Understanding Programming Languages. John Wiley & Sons, Chichester, West Sussex, South East England, England, U.K., 1996.

[34] James Bender and Jeff McWherter. Professional Test Driven Development with C#: Developing Real-World Applications with TDD. Wrox Programmer to Programmer™. Wiley Publishing, Indianapolis, IN, 2011.

[35] Lionel Bening and Harry Foster. Principles of Verifiable RTL Design: A Functional Coding Style Supporting Verification Processes in Verilog. Kluwer Academic Publishers, New York, NY, 2000.

[36] Lionel Bening and Harry Foster. Principles of Verifiable RTL Design: A functional coding style supporting verification processes in Verilog. Kluwer Academic Publishers, New York, NY, second edition, 2001.

[37] Jon Bentley. Programming Pearls. Addison-Wesley, Reading, MA, second edition, 2000.

[38] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. ACM SIGARCH Computer Architecture News, 39(2):1–7, May 2011.

[39] David Black, Hemendra Talesara, and Neil Johnson. Using agile techniques with ESL: For better results. Available online from *North American SystemC User's Group (NASCUG): NASCUG 14* at: http://www.nascug.org/events/14th/Agile_ESL.pdf; self-published; September 25, 2018 was the last accessed date, 2010.

[40] Steve Blackburn and Matthias Hauswirth. Artifact evaluation. Available online from *University of Lugano: Faculty of Informatics: Evaluate Collaboratory, Experimental Evaluation of Software and Systems in Computer Science* at: http://evaluate.inf.usi.ch/artifacts; June 25, 2017 was the last accessed date.

[41] Peter A. Blume. The LabVIEW Style Book. Prentice Hall, Upper Saddle River, NJ, 2007.

[42] Barry Boehm and Richard Turner. Balancing Agility and Discipline: A Guide for the Perplexed. Pearson Education, Boston, MA, 2004.

[43] Nicola Bombieri, Franco Fummi, and Valerio Guarnieri. Fast: An rtl fault simulation framework based on rtl-to-tlm abstraction. Journal of Electronic Testing, 28(4):495–510, August 2012.

[44] Seshu Brahma. The benefits of personal projects. Available online from *The University of Texas at Austin: College of Natural Sciences: Department of Computer Science: ReadMe – UTCS Student Blogs: Seshu Brahma's blog* at: http://www.cs.utexas.edu/blog/benefits-personal-projects; April 17, 2017 was the last accessed date, April 14 2017.

[45] Bo Brinkman, Catherine Flick, Don Gotterbarn, Keith Miller, Kate Vazansky, and Marty J. Wolf. Listening to professional voices: Draft 2 of the ACM code of ethics and professional conduct. Communications of the ACM, 60(5):105–111, May 2017.

[46] Bo Brinkman, Don Gotterbarn, Keith Miller, and Marty J. Wolf. Making a positive impact: Updating the ACM code of ethics. Communications of the ACM, 59(12):7–13, December 2016.

[47] Bo Brinkman, Don Gotterbarn, Keith W. Miller, and Marty J. Wolf. All hands on deck for ACM ethics: Updating the Code, revising enforcement, promoting integrity. ACM Computers and Society: Newsletter of the ACM Special Interest Group on Computers and Society, 46(3):5–8, November 2016.

[48] Frederick P. Brooks, Jr. The Mythical Man-Month: Essays on Software Engineering. Addison-Wesley, Reading, MA, 20^{th} anniversary edition, 1995.

[49] Frederick P. Brooks, Jr. The Design of Design: Essays from a Computer Scientist. Pearson Education, Boston, MA, 2010.

[50] William J. Brown, Raphael C. Malveau, Hays W. "Skip" McCormick, III, and Thomas J. Mowbray. AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis. John Wiley & Sons, Hoboken, NJ, 1998.

[51] Ian Burgess. Regression Testing: Gate-Level Functional Verification Is Imperative and Equivalence Checking Provides the Solution. Mentor Graphics Corporation, Wilsonville, OR, 2004.

[52] Mark Burton and Adam Morawiec. Platform Based Design at the Electronic System Level: Industry Perspectives and Experiences. Springer, Dordrecht, The Netherlands, 2006.

[53] Giorgio C. Buttazzo. Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, volume 24 of Real-Time Systems Series. Springer Science+Business Media, LLC, New York, NY, third edition, 2011.

[54] Field Cady. The Data Science Handbook. John Wiley & Sons, Hoboken, NJ, 2017.

[55] Luiz Fernando Capretz and Miriam A. M. Capretz. Object-Oriented Software Design and Maintenance, volume 6 of Series on Software Engineering and Knowledge Engineering. World Scientific Publishing, Singapore, 1996.

[56] Tom Cargill. C++ Programming Style. Addison-Wesley, Reading, MA, 1992.

[57] Frank M. Carrano. Data Structures and Abstractions with Java. Prentice Hall, Upper Saddle River, NJ, third edition, 2012.

[58] Scott Chacon. Pro Git. Apress, New York, NY, 2009.

[59] Scott Chacon and Ben Straub. Pro Git: Everything You Need to Know About Git. The Expert's Voice$^{\text{ő}}$. Apress Media, LLC, Berkeley, CA, second edition, 2014.

[60] Henry Chang, Larry Cooke, Merrill Hunt, Grant Martin, Andrew McNelly, and Lee Todd. Surviving the SoC Revolution: A Guide to Platform-Based Design. Kluwer Academic Publishers, New York, NY, 2002.

[61] Pankaj P. Chauhan. Verification of Large Industrial Circuits Using SAT Based Reparameterization and Automated Abstraction-Refinement. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, May 2007.

[62] Jess Chen, Michael Henrie, Monte F. Mar, and Mladen Nizic. Mixed-Signal Methodology Guide: Advanced Methodology for AMS IP and SoC Design, Verification, and Implementation. Lulu Enterprises, Raleigh, NC, August 2012.

[63] Alistair Cockburn. Agile Software Development: The Cooperative Game. Agile Software Development Series. Pearson Education, Boston, MA, second edition, 2007.

[64] Code Ocean staff. Code ocean: Discover & run scientific code. Available online at: https://codeocean.com; February 2, 2018 was the last accessed date, 2017.

[65] Ken Collier. Agile Analytics: A Value-Driven Approach to Business Intelligence and Data Warehousing. Agile Software Development Series. Addison-Wesley, Boston, MA, 2012.

[66] Hewlett-Packard Company. Dave packard's 11 simple rules. Available online in *Hewlett-Packard Company: Retiree Home: History: Founders & early contributors–Dave Packard* at: http://www.hp.com/retiree/history/founders/packard/11rules.html; March 15, 2014 was the last accessed date, 2012.

[67] Jason Cong and Joseph R. Shinnerl. Multilevel Optimization in VLSICAD, volume 14 of Combinatorial Optimization. Kluwer Academic Publishers, New York, NY, 2003.

[68] Jon Conway and Steve Watts. A Software Engineering Approach to LabVIEW™. National Instruments Virtual Instrumentation Series. Prentice Hall, Upper Saddle River, NJ, 2003.

[69] James O. Coplien and Neil B. Harrison. Organizational Patterns of Agile Software Development 2004. Prentice-Hall, Upper Saddle River, NJ, 2005.

[70] Lawrence Corr and Jim Stagnitto. Agile Data Warehouse Design: Collaborative Dimensional Modeling, from Whiteboard to Star Schema 2011. DecisionOne Press, Leeds, West Yorkshire, Yorkshire and the Humber, England, U.K., 2011.

[71] Victor Costan, Titus Winters, Ted Osborne, Wei Zhang, Tom Manshreck, Stephen Hicks, Mark Mentovai, Geoff Romer, frp, Alan Yee, and Ackermann Yuriy. Google C++ style guide. Available online from *GitHub: Google: styleguide* at: https://google.github.io/styleguide/cppguide.html and https://github.com/google/styleguide/blob/gh-pages/cppguide.html; September 17, 2019 was the last accessed date, September 5 2018.

[72] Coverity. Coverity scan: 2012 open source report. Technical report, Coverity, Inc., San Francisco, CA, 2012.

[73] John W. Creswell. Research Design: Qualitative, Quantitative, and Mixed Methods Approaches. SAGE Publications, Thousand Oaks, CA, fourth edition, 2014.

[74] Edward Crookshanks. Practical Software Development Techniques: Tools and Techniques for Building Enterprise Software. Nokel Services, third edition, 2012.

[75] Aristides Dasso and Ana Funes. Verification, Validation and Testing in Software Engineering. Idea Group Publishing, Hershey, PA, 2007.

[76] Datopian (Atomatic Ltd), Inc. staff. DataHub: Frictionless data, data online made simple. Available online at: http://datahub.io; February 2, 2018 was the last accessed date, 2017.

[77] Rachel Davies and Liz Sedley. Agile Coaching. Pragmatic Bookshelf. The Pragmatic Programmers, Raleigh, NC, 2009.

[78] Chad Davis. Mental hygiene for software engineers: Avoiding micro level technical debt. Available online from *DZone* at: https://dzone.com/articles/mental-hygiene-software; September 27, 2018 was the last accessed date, January 23 2013.

[79] Andrea De Lucia, Filomena Ferrucci, Genny Tortora, and Maurizio Tucci. Emerging Methods, Technologies, and Process Management in Software Engineering. John Wiley & Sons, Hoboken, NJ, 2008.

[80] Mathias De Maré and other contributors. Mercurial: the definitive guide. Available online from *Mercurial community: Mercurial SCM: Mercurial – the definitive guide* as version/build 1.90-ebb51bf94742 at: https://book.mercurial-scm.org/read/ and https://book.mercurial-scm.org/; self-published; February 15, 2019 was the last accessed date, 2015.

[81] Douglas Densmore and Abhijit Davare. A Platform-Based Design Methodology for the Electronic System Level: Frameworks, Design Flows, and Case Studies. VDM Verlag Dr. Müller, Saarbrücken, Germany, 2008.

[82] Andrew DeOrio and Andrew Giugliano. Long-term effects of partner programming in an introductory computer science sequence. In Proceedings of the 123rd ASEE Annual Conference and Exposition on Engineering Education, number 15189 in ASEE Conferences and Meetings, New Orleans, LA, June 26–29 2016. American Society for Engineering Education, American Society for Engineering Education.

[83] Esther Derby and Diana Larsen. Agile Retrospectives: Making Good Teams Great. The Pragmatic Programmers, Raleigh, NC, 2006.

[84] Davide Di Gennaro. Advanced C++ metaprogramming. Self-published, 2011.

[85] Alex Doboli and Edward H. Currie. Introduction to Mixed-Signal, Embedded Design. Springer Science+Business Media, LLC, New York, NY, 2011.

[86] John F. Dooley. Software Development, Design and Coding: With Patterns, Debugging, Unit Testing, and Refactoring. Apress Media, LLC, Berkeley, CA, second edition, 2017.

[87] Evan Driscoll. Latexdiff notes. Available online from *University of Wisconsin-Madison: University of Wisconsin-Madison College of Engineering: Department of Computer Sciences: Evan Driscoll's web page: Writings on Software: LaTeX* at: http://pages.cs.wisc.edu/~driscoll/software/latex/latexdiff.html; February 15, 2016 was the last accessed date.

[88] Evan Driscoll. Scons documentation recommendatitons. Available online from *Evan Driscoll's web page: Writings on Software: SCons*, the Department of Computer Sciences, University of Wisconsin–Madison College of Engineering, University of Wisconsin–Madison at: http://pages.cs.wisc.edu/~driscoll/software/scons/index.html; February 15, 2016 was the last accessed date.

[89] Evan Driscoll. Version control. Available online from *Evan Driscoll's web page: Writings on Software*, the Department of Computer Sciences, University of Wisconsin–Madison College of Engineering, University of Wisconsin–Madison at: http://pages.cs.wisc.edu/~driscoll/software/vcs/; February 15, 2016 was the last accessed date, February 22 2012.

[90] Jutta Eckstein. Agile Software Development with Distributed Teams: Staying Agile in a Global World. Dorset House Publishing, New York, NY, 2010.

[91] Benny Evangelista. Facebook's hacker way – "move fast and break things". Available online from *The San Francisco Chronicle: The SF Gate: Blogs at SFGate.com: The Technology Chronicles* at: http://blog.sfgate.com/techchron/2012/02/01/facebooks-hacker-way-move-fast-and-break-things/; October 9, 2014 was the last accessed date, February 1 2012.

[92] Eric Evans. Domain-Driven Design: Tackling Complexity in the Heart of Software. Pearson Education, Boston, MA, 2004.

[93] Philippe Faes. What is hardware refactoring? Available online from *Sigasi: Sigasi Insights: Tech Articles* at: http://insights.sigasi.com/tech/hardware_refactoring.html; September 26, 2018 was the last accessed date, December 18 2008.

[94] Philippe Paul Henri Faes and Hendrik Richard Pieter Eeckhaut. Device and method for refactoring hardware code. Available online from *Google Patents* at: https://patents.google.com/patent/EP2175387A1/en; September 26, 2018 was the last accessed date, April 14 2010.

[95] Michael C. Feathers. Working Effectively with Legacy Code. Robert C. Martin. Pearson Education, Upper Saddle River, NJ, 2005.

[96] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. How to Design Programs: An Introduction to Programming and Computing. The MIT Press, Cambridge, MA, 2001.

[97] Mark Fewster and Dorothy Graham. Software Test Automation: Effective Use of Test Execution Tools. ACM Press and Addison-Wesley, New York, NY and Harlow, Essex, England, U.K., 1999.

[98] Jay Fields, Shane Harvie, Martin Fowler, and Kent Beck. Refactoring. Addison-Wesley Signature Series. Addison-Wesley, Boston, MA, ruby edition, 2010.

[99] figshare LLP staff. figshare: credit for all your research. Available online at: https://figshare.com/; November 4, 2016 was the last accessed date, 2016.

[100] Michael Fingeroff. High-Level Synthesis Blue Book. Xlibris Corporation, Bloomington, IN, 2010.

[101] Rachel Fong. Move fast and break things. Available online in *MIT Admissions: Blogs: Academics & Research* at: http://mitadmissions.org/blogs/entry/move_fast_and_break_things; October 9, 2014 was the last accessed date, June 25 2011.

[102] Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. Refactoring: Improving the Design of Existing Code. Addison-Wesley Object Technology Series. Addison-Wesley, Reading, MA, 1999.

[103] Armando Fox and David Patterson. Engineering Long-Lasting Software: An Agile Approach Using SaaS and Cloud Computing. Strawberry Canyon LLC, beta 0.9.1 edition, January 10 2013.

[104] Daniel França. C++ has become more Pythonic-Jeff Preshing. Available online from *Standard C++: Blog* at: https://isocpp.org/blog/2014/12/c-has-become-more-pythonic; self-published; February 1, 2018 was the last accessed date, December 2 2014.

[105] Ronny Frevert, Joachim Haase, Roland Jancke, Uwe Knöchel, Peter Schwarz, Ralf Kakerow, and Mohsen Darianian. Modeling and Simulation for RF System Design. Springer, Dordrecht, The Netherlands, 2005.

[106] Steve Furber. ARM System-on-Chip Architecture. Pearson Education, Harlow, Essex, East of England, England, U.K., second edition, 2000.

[107] Brian Gadoury. Keep your tools sharp to avoid personal technical debt. Available online from *End Point: Our Blog* at: https://www.endpoint.com/blog/2010/10/12/keep-your-tools-sharp-to-avoid-personal; September 26, 2018 was the last accessed date, October 12 2010.

[108] Daniel D. Gajski, Samar Abdi, Andreas Gerstlauer, and Gunar Schirner. Embedded System Design: Modeling, Synthesis and Verification. Springer Science+Business Media, LLC, New York, NY, 2009.

[109] Christopher Gandrud. Reproducible Research with R and RStudio. Chapman & Hall/CRC The R Series. CRC Press, Boca Raton, FL, 2014.

[110] Christopher Gandrud. Reproducible Research with R and RStudio. Chapman & Hall/CRC The R Series. CRC Press, Boca Raton, FL, second edition, 2015.

[111] Matthias Geier. Reproducible research. Available online from *Matthias Geier's web page* at: http://mg.readthedocs.io/reproducible_research.html; self-published; April 24, 2017 was the last accessed date.

[112] gem5 developers. The gem5 simulator system: A modular platform for computer system architecture research. Available online at: http://www.gem5.org/Main_Page; self-published; October 9, 2014 was the last accessed date, October 27 2014.

[113] Andreas Gerstlauer, Rainer Dömer, Junyu Peng, and Daniel D. Gajski. System Design: A Practical Guide with SpecC. Kluwer Academic Publishers, Norwell, MA, 2001.

[114] Imran Ghani, Dayang Norhayati Abang Jawawi, Siva Dorairaj, and Ahmed Sidky. Emerging Innovations in Agile Software Development. Advances in Systems Analysis, Software Engineering, and High Performance Computing. Information Science Reference, Hershey, PA, 2016.

[115] Joseph (Yossi) Gil and David H. Lorenz. Design patterns and language design. IEEE Computer, 31(3):118–120, March 1998.

[116] Git contributors. Community. Available online at: http://www.git-scm.com/community; August 1, 2014 was the last accessed date, 2014.

[117] Git contributors. Documentation. Available online at: http://www.git-scm.com/doc; August 1, 2014 was the last accessed date, 2014.

[118] Git contributors. Reference. Available online at: http://www.git-scm.com/docs; August 1, 2014 was the last accessed date, 2014.

[119] GitHub staff. Git reference. Available online at: http://gitref.org/; August 1, 2014 was the last accessed date, 2014.

[120] GitHub staff. GitHub. Available online at: https://github.com/; November 4, 2016 was the last accessed date, 2016.

[121] GitHub Training Team. GitHub Git cheat sheet. Available online at: https://training.github.com/kit/downloads/github-git-cheat-sheet.pdf; August 1, 2014 was the last accessed date, 2014.

[122] GitHub Training Team. GitHub training. Available online at: https://training.github.com/; August 14, 2014 was the last accessed date, 2014.

[123] GitHub Training Team. GitHub training kit: Open source teaching resources – open source training content. present your way to better version control. Available online at: https://training.github.com/kit/; August 14, 2014 was the last accessed date, 2014.

[124] Marcelo Glusman, Gila Kamhi, Sela Mador-Haim, Ranan Fraer, and Moshe Y. Vardi. Multiple-counterexample guided iterative abstraction refinement: An industrial evaluation. In Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2003), volume 2619 of Lecture Notes in Computer Science, pages 176–191, Warsaw, Poland, April 7–11 2003. European Association for Theoretical Computer Science (EATCS), European Association for Programming Languages and Systems (EAPLS), European Association of Software Science and Technology (EASST), ACM Special Interest Group on Software Engineering (SIGSOFT), ACM Special Interest Group on Programming Languages (SIGPLAN), and ACM Special Interest Group on Algorithms and Computation Theory (SIGACT), Springer-Verlag Berlin Heidelberg.

[125] Richard Goering. Can agile software development methods help SoC design? Available online from *Cadence Design Systems: Cadence Community: Blogs: Industry Insights Blogs* at: https://community.cadence.com/cadence_blogs_8/b/ii/posts/can-agile-software-development-methods-help-soc-design; September 25, 2018 was the last accessed date, August 18 2010.

[126] Luis Gonçalves and Ben Linders. Getting value out of agile retrospectives: A toolbox of retrospective exercises. Self-published, Raleigh, NC, September 12 2014.

[127] Google Colab staff. Colaboratory. Available online from *Google: Google AI: Research: Google Colab* at: https://colab.research.google.com/; February 13, 2019 was the last accessed date, 2019.

[128] Google styleguide contributors. styleguide: Google style guides. Available online from *GitHub: Google* at: http://google.github.io/styleguide/ and https://github.com/google/styleguide; September 17, 2019 was the last accessed date, September 10 2019.

[129] Don Gotterbarn, Bo Brinkman, Catherine Flick, Michael S. Kirkpatrick, Keith Miller, Kate Varansky, Marty J. Wolf, Eve Anderson, Ron Anderson, Amy Bruckman, Karla Carter, Michael Davis, Penny Duquenoy, Jeremy Epstein, Kai Kimppa, Lorraine Kisselburgh, Shrawan Kumar, Andrew McGettrick, Natasa Milic-Frayling, Denise Oram, Simon Rogerson, David Shama, Janice Sipior, Eugene Spafford, and Les Waguespack. ACM code of ethics and professional conduct. Available online from the *ACM Committee on Professional Ethics* at: https://www.acm.org/code-of-ethics; September 19, 2018 was the last accessed date, June 22 2018.

[130] Don Gotterbarn, Bo Brinkman, Catherine Flick, Keith Miller, Marty Wolf, Kate Vazansky, Florence Appel, Karla Carter, Fran Grodzinsky, Michael S. Kirkpatrick, Anthony Lobo, Denise Oram, and Simon Rogerson. ACM code of ethics and professional conduct. Available online from the *ACM Committee on Professional Ethics* at: https://ethics.acm.org; September 19, 2018 was the last accessed date, June 22 2018.

[131] Don Gotterbarn, Amy Bruckman, Catherine Flick, Keith Miller, and Marty J. Wolf. ACM code of ethics: A guide for positive action. Communications of the ACM, 61(1):121–128, January 2018.

[132] Don Gotterbarn, Keith Miller, and Simon Rogerson. Software engineering code of ethics. Communications of the ACM, 40(11):110–118, November 1997.

[133] Donald Gotterbarn, Keith Miller, Simon Rogerson, Steve Barber, Peter Barnes, Ilene Burnstein, Michael Davis, Amr El-Kadi, N. Ben Fairweather, Milton Fulghum, N. Jayaram, Tom Jewett, Mark Kanko, Ernie Kallman, Duncan Langford, Joyce Currie Little, Ed Mechler, Manuel J. Norman, Douglas Phillips, Peter Ron Prinzivalli, Patrick Sullivan, John Weckert, Vivian Weil, S. Weisband, and Laurie Honour Werth. The joint ACM/IEEE-CS software engineering code of ethics and professional practice. Available online from *ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices (SEEPP)* at: https://ethics.acm.org/code-of-ethics/software-engineering-code/; September 19, 2018 was the last accessed date, 1999.

[134] Yuepu Guo. Measuring and Monitoring Technical Debt. PhD thesis, University of Maryland, Baltimore County, Baltimore, MD, 2016.

[135] Rajesh Gupta, Paul Le Guernic, Sandeep Kumar Shukla, and Jean-Pierre Talpin. Formal Methods and Models for System Design: A System Level Perspective. Springer Science+Business Media Dordrecht, Dordrecht, The Netherlands, 2004.

[136] Philip N. Hagelberg. Leiningen. Available online from at: https://github.com/technomancy/leiningen; self-published; October 23, 2017 was the last accessed date, October 21 2017.

[137] Max Hailperin, Barbara Kaiser, and Karl Knight. Concrete Abstractions: An Introduction to Computer Science Using Scheme. Brooks/Cole, Pacific Groove, CA, 1999.

[138] Junio C. Hamano, Linus Torvalds, Shawn Pearce, Johannes Schindelin, Nicolas Pitre, René Scharfe, Jeff King, Jonathan Nieder, Johan Herland, Johannes Sixt, Sverre Rabbelier, Michael J. Gruber, Nguyễn Thái Ngọc Duy, Ævar Arnfjörð Bjarmason, and Thomas Rast. Git. Available online at: http://www.git-scm.com/ and http://git-blame.blogspot.com/; August 1, 2014 was the last accessed date, August 12 2014.

[139] David Harel, Dexter Kozen, and Jerz Tiuryn. Dynamic Logic. Foundations of Computing. The MIT Press, Cambridge, MA, 2000.

[140] David Money Harris and Sarah L. Harris. Digital Design and Computer Architecture. Morgan Kaufmann, Waltham, MA, 2007.

[141] Steve Haynal. Refactoring to prepare RTL for reuse. Available online from *SofterHardware: Publications* at: http://www.softerhardware.com/pdf/ip08a.pdf; self-published; September 26, 2018 was the last accessed date, 2008.

[142] Steve Haynal. Refactoring to prepare RTL for reuse. Available online from *Design & Reuse: D&R Industry Articles* at: https://www.design-reuse.com/articles/20119/refactoring-rtl-reuse.html; linked from https://www.design-reuse.com/articles/archive200902.html; September 26, 2018 was the last accessed date, February 16 2009.

[143] Steve Haynal, Timothy Kam, Michael Kishinevsky, Emily Shriver, and Xinning Wang. A SystemVerilog rewriting system for RTL abstraction with Pentium case study. In Proceedings of the 6^{th} ACM/IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE '08), pages 79–88, Anaheim, CA, June 5–7 2008. ACM Special Interest Group

on Design Automation (SIGDA), ACM Special Interest Group on Embedded Systems (SIGBED), IEEE Circuits and Systems Society (CAS), and IEEE Council on EDA (CEDA), IEEE Press.

[144] John Hennessy and David Patterson. A new golden age for computer architecture: Domain-specific hardware/software co-design, enhanced security, open instruction sets, and agile chip development. Available online from *IEEE Consultants' Network of Silicon Valley (IEEE-CNSV): Events* at: https://californiaconsultants.org/event/annual-cnsv-dinner-meeting-a-new-golden-age-for-computer-architecture/ and https://californiaconsultants.org/wp-content/uploads/2018/04/CNSV-1806-Patterson.pdf; July 20, 2018 was the last accessed date, June 13 2018.

[145] Tsung-Yi Ho, Yao-Wen Chang, and Sao-Jie Chen. Full-Chip Nanometer Routing Techniques. Analog Circuits and Signal Processing. Springer, Dordrecht, The Netherlands, 2007.

[146] Todd Hoff. C++ coding standard. Available online from *Possibility Outpost: Programmer's Corner* at: http://www.possibility.com/Cpp/CppCodingStandard.html; February 16, 2016 was the last accessed date, March 1 2008.

[147] Jez Humble and David Farley. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley Signature Series. Pearson Education, Boston, MA, 2011.

[148] Andy Hunt. Pragmatic Thinking and Learning: Refactor Your Wetware. Pragmatic Bookshelf. The Pragmatic Programmers, Raleigh, NC, September 15 2008.

[149] John Hunt. Agile Software Construction. Springer-Verlag London, London, U.K., 2006.

[150] Mohamed Hamed Zaki Hussein. Techniques for the Formal Verification of Analog and Mixed-Signal Designs. PhD thesis, Concordia University, Montréal, Québec, Canada, 2008.

[151] Michael Huth and Mark Ryan. Logic in Computer Science: Modelling and Reasoning about Systems. Cambridge University Press, New York, NY, second edition, 2004.

[152] IEEE Board of Directors. IEEE code of ethics. Available online from *IEEE* at: http://www.ieee.org/about/ieee_code_of_conduct.pdf; October 2, 2014 was the last accessed date, June 2014.

[153] IEEE Board of Directors. IEEE code of ethics. In IEEE Policies, page 7.3. Institute of Electrical and Electronics Engineers, New York, NY, 2014.

[154] IEEE Board of Directors. IEEE policies. Technical report, Institute of Electrical and Electronics Engineers, New York, NY, August 2014.

[155] IEEE Board of Directors. Nondiscrimination policy. In IEEE Policies, page 9.8. Institute of Electrical and Electronics Engineers, New York, NY, 2014.

[156] Intel Corporation staff. Intel® 64 and IA-32 Architectures Optimization Reference Manual. Intel Corporation, Santa Clara, CA, September 2015.

[157] International DOI Foundation staff. Digital object identifier system. Available online at: https://www.doi.org/; January 20, 2017 was the last accessed date, January 10 2017.

[158] Pankaj Jalote. A Concise Introduction to Software Engineering. Undergraduate Topics in Computer Science. Springer-Verlag London, London, U.K., 2008.

[159] Ron Jeffries. The Nature of Software Development: Keep It Simple, Make It Valuable, Build It Piece by Piece. The Pragmatic Programmers, Raleigh, NC, 2015.

[160] Randall W. Jensen. Improving Software Development Productivity: Effective Leadership and Quantitative Methods in Software Management. Prentice Hall, Upper Saddle River, NJ, 2015.

[161] Neil Johnson and Bryan Morris. AgileSoC: Bring agile to the world of hardware development. Available online at: http://agilesoc.com; self-published; September 25, 2018 was the last accessed date, 2018.

[162] Jon. What does pythonic mean? [closed]. Available online from *Stack Exchange Inc.: Stack Overflow: Questions* at: https://stackoverflow.com/questions/25011078/what-does-pythonic-mean; February 1, 2018 was the last accessed date, July 29 2014.

[163] Hubert Kaeslin. Top-Down Digital VLSI Design: From Architectures to Gate-Level Circuits and FPGAs. Morgan Kaufmann, Waltham, MA, 2015.

[164] Michael Keating and Pierre Bricaud. Reuse Methodology Manual: For System-on-a-Chip Designs. Kluwer Academic Publishers, Dordrecht, The Netherlands, second edition, 1999.

[165] Michael Keating and Pierre Bricaud. Reuse Methodology Manual: For System-on-a-Chip Designs. Kluwer Academic Publishers, Dordrecht, The Netherlands, third edition, 2002.

[166] Eric R. Keiter, Thomas V. Russo, Richard L. Schiek, Peter E. Sholander, Heidi K. Thornquist, Ting Mei, Jason C. Verley, and David G. Baur. Building guide: How to build Xyce™ from source code. Technical Report SAND2013-7294, Sandia National Laboratories, Sandia Corporation, Albuquerque, NM, August 2013.

[167] Allan Kelly. Changing Software Development: Learning to Become Agile. John Wiley & Sons, Chichester, West Sussex, South East England, England, U.K., 2008.

[168] Allan Kelly. Xanpan: Team Centric Agile Software Development – Combining Kanban and XP - inspiration for creating your own hybrid. Software Strategy Ltd., London, U.K., 2015.

[169] Joshua Kerievsky. Refactoring to Patterns. Addison-Wesley Signature Series. Addison-Wesley, Boston, MA, 2005.

[170] Brian W. Kernighan and Rob Pike. The Practice of Programming. Addison-Wesley Professional Computing Series. Addison-Wesley, Reading, MA, 1999.

[171] Brian W. Kernighan and P. J. Plauger. The Elements of Programming Style. McGraw-Hill, New York, NY, second edition, 1982.

[172] Kurt Keutzer, Sharad Malik, A. Richard Newton, Jan M. Rabaey, and Alberto Sangiovanni-Vincentelli. System-level design: Orthogonalization of concerns and platform-based design. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 19(12):1523–1543, December 2000.

[173] Joshua Kim. 'whiplash', the MIT media lab, and the rest of us. Available online from *Inside Higher Ed: Higher Education Blogs: Blog U* at: https://www.insidehighered.com/blogs/technology-and-learning/whiplash-mit-media-lab-and-rest-us; February 24, 2017 was the last accessed date, January 22 2017.

[174] Donald E. Knuth. Literate programming. The Computer Journal, 27(2):97–111, 1984.

[175] Donald E. Knuth. Literate Programming. Center for the Study of Language and Information – Lecture Notes. The University of Chicago Press, Chicago, IL, 1992.

[176] Philip Koopman. Better Embedded System Software. Drumnadrochit Education, 2010.

[177] Helmut Kopka and Patrick W. Daly. Guide to LaTeX. Addison-Wesley Series on Tools and Techniques for Computer Typesetting. Addison-Wesley, Boston, MA, fourth edition, 2004.

[178] Derrick G. Kourie and Bruce W. Watson. The Correctness-by-Construction Approach to Programming. Springer-Verlag Berlin Heidelberg, Heidelberg, Germany, 2012.

[179] R. S. Krishnamachari and P. Y. Papalambros. Hierarchical decomposition synthesis in optimal systems design. ASME Journal of Mechanical Design, 119(4):448–457, December 1 1997.

[180] Ramprasad S. Krishnamachari and Panos Y. Papalambros. Hierarchical decomposition synthesis in optimal systems design. Technical Report UMR2480, University of Michigan, Ann Arbor, MI, August 1995.

[181] Shriram Krishnamurthi. Artifact evaluation for software conferences. Available online from *Brown University: Computer Science Department: Prof. Shriram Krishnamurthi's web page* at: http://www.artifact-eval.org; self-published; June 25, 2017 was the last accessed date.

[182] Bruce H. Krogh, Edward Lee, Insup Lee, Al Mok, George Pappas, Raj Rajkumar, Harvey Rubin, Alberto Sangiovanni-Vincentelli, Lui Sha, Kang Shin, Jack Stankovic, Janos Sztipanovits, Wayne Wolf, and Wei Zhao. Report: Cyber-physical systems summit. Technical report, National Science Foundation, St. Louis, MO, April 24–25 2008.

[183] Thomas Kropf. Introduction to Formal Hardware Verification. Springer-Verlag Berlin Heidelberg, Heidelberg, Germany, 1999.

[184] Philippe Kruchten, Robert L. Nord, and Ipek Ozkaya. Technical debt: From metaphor to theory and practice. IEEE Software, 29(6):18–21, November–December 2012.

[185] Sudipta Kundu, Sorin Lerner, and Rajesh K. Gupta. High-Level Verification: Methods and Tools for Verification of System-Level Designs. Springer Science+Business Media, LLC, New York, NY, 2011.

[186] David Kushner. Facebook philosophy: Move fast and break things – hacker culture is alive and well at Facebook. Available online in *IEEE Spectrum* at: http://spectrum.ieee.org/at-work/innovation/facebook-philosophy-move-fast-and-break-things/0; October 9, 2014 was the last accessed date, June 1 2011.

[187] Gayle Laakmann. Cracking the technical interview: 150 technical interview questions and solutions, written by experts. Self-published, 2009.

[188] Jean J. Labrosse. Embedded Systems Building Blocks: Complete and Ready-to-Use Modules in C. CMP Books, Lawrence, KS, second edition, 1999.

[189] Phillip A. Laplante. Real-Time Systems Design and Analysis. John Wiley & Sons, Hoboken, NJ, third edition, 2004.

[190] Phillip A. Laplante. Requirements Engineering for Software and Systems. Auerbach Applied Software Engineering. CRC Press, Boca Raton, FL, second edition, 2014.

[191] Phillip A. Laplante and Seppo J. Ovaska. Real-Time Systems Design and Analysis: Tools for the Practitioner. John Wiley & Sons and IEEE Press, Hoboken, NJ and Piscataway, NJ, fourth edition, 2012.

[192] Craig Larman. Agile and Iterative Development: A Manager's Guide. The Agile Software Development. Addison-Wesley, Boston, MA, 2004.

[193] Craig Larman. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. Prentice Hall, Upper Saddle River, NJ, third edition, 2005.

[194] Craig Larman and Victor R. Basili. Iterative and incremental development: A brief history. IEEE Computer, 36(6):47–56, June 2003.

[195] Craig Larman and Bas Vodde. Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum. Pearson Education, Boston, MA, 2009.

[196] Craig Larman and Bas Vodde. Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum. Pearson Education, Boston, MA, 2010.

[197] Diana Larsen and Ainsley Nies. Liftoff: Launching agile teams & projects. Self-published, Hillsboro, OR, 2011.

[198] Edward Ashford Lee and Sanjit Arunkumar Seshia. Introduction to Embedded Systems: A Cyber-Physical Systems Approach. The MIT Press, Cambridge, MA, second edition, 2017.

[199] Kent D. Lee. Programming Languages: An Active Learning Approach. Springer Science+Business Media, LLC, New York, NY, 2008.

[200] Dean Leffingwell. Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise. Agile Software Development Series. Addison-Wesley, Boston, MA, 2011.

[201] Mike Levin. Incurring more personal technical debt, but so what? Available online from *Medium: Mike Levin, SEO in NYC* at: https://medium.com/@mikelevinseo/incurring-more-personal-technical-debt-but-so-what-aad4bca1fa5c; September 27, 2018 was the last accessed date, October 27 2017.

[202] Ming Li and Paul Vitányi. An Introduction to Kolmogorov Complexity and Its Applications. Texts in Computer Science. Springer Science+Business Media, LLC, New York, NY, third edition, 2008.

[203] Mark Liberman. Replicability vs. reproducibility – or is it the other way around? Available online from the *University of Pennsylvania: School of Arts & Sciences: Linguistic Data Consortium (LDC)* at: http://languagelog.ldc.upenn.edu/nll/?p=21956; April 21, 2017 was the last accessed date, October 31 2015.

[204] Eric Lippert. Vexing exceptions. Available online from *Microsoft Corporation: Microsoft Developer network: MSDN Blogs: Fabulous Adventures In Coding, Eric Lippert's Erstwhile Blog* at: https://blogs.msdn.microsoft.com/ericlippert/2008/09/10/vexing-exceptions/; September 19, 2018 was the last accessed date, September 10 2008.

[205] Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland, and David Svoboda. Java™ Coding Guidelines: 75 Recommendations for Reliable and Secure Programs. The SEI Series in Software Engineering. Addison-Wesley, Upper Saddle River, NJ, 1995.

[206] Kenneth C. Louden and Kenneth Alfred Lambert. Programming Languages: Principles and Practices. Course Technology, Boston, MA, 2012.

[207] Zhonghai Lu. Design and Analysis of On-Chip Communication for Network-on-Chip Platforms. PhD thesis, KTH Royal Institute of Technology, Stockholm, Sweden, March 2007.

[208] Ian Macdonald. The unofficial Ruby usage guide. Available online from *Caliban – Opinion and Righteous Anger: Ruby Projects: Recommended Tutorials* at: http://www.caliban.org/ruby/rubyguide.shtml; self-published; October 20, 2015 was the last accessed date.

[209] Vijay K. Madisetti and Chonlameth Arpikanondt. A Platform-Centric Approach to System-on-Chip (SOC) Design. Springer Science+Business Media, Inc., Boston, MA, 2005.

[210] Thomas Mailund. Beginning Data Science in R: Data Analysis, Visualization, and Modelling for the Data Scientist. Apress Media, LLC, Berkeley, CA, 2017.

[211] Gary Marrer. Fundamentals of Programming: With Object Orientated Programming. Laptop Press LLC, python edition, November 30 2009.

[212] Grant Martin and Henry Chang. Winning the SoC Revolution: Experiences in Real Design. Kluwer Academic Publishers, Norwell, MA, 2003.

[213] Robert C. Martin. Agile Software Development, Principles, Patterns, and Practices. Prentice-Hall, Upper Saddle River, NJ, 2002.

[214] Robert C. Martin. Clean Code: A Handbook of Agile Software Craftsmanship. Robert C. Martin. Pearson Education, Upper Saddle River, NJ, 2009.

[215] Robert C. Martin and Micah Martin. Agile Principles, Patterns, and Practices in C#. Robert C. Martin. Prentice Hall, Boston, MA, 2007.

[216] Jon Masters and Richard Blum. Professional Linux Programming. Programmer to Programmer™. Wiley Publishing, Indianapolis, IN, 2007.

[217] Steve McConnell. Rapid Development: Taming Wild Software Schedules. Microsoft Press, Redmond, WA, 1996.

[218] Steve McConnell. Code Complete: A Practical Handbook of Software Construction. Microsoft Press, Redmond, WA, second edition, 2004.

[219] Annabelle McIver and Carroll Morgan. Programming Methodology. Monographs in Computer Science. Springer-Verlag New York, New York, NY, 2003.

[220] Annabelle McIver and Carroll Morgan. Abstraction, Refinement and Proof for Probabilistic Systems. Monographs in Computer Science. Springer Science+Business Media, Inc., New York, NY, 2005.

[221] Paul McLellan. Agile IC development. Available online from *SemiWiki.com: Semiconductor Expert Forum* at: https://www.semiwiki.com/forum/content/3889-agile-ic-development.html; September 23, 2018 was the last accessed date, October 1 2014.

[222] Mercurial contributors. Learning Mercurial in workflows. Available online from the *Mercurial community: Mercurial SCM* at: https://www.mercurial-scm.org/guide; self-published; February 15, 2019 was the last accessed date.

[223] Mercurial contributors. Mercurial SCM. Available online from the *Mercurial community* at: https://www.mercurial-scm.org/; self-published; February 15, 2019 was the last accessed date.

[224] Mercurial contributors. A tutorial on using Mercurial. Available online from *Mercurial community: Mercurial SCM: Mercurial*

Wiki

at: https://www.mercurial-scm.org/wiki/Tutorial; self-published; February 15, 2019 was the last accessed date, September 2 2013.

[225] Mercurial contributors. Understanding mercurial. Available online from *Mercurial community: Mercurial SCM: Mercurial*

Wiki

at: https://www.mercurial-scm.org/wiki/UnderstandingMercurial; self-published; February 15, 2019 was the last accessed date, September 2 2013.

[226] Mercurial contributors. Mercurial

wiki

. Available online from *Mercurial community: Mercurial SCM* at: https://www.mercurial-scm.org/wiki/; self-published; February 15, 2019 was the last accessed date, February 1 2019.

[227] Gerard Meszaros. xUnit Test Patterns: Refactoring Test Code. Addison-Wesley Signature Series. Pearson Education, Boston, MA, 2007.

[228] Bertrand Meyer. Object-Oriented Software Construction. Prentice-Hall, Upper Saddle River, NJ, 1997.

[229] Ali Mili and Fairouz Tchier. Software Testing: Concepts and Operations. John Wiley & Sons, Hoboken, NJ, 2015.

[230] Mike Mintz and Robert Ekendahl. Hardware Verification with *SystemVerilog*: An Object-Oriented Framework. Springer Science+Business Media, LLC, New York, NY, 2007.

[231] Trevor Misfeldt, Gregory Bumgardner, and Andrew Gray. The Elements of C++ Style. Cambridge University Press, New York, NY, 2004.

[232] Jayadev Misra. A Discipline of Multiprogramming: Programming Theory for Distributed Applications. Monographs in Computer Science. Springer-Verlag New York, New York, NY, 2001.

[233] John C. Mitchell. Concepts in Programming Languages. Cambridge University Press, Cambridge, U.K., 2002.

[234] Antonio F. Mondragon-Torres. An agile embedded systems capstone course: Overview, experiences, and lessons learned. In Proceedings of the 43^{rd} IEEE Frontiers in Education Conference (FIE 2013), Oklahoma City, OK, October 23–26 2013. American Society for Engineering Education, Institute of Electrical and Electronics Engineers, and IEEE Computer Society, IEEE Press.

[235] Antonio F. Mondragon-Torres, Alexander Kozitsky, Clifford Bundick, Edward Mc Kenna Jr., Eric Alley, Matthew Lloyd, Peter Stanley, and Roger Lane. Work in progress – an agile embedded systems design capstone course. In Proceedings of the 41^{st} Annual ASEE/IEEE Frontiers in Education Conference (FIE 2011), pages F4F–1 – F4F–3, Rapid City, SD, October 12–15 2011. American Society for Engineering Education, Institute of Electrical and Electronics Engineers, and IEEE Computer Society, IEEE Press.

[236] Jean-François Monin and Michael G. Hinchey. Understanding Formal Methods. Springer-Verlag London, London, U.K., 2003.

[237] Daniel J. Mosley and Bruce A. Posey. Just Enough Software Test Automation. Just Enough, Yourdon Press Series. Prentice Hall, Upper Saddle River, NJ, 2002.

[238] Matthias Müller-Hannemann and Stefan Schirra. Algorithm Engineering: Bridging the Gap between Algorithm Theory and Practice, volume 5971 of Lecture Notes in Computer Science. Springer-Verlag Berlin Heidelberg, Heidelberg, Germany, 2010.

[239] Ware Myers. Why software developers refuse to improve. IEEE Computer, 31(4):110–112, April 1998.

[240] Mythic Inc. staff. Hardware devops engineer. Available online at: https://www.linkedin.com/jobs/view/hardware-devops-engineer-at-mythic-1078283501; September 10, 2019 was the last accessed date, December 2018.

[241] Huy Nguyen, Hai-My Nguyen, and Minh-Luan Tran. TexPaste: Pastebin for mathematicians. Available online at: http://www.texpaste.com; self-published; June 4, 2018 was the last accessed date, October 12 2013.

[242] Pierluigi Nuzzo and Alberto Sangiovanni-Vincentelli. Robustness in analog systems: Design techniques, methodologies and tools. In Proceedings of the 6^{th} IEEE International Symposium on Industrial Embedded Systems (SIES 2011), pages 194–203, Västerås, Sweden, June 15–17 2011. IEEE Press.

[243] Dan Olsen. The Lean Product Playbook: How to Innovate with Minimum Viable Products and Rapid Customer Feedback. John Wiley & Sons, Hoboken, NJ, 2015.

[244] Zhiyang Ong. Test automation for software quality assurance: A comparative study of hardware and software test automation. Available online at: https://sites.google.com/site/zhiyangong/zhiyang_internship_report2.pdf?attredirects=0; March 3, 2011 was the last accessed date, August 8 2006.

[245] Oracle Corporation staff. How to write Doc comments for the Javadoc tool. Available online from *Oracle Corporation: Oracle Technology Network: Recently Published Technical Articles: Articles About Java Technology* at: http://www.oracle.com/technetwork/articles/java/index-137868.html; January 30, 2017 was the last accessed date.

[246] Oracle Corporation staff. Javadoc tool. Available online from *Oracle Corporation: Oracle Technology Network: Recently Published Technical Articles: Articles About Java Technology* at: http://www.oracle.com/technetwork/articles/java/index-jsp-135444.html; January 30, 2017 was the last accessed date.

[247] Oracle Corporation staff. Javadoc tool. Available online from *Oracle Corporation: Oracle Technology Network: Oracle Technology Network for Java Developers: Java SE at a Glance: Java SE Documentation* at: http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html; January 30, 2017 was the last accessed date.

[248] Oracle Corporation staff. javadoc. Available online from *Oracle Corporation: Oracle Help Center: Java Platform, Standard Edition (Java SE) 8: Java Platform Standard Edition 8 Documentation: JDK Tools and Utilities: Java Platform, Standard Edition Tools Reference: §5 Create and Build Applications* at: https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html; January 30, 2017 was the last accessed date, January 2016.

[249] Andrew Oram and Steve Talbott. Managing Projects with make: C Programming Utility. O'Reilly & Associates, Sebastopol, CA, second edition, 1991.

[250] Andy Oram and Greg Wilson. Beautiful Code: Leading Programmers Explain How They Think. O'Reilly Media, Sebastopol, CA, 2007.

[251] Andy Oram and Greg Wilson. Making Software: What Really Works, and Why We Believe It. Theory in Practice. O'Reilly Media, Sebastopol, CA, 2011.

[252] Ted Osborne and Salas leVirve. Google Python style guide. Available online from *GitHub: Google: styleguide* at: http://google.github.io/styleguide/pyguide.html and https://github.com/google/styleguide/blob/gh-pages/pyguide.md; September 17, 2019 was the last accessed date, December 5 2018.

[253] Ted Osborne and Nodir Turakulov. Philosophy. Available online from *GitHub: Google: styleguide: Google documentation guide* at: https://github.com/google/styleguide/blob/gh-pages/docguide/philosophy.md; September 17, 2019 was the last accessed date, January 30 2018.

[254] Ted Osborne, Nodir Turakulov, and Ackermann Yuriy. Documentation best practices. Available online from *GitHub: Google: styleguide: Google documentation guide* at: https://github.com/google/styleguide/blob/gh-pages/docguide/best_practices.md; September 17, 2019 was the last accessed date, January 30 2018.

[255] Bryan O'Sullivan. Mercurial: The Definitive Guide. Modern Software for Collaboration. O'Reilly Media, Sebastopol, CA, 2009.

[256] Harry J. Paarsch and Konstantin Golyaev. A Gentle Introduction to Effective Computing in Quantitative Research: What Every Research Assistant Should Know. The MIT Press, London, U.K., 2016.

[257] Ashwin Pajankar. Python Unit Test Automation: Practical Techniques for Python Developers and Testers. Apress Media, LLC, Berkeley, CA, 2017.

[258] Oren Patashnik. BibTeXing. Available online at: http://mirrors.ctan.org/biblio/bibtex/base/btxdoc.pdf; September 24, 2014 was the last accessed date, February 8 1988.

[259] Benjamin C. Pierce. Types and Programming Languages. The MIT Press, Cambridge, MA, 2002.

[260] Benjamin C. Pierce, Arthur Azevedo de Amorim, Chris Casinghino, Marco Gaboardi, Michael Greenberg, Cătălin Hriţcu, Vilhelm Sjöberg, Brent Yorgey, Loris D'Antoni, Andrew W. Appel,

Arthur Charguéraud, Anthony Cowley, Jeffrey Foster, Dmitri Garbuzov, Michael Hicks, Ranjit Jhala, Greg Morrisett, Jennifer Paykin, Mukund Raghothaman, Chung-chieh Shan, Leonid Spesivtsev, Andrew Tolmach, Stephanie Weirich, and Steve Zdancewic. Software Foundations. University of Pennsylvania, Philadelphia, PA, Spring 2017.

[261] Hasso Plattner, Christoph Meinel, and Larry Leifer. Design Thinking: Understand – Improve – Apply. Understanding Innovation. Springer-Verlag Berlin Heidelberg, Heidelberg, Germany, 2011.

[262] Zissis Poulos and Andreas Veneris. Failure triage in RTL regression verification. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 37(9):1893–1906, September 2018.

[263] Zissis Poulos, Yu-Shen Yang, and Andreas Veneris. A failure triage engine based on error trace signature extraction. In Proceedings of the 15^{th} IEEE Symposium On-Line Testing (IOLTS 2013), pages 73–78, Chania, Crete, Greece, July 8–10 2013. IEEE Council on EDA (CEDA), and IEEE Computer Society Test Technology Technical Council (TTTC), IEEE Press.

[264] Zissis Poulos, Yu-Shen Yang, Andreas Veneris, and Bao Le. Simulation and satisfiability guided counter-example triage for RTL design debugging. In Proceedings of the 15^{th} IEEE International Symposium on Quality Electronic Design (ISQED 2014), pages 618–624, Santa Clara, CA, March 3–5 2014. IEEE Electron Devices Society (EDS), IEEE Reliability Society, IEEE Components, Packaging and Manufacturing Technology Society, and IEEE Circuits and Systems Society (CAS), IEEE Press.

[265] Zissis Paraskevas Poulos. High level debugging techniques for modern verification flows. Master's thesis, University of Toronto, Toronto, Canada, July 4 2014.

[266] Zissis Paraskevas Poulos. Scalable Inference in Hardware Verification and Social Graph Reasoning. PhD thesis, University of Toronto, Toronto, Canada, November 2018.

[267] Terrence W. Pratt and Marvin V. Zelkowitz. Programming Languages: Design and Implementation. Prentice Hall, Upper Saddle River, NJ, fourth edition, 2001.

[268] Christian R. Prause. Reputation-based self-management of software process artifact quality in consortium research projects. In Proceedings of the 19^{th} ACM SIGSOFT Symposium and the 13^{th} European Conference on Foundations of Software Engineering (ESEC/FSE'11), pages 380–383, Szeged, Hungary, September 5–9 2011. ACM Special Interest Group on Software Engineering (SIGSOFT), ACM Press.

[269] Jeff Preshing. C++ has become more Pythonic. Available online from *Preshing on Programming* at: http://preshing.com/20141202/cpp-has-become-more-pythonic/; February 1, 2018 was the last accessed date, December 2 2014.

[270] Roger S. Pressman. Software Engineering: A Practitioner's Approach. McGraw-Hill Series in Computer Science. McGraw-Hill, New York, NY, fifth edition, 2001.

[271] Roger S. Pressman. Software Engineering: A Practitioner's Approach. McGraw-Hill, New York, NY, seventh edition, 2010.

[272] Programming Research Ltd. staff. High integrity C++: Coding standard version 4.0. Technical report, Programming Research Ltd., Hersham, Surrey, England, U.K., October 3 2013.

[273] Ken Pugh. Lean-Agile Acceptance Test-Driven Development: Better Software Through Collaboration. Net Objectives Lean-Agile. Addison-Wesley, Boston, MA, 2011.

[274] Ben Rady and Rod Coffin. Continuous Testing: With Ruby, Rails, and JavaScript. Pragmatic Bookshelf. The Pragmatic Programmers, Raleigh, NC, 2011.

[275] Prakash Rashinkar, Peter Paterson, and Leena Singh. System-on-a-Chip Verification: Methodology and Techniques. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

[276] Jonathan Rasmusson. The Agile Samurai: How Agile Masters Deliver Great Software. Pragmatic Bookshelf. The Pragmatic Programmers, Raleigh, NC, September 15 2010.

[277] Eric S. Raymond. The Art of UNIX Programming. Addison-Wesley Professional Computing Series. Pearson Education, Boston, MA, 2004.

[278] Robert L. Read. An essay on how to be a programmer. Available online at: http://samizdat.mines.edu/howto/HowToBeAProgrammer.html and http://samizdat.mines.edu/howto/HowToBeAProgrammer.pdf; April 2, 2014 was the last accessed date, 2003.

[279] Martin Reddy. API Design for C++. Morgan Kaufmann, Burlington, MA, 2011.

[280] Kenneth Reitz. Code style. Available online from *The Hitchhiker's Guide to Python* at: http://docs.python-guide.org/en/latest/writing/style/; self-published; February 1, 2018 was the last accessed date, 2016.

[281] Kenneth Reitz and Tanya Schlusser. The Hitchhiker's Guide to Python: Best Practices for Development. O'Reilly Media, Sebastopol, CA, 2016.

[282] John C. Reynolds. Theories of Programming Languages. Cambridge University Press, New York, NY, 1998.

[283] Leanna Rierson. Developing Safety-Critical Software: A Practical Guide for Aviation Software and DO-178C Compliance. CRC Press, Boca Raton, FL, 2013.

[284] Chris Riesbeck. Useful C++ / Unix resources. Available online from *Prof. Chris Riesbeck's web page: Programming*, Computer Science Division, Department of Electrical Engineering and Computer Science, Robert R. McCormick School of Engineering and Applied Science, Northwestern University at: http://www.cs.northwestern.edu/~riesbeck/programming/c++/; September 30, 2015 was the last accessed date, July 2 2009.

[285] Kristian Rother. Pro Python Best Practices: Debugging, Testing and Maintenance. Apress Media, LLC, Berkeley, CA, 2017.

[286] Johanna Rothman. Manage It!: Your Guide to Modern, Pragmatic Project Management. Pragmatic Bookshelf. The Pragmatic Programmers, Raleigh, NC, 2007.

[287] Johanna Rothman. Manage Your Project Portfolio: Increase Your Capacity and Finish More Projects. Pragmatic Bookshelf. The Pragmatic Programmers, Raleigh, NC, 2009.

[288] Johanna Rothman. Manage Your Project Portfolio: Increase Your Capacity and Finish More Projects. The Pragmatic Programmers, Raleigh, NC, second edition, 2016.

[289] Johanna Rothman. Create Your Successful Agile Project: Collaborate, Measure, Estimate, Deliver. The Pragmatic Programmers, Raleigh, NC, 2017.

[290] RunMyCode Association contributors. RunMyCode. Available online at: http://www.runmycode.org/; November 13, 2016 was the last accessed date, 2013.

[291] Sean A. Safarpour. Formal Methods in Automated Design Debugging. PhD thesis, University of Toronto, Toronto, Canada, 2009.

[292] David Sale. Testing Python: Applying Unit Testing, TDD, BDD and Acceptance Testing. John Wiley & Sons, Chichester, West Sussex, South East England, England, U.K., 2014.

[293] Resve Saleh, Steve Wilton, Shahriar Mirabbasi, Alan Hu, Mark Greenstreet, Guy Lemieux, Partha Pratim Pande, Cristian Grecu, and Andre Ivanov. System-on-chip: Reuse and integration. Proceedings of the IEEE, 94(6):1050–1069, June 2006.

[294] Miro Samek. Practical UML Statecharts in C/C++: Event-Driven Programming for Embedded Systems. Newnes, Burlington, MA, second edition, 2009.

[295] Alberto Sangiovanni-Vincentelli. Quo vadis, SLD? reasoning about the trends and challenges of system level design. Proceedings of the IEEE, 95(3):467–506, March 2007.

[296] Alberto Sangiovanni-Vincentelli. Is a unified methodology for system-level design possible? IEEE Design & Test of Computers, 25(4):346–357, July–August 2008.

[297] Alberto Sangiovanni-Vincentelli and Grant Martin. Platform-based design and software design methodology for embedded systems. IEEE Design & Test of Computers, 18(6):23–33, November–December 2001.

[298] Sergio Saponara, Pierluigi Nuzzo, Paolo D'Abramo, and Luca Fanucci. EURASIP Journal on Embedded Systems: Design Methodologies and Innovative Architectures for Mixed-Signal Embedded Systems. Hindawi Publishing Corporation, Cairo, Egypt, 2010.

[299] Peter Savage. Git in the trenches. Available online at: http://cbx33.github.io/gitt/, http://cbx33.github.io/gitt/download.html, and https://github.com/cbx33/gitt; self-published; August 1, 2014 was the last accessed date, August 2011.

[300] Stephen R. Schach. Object-Oriented and Classical Software Engineering. McGraw-Hill, New York, NY, fifth edition, 2002.

[301] Stephen R. Schach. Object-Oriented and Classical Software Engineering. McGraw-Hill, New York, NY, seventh edition, 2007.

[302] Stephen R. Schach. Object-Oriented and Classical Software Engineering. McGraw-Hill, New York, NY, eighth edition, 2011.

[303] Quirin Schiermeier. Data management made simple. Nature, 555(7697):403–405, March 13 2018.

[304] Anna Schneider. How to become a data scientist before you graduate. Berkeley Science Review, July 30 2013.

[305] Helmut Schwichtenberg and Ralf Steinbrüggen. Proceedings of the NATO Advanced Science Institute on Proof and System-Reliability, volume 62 of NATO Science Series II: Mathematics, Physics and Chemistry. Kluwer Academic Publishers, Marktoberdorf, Germany, July 24 – August 5 2002.

[306] Science Infusion Software Engineering Process Group staff. Science infusion software engineering process group (SISEPG) C++ programming standards and guidelines. Available online as Version 1.11 from *United States Department of Commerce: National Oceanic and Atmospheric Administration: National Weather Service: Office of Hydrologic Development, now known as the National Water Center: Hydrology Laboratory* at: http://www.nws.noaa.gov/oh/hrl/developers_docs/C++_Software_Standards.pdf; October 27, 2015 was the last accessed date, November 17 2006.

[307] Michael L. Scott. Programming Language Pragmatics. Morgan Kaufmann, Burlington, MA, third edition, 2009.

[308] Peter Sestoft. Programming Language Concepts, volume 50 of Undergraduate Topics in Computer Science. Springer-Verlag London, London, U.K., 2012.

[309] James Shore and Shane Warden. The Art of Agile Development. Theory in Practice. O'Reilly Media, Sebastopol, CA, 2008.

[310] Leena Singh, Leonard Drucker, and Neyaz Khan. Advanced Verification Techniques: A SystemC Based Approach for Successful Tapeout. Kluwer Academic Publishers, Boston, MA, 2004.

[311] Kevin A. Smith and Doug Kramer. Requirements for writing Java API specifications. Available online from *Oracle Corporation: Oracle Technology Network: Oracle Technology Network for Java Developers: Java SE at a Glance: Java SE Documentation* at: http://www.oracle.com/technetwork/java/javase/documentation/index-142372.html; January 30, 2017 was the last accessed date, January 2003.

[312] Ian Sommerville. Software Engineering. Pearson Education, Harlow, Essex, East of England, England, U.K., eighth edition, 2007.

[313] Joel Spolsky. Joel on Software: And on Diverse and Occasionally Related Matters That Will Prove of Interest to Software Developers, Designers, and Managers, and to Those Who, Whether by Good Fortune or Ill Luck, Work with Them in Some Capacity. Apress, Berkeley, CA, 2004.

[314] Joel Spolsky. More Joel on Software: Further Thoughts on Diverse and Occasionally Related Matters That Will Prove of Interest to Software Developers, Designers, and Managers, and to Those who, Whether by Good Fortune or Ill Luck, Work with Them in Some Capacity. Apress, Berkeley, CA, 2008.

[315] Richard Stallman and Bob Chassell. GNU Texinfo 6.3. Available online from *The GNU Operating System and the Free Software Movement: GNU Software: Texinfo – The GNU Documentation System: Texinfo manuals: GNU Texinfo manual* at: https://www.gnu.org/software/texinfo/manual/texinfo/texinfo.html; November 4, 2016 was the last accessed date, September 11 2016.

[316] Richard Stallman and Bob Chassell. GNU Texinfo manual. Available online from *The GNU Operating System and the Free Software Movement: GNU Software: Texinfo – The GNU Documentation System: Texinfo manuals* at: https://www.gnu.org/software/texinfo/manual/texinfo/; November 4, 2016 was the last accessed date, September 11 2016.

[317] Richard Stallman and Bob Chassell. Texinfo: The GNU documentation system. Available online from *The GNU Operating System and the Free Software Movement: GNU Software* at: https://www.gnu.org/software/texinfo/; November 4, 2016 was the last accessed date, September 11 2016.

[318] Mircea R. Stan, Paul D. Franzon, Seth Copen Goldstein, John C. Lach, and Matthew M. Ziegler. Molecular electronics: From devices and interconnect to circuits and architecture. Proceedings of the IEEE, 91(11):1940–1957, November 2003.

[319] Andrew Stellman and Jennifer Greene. Learning Agile: Understanding Scrum, XP, Lean, and Kanban. O'Reilly Media, Sebastopol, CA, 2015.

[320] Chris Sterling. Managing Software Debt: Building for Inevitable Change. The Agile Software Development. Addison-Wesley, Boston, MA, 2011.

[321] Perdita Stevens and Rob Pooley. Using UML: Software Engineering with Objects and Components. Addison-Wesley Object Technology Series. Addison-Wesley, Essex, England, U.K., 2000.

[322] Thomas Stober and Uwe Hansmann. Agile Software Development: Best Practices for Large Software Development Projects. Springer-Verlag Berlin Heidelberg, Heidelberg, Germany, 2010.

[323] Victoria Stodden, Friedrich Leisch, and Roger D. Peng. Implementing Reproducible Research. Chapman & Hall/CRC The R Series. CRC Press, Boca Raton, FL, 2014.

[324] Venkat Subramaniam and Andy Hunt. Practices of an Agile Developer: Working in the Real World. The Pragmatic Programmers, Raleigh, NC, 2006.

[325] Joshua Suereth and Matthew Farwell. sbt in Action: The simple Scala build tool. Manning Publications Co., Shelter Island, NY, 2016.

[326] Xuening Sun. Compositional Design of Analog Systems Using Contracts. PhD thesis, University of California, Berkeley, Berkeley, CA, May 12 2011.

[327] Travis Swicegood. Pragmatic Guide to Git. Pragmatic Bookshelf. The Pragmatic Programmers, Raleigh, NC, November 1 2010.

[328] Alexander Tarlinder. Developer Testing: Building Quality into Software. Addison-Wesley Signature Series. Addison-Wesley, Boston, MA, 2017.

[329] Risk Danger technoweenie Olson, Brandon Keepers, Anatoly Borodin, Lucas Werkmeister, Steve Streeting, and Scott Barron. Git large file storage. Available online at: https://git-lfs.github.com/ and https://github.com/github/git-lfs; April 1, 2016 was the last accessed date, March 31 2016.

[330] The SciPy community. NumPy reference. Available online from *SciPy.org: Numpy and Scipy Documentation: NumPy v1.17 Manual* as release 1.17 at: https://docs.scipy.org/doc/numpy/reference/; self-published; January 20, 2020 was the last accessed date, July 26 2019.

[331] Eno Thereska. C++ coding standard. Available online from *Eno Thereska's home page: Department of Electrical and Computer Engineering, College of Engineering, Carnegie Mellon University* at: https://users.ece.cmu.edu/~eno/coding/CppCodingStandard.html; February 16, 2016 was the last accessed date.

[332] Shyamkumar Thoziyoor, Jay Brockman, and Daniel Rinzler. Pim lite: a multithreaded processor-in-memory prototype. In Proceedings of the 15^{th} ACM Great Lakes Symposium on VLSI (GLSVLSI '05), pages 64–69, Chicago, IL, April 17–19 2005. Association for Computing Machinery, ACM Press.

[333] Adam Tornhill. Software Design X-Rays: Fix Technical Debt with Behavioral Code Analysis. The Pragmatic Programmers, Raleigh, NC, 2018.

[334] Allen B. Tucker and Robert E. Noonan. Programming Languages: Principles and Paradigms. McGraw-Hill, New York, NY, second edition, 2007.

[335] Nodir Turakulov, Ted Osborne, and Ackermann Yuriy. Markdown style guide. Available online from *GitHub: Google: styleguide: Google documentation guide* at: https://github.com/google/styleguide/blob/gh-pages/docguide/style.md; September 17, 2019 was the last accessed date, January 30 2018.

[336] Greg L. Turnquist. Python Testing Cookbook: Over 70 simple but incredibly effective recipes for taking control of automate testing using powerful Python testing tools. Quick answers to common problems. Packt Publishing, Birmingham, West Midlands, West Midlands, England, U.K., 2011.

[337] Vijay K. Vaishnavi and William Kuechler Jr. Design Science Research Methods and Patterns: Innovating Information and Communication Technology. Auerbach Publications, Boca Raton, FL, 2008.

[338] Joseph S. Valacich, Joey F. George, and Jeffrey A. Hoffer. Essentials of Systems Analysis and Design. Pearson Education, Upper Saddle River, NJ, fifth edition, 2012.

[339] Jonathan W. Valvano. Embedded Microcomputer Systems: Real Time Interfacing. Cengage Learning, Stamford, CT, third edition, 2007.

[340] Dimitri van Heesch. Doxygen. Available online at M.C.G.V. Stack: http://www.stack.nl/~dimitri/doxygen/ and http://doxygen.org/; November 4, 2016 was the last accessed date, September 11 2016.

[341] Guido van Rossum, Barry Warsaw, and Nick Coghlan. Style guide for Python code. Available online from *Python: Python Developer's Guide: Python Enhancement Proposals: Python Enhancement Proposal 0* at: https://www.python.org/dev/peps/pep-0008/; October 20, 2015 was the last accessed date, August 1 2013.

[342] Doug Vucevic and Wayne Yaddow. Testing the Data Warehouse Practicum: Assuring Data Content, Data Structures and Quality. Trafford Publishing, Bloomington, IN, 2012.

[343] Chao Wang. Abstraction Refinement for Large Scale Model Checking. PhD thesis, University of Colorado at Boulder, Boulder, CO, 2004.

[344] Chao Wang, Gary D. Hachtel, and Fabio Somenzi. Abstraction Refinement for Large Scale Model Checking. Integrated Circuits and Systems. Springer Science+Business Media, LLC, New York, NY, 2006.

[345] David A. Watt. Programming Language Design Concepts. John Wiley & Sons, Chichester, West Sussex, South East England, England, U.K., 2004.

[346] John L. Weatherwax. Solution manuals and various book notes. Available online at: http://waxworksmath.com/ce_solutionmanuals.asp; self-published; May 31, 2015 was the last accessed date, 2008.

[347] James A. Whittaker. Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design. Addison-Wesley, Boston, MA, 2009.

[348] Andreas Wieferink, Heinrich Meyr, and Rainer Leupers. Retargetable Processor System Integration into Multi-Processor System-on-Chip Platforms. Springer Science+Business Media B.V., Dordrecht, The Netherlands, 2008.

[349] Karl Wiegers. Peer Reviews in Software: A Practical Guide. Addison-Wesley Information Technology Series. Addison-Wesley, Reading, MA, 2002.

[350] Karl Wiegers and Joy Beatty. Software Requirements. Best Practices. Microsoft Press, Redmond, WA, third edition, 2013.

[351] Wikibooks contributors. Exception handling. In C++ Programming, chapter 4. Wikimedia Foundation, San Francisco, CA, February 11 2016.

[352] Wikipedia contributors. Exception safety. Available online from *Wikipedia, The Free Encyclopedia: CATEGORY* at: https://en.wikipedia.org/wiki/Exception_safety; October 29, 2016 was the last accessed date, September 26 2016.

[353] Wikipedia contributors. Indent style. Available online in *Wikipedia, The Free Encyclopedia: Source code* at: https://en.wikipedia.org/wiki/Indent_style; November 4, 2016 was the last accessed date, October 30 2016.

[354] Wikipedia contributors. Natural language programming. Available online in *Wikipedia, The Free Encyclopedia: Computer programming* at: https://en.wikipedia.org/wiki/Natural_language_programming; November 4, 2016 was the last accessed date, November 2 2016.

[355] Wikipedia contributors. Semipredicate problem. Available online from *Wikipedia, The Free Encyclopedia: Programming language topics* at: https://en.wikipedia.org/wiki/Semipredicate_problem; October 29, 2016 was the last accessed date, October 23 2016.

[356] Wikipedia contributors. Texinfo. Available online in *Wikipedia, The Free Encyclopedia: Markup languages* at: https://en.wikipedia.org/wiki/Texinfo; November 4, 2016 was the last accessed date, April 29 2016.

[357] Wikipedia contributors. Naming convention (programming). Available online from *Wikipedia, The Free Encyclopedia: Source code* at: https://en.wikipedia.org/wiki/Naming_convention_(programming); March 10, 2017 was the last accessed date, February 26 2017.

[358] Wikipedia contributors. Programming paradigm. Available online from *Wikipedia, The Free Encyclopedia: Programming paradigms* at: https://en.wikipedia.org/wiki/Programming_paradigm; July 12, 2017 was the last accessed date, July 5 2017.

[359] Wikipedia contributors. Code refactoring. Available online from *Wikipedia, The Free Encyclopedia: 21st-century revolutions* at: https://en.wikipedia.org/wiki/Code_refactoring; September 25, 2018 was the last accessed date, September 20 2018.

[360] Ron Wilson. Agile development in the SoC design world? Available online from *Embedded.com: Insights: Agile Collection* at: https://www.embedded.com/electronics-blogs/agile-collection/4218869/Agile-development-in-the-SoC-design-world-; September 25, 2018 was the last accessed date, August 17 2011.

[361] Glynn Winskel. The Formal Semantics of Programming Languages: An Introduction. The MIT Press, Cambridge, MA, 1993.

[362] Clifford Wolf. Clifford on programming style. Available online from *Clifford Wolf's Personal Homepage* at: http://www.clifford.at/style.html; self-published; March 14, 2016 was the last accessed date.

[363] Marty J. Wolf. The acm code of ethics: a call to action. Communications of the ACM, 59(12):6, December 2016.

[364] Jeff Younker. Foundations of Agile Python Development. Apress, Berkeley, CA, 2008.

[365] Andreas Zeller. Why Programs Fail: A Guide to Systematic Debugging. Morgan Kaufmann, Burlington, MA, second edition, 2009.

[366] Kaiping Zeng and Sorin A. Huss. RAMS: A VHDL-AMS code refactoring tool supporting high level analog synthesis. In Proceedings of the 6^{th} IEEE Computer Society Annual Symposium on VLSI: New Frontiers in VLSI Design (ISVLSI'05), Tampa, FL, May 11–12 2005. IEEE Computer Society, IEEE Press.

[367] Kaiping Zeng and Sorin A. Huss. Architecture refinements by code refactoring of behavioral VHDL-AMS models. In Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS 2006), pages 1475–1478, Island of Kos, Greece, May 21–24 2006. IEEE Press.

[368] Naijun Zhan, Shuling Wang, and Hengjun Zhao. Formal Verification of Simulink/Stateflow Diagrams: A Deductive Approach. Springer International Publishing AG, Cham, Switzerland, 2017.

[369] Mark Zuckerberg. Letter from Mark Zuckerberg. Available online as *Form S-1 REGISTRATION STATEMENT Under The Securities Act of 1933, Facebook, Inc.* at: http://www.sec.gov/Archives/edgar/data/1326801/000119312512034517/d287954ds1.htm#toc287954_10; October 9, 2014 was the last accessed date, February 1 2012.