# Iterative Filtering of Retrieved Information to Increase Relevance

Robert Zeidman
President, Zeidman Consulting
15565 Swiss Creek Lane
Cupertino, CA 95014 USA
+1.408.741.5809

Bob@ZeidmanConsulting.com

## ABSTRACT

Efforts have been underway for years to find more effective ways to retrieve information from large knowledge domains. This effort is now being driven particularly by the Internet and the vast amount of information that is available to unsophisticated users. In the early days of the Internet, some effort involved allowing users to enter Boolean equations of search terms into search engines, for example, rather than just a list of keywords. More recently, effort has focused on understanding a user's desires from past search histories in order to narrow searches. Also there has been much effort to improve the ranking of results based on some measure of relevancy. This paper discusses using iterative filtering of retrieved information to focus in on useful information. This work was done for finding source code correlation and the author extends his findings to Internet searching and e-commerce. The paper presents specific information about a particular filtering application and then generalizes it to other forms of information retrieval.

## Keywords

Correlation, E-Commerce, Filtering, Information Retrieval, Intellectual Property, Internet, Search Engine, Source Code, World Wide Web.

## 1. INTRODUCTION

Years ago I heard a story about the Department of Defense contracting a software company, in a time of war, to create a program to automatically translate from the enemy's language to English. The software went through rigorous testing and numerous revisions, but because of idioms and slang and the usual ambiguities of human languages, the program could only correctly interpret about 80% of what was input. The DOD eventually gave up on the project because it couldn't afford to misinterpret 20% of all enemy correspondences and so it continued to translate documents manually. That is until one clever person realized that a human reviewing automatically translated documents for errors was able to correct those errors in a fraction of the time it would take to translate entire documents manually. Just because the program was not 100% effective did not mean it was not effective at all.

The lesson here is that human interaction should not be discounted. Sources of electronic information are becoming more widely available, and non-technical users are required to access that information. Sophisticated algorithms for automatically figuring out a user's requirements and retrieving information are being developed and improved, but it is doubtful that these algorithms will ever be 100% effective. Filtering the information, as defined in this paper, refers to the process of refining retrieved information to eliminate less relevant results. Allowing a user to iteratively filter the information until the results are manageable is an important process. This paper defines the process and offers examples of current work and areas for future research.

## 2. CODEMATCH

I have been working for the past decade as an expert witness in intellectual property cases and have been asked on many occasions to examine software source code from a plaintiff and a defendant to determine whether one has plagiarized (stolen) code from the other. I found that the few existing tools for "plagiarism detection"[1] were too inaccurate for a situation where hundreds of millions of dollars could be at stake. So I developed my own tool called CodeMatch™.

After using CodeMatch on a number of cases, I found that it shared a deficiency with the other tools. Once two sets of software source code files were compared, and results were presented to the user, those results could not be further refined. Because a large comparison could take up to a week to deliver results, it was impractical to rerun the comparison using new settings. It was practical however to filter the results to obtain a more manageable and more relevant set of results to examine.

The operation of CodeMatch is briefly described below as well as the problems that arose and how those problems were solved using filtering.

---

[1] Prior to CodeMatch there were generally two categories of tools, "plagiarism detection" and "clone detection," that each were looking for software similarities but with two different purposes. CodeMatch is actually a more general "Source code correlation tool" that can be used to find plagiarism or clones or other kinds of similarities. See Zeidman [1] for more information.

## 2.1 Source Code Correlation Algorithms

CodeMatch compares a set of source code files, set 1, to another set of source code files, set 2, by first dividing the source code into elements defined in the following table.

| Software source code elements | Description |
|---|---|
| Statements | Cause actions to occur. Sequence dependent. |
| Instructions | Signify the actions to take place. |
| Control words | Control the program flow (e.g. "if", "case", "goto", "loop"). |
| Operators | Manipulate data (e.g., +, -. *. /). |
| Identifiers | Reference code or data. |
| Variables | Identify data. |
| Constants | Identify constants. |
| Functions | Identify code. |
| Labels | Specify locations in the program. |
| Comments | For documentation. Cause no actions to occur. |

**Table 1. Source code structure**

The program then compares these elements separately for each source code file and determines a correlation for each element and a source code correlation $\rho$ according to the formula below:

$$\rho = k_i\rho_i + k_s\rho_s + k_c\rho_c + k_q\rho_q \qquad (1)$$

where

- $\rho_s$ = statement correlation
- $k_s$ = the weight given to the statement correlation
- $\rho_c$ = comment correlation
- $k_c$ = the weight given to the comment correlation
- $\rho_i$ = identifier correlation
- $k_i$ = the weight given to the identifier correlation
- $\rho_q$ = instruction sequence correlation
- $k_q$ = the weight given to instruction sequence correlation

CodeMatch uses five algorithms to determine the correlation for each of these correlations. These algorithms are:

- Statement Matching: the number of identical statements after certain basic transformations have been made. Used to determine statement correlation.
- Comment Matching: the number of identical comments after certain basic transformations have been made. Used to determine comment correlation.
- Identifier Matching: the number of identical identifiers. Used to determine identifier correlation.
- Partial Identifier Matching: the number of identical identifiers. Used to determine identifier correlation.
- Instruction sequences: the longest sequence of identical instructions. Used to determine instruction sequence correlation.

## 2.2 Presentation of Results

For each file in set 1, the files in set 2 are listed along with their correlation scores in order of highest to lowest score as shown in Figure 1. By clicking on the correlation score for any pair of files, a detailed report is brought up showing the different algorithms that were run and which specific elements of each file had correlation. For example, if the comment correlation is nonzero, the detailed report would show all matching comments in the two files.

**D:\CodeSuite\Code Development\test\C\files 1\aaa.c**

| Score | Compared To File |
|---|---|
| 100 | D:\CodeSuite\Code Development\test\C\files 2\aaa.c |
| 12 | D:\CodeSuite\Code Development\test\C\files 2\bpf_dump_strings.c |
| 12 | D:\CodeSuite\Code Development\test\C\files 2\semicolon_test.c |
| 12 | D:\CodeSuite\Code Development\test\C\files 2\test\bpf_image.c |

**Figure 1. CodeMatch report**

## 2.3 Superfluous Results

In reviewing the results of the comparison, often some specific files or specific source code elements would show up throughout the results, skewing the results and hiding the important correlation information. For example, open source files may have been used in one or both sets of files. In searching for plagiarized code, the open source files would be highly correlated with each other, but these correlations were not important.

Similarly, there are specific statements, comments, and identifiers that can be found in files that increase the correlation but are not relevant to finding plagiarized code though they may be relevant to finding other kinds of correlation like code clones. A user searching for plagiarized code may find that two programs running on the Microsoft Windows operating system both use the same system calls. Thus files with these system calls will have a higher correlation but for reasons that are unimportant to finding plagiarism.

Had these results been known up front, some of them could have been eliminated before the correlation was calculated. However, given the number of files and the number of source code elements, it was impractical to find these elements before performing the correlation. Also the correlation itself pointed out many of these superfluous elements.

## 2.4 CodeMatch Post-Process Filtering

In order to make examination of the correlation results more useful, and to allow the user to focus in on the kinds of correlation that is most important, I added the ability to filter the results. After CodeMatch produces a database of results, the following filtering can be performed on the database.

- Statement filtering: A list of statements is created by the user. Any correlation due to a statement on this list is eliminated and the statement correlation score is decremented appropriately and the overall correlation score is decremented appropriately.
- Comment filtering: A list of comments is created by the user. Any correlation due to a comment on this list is eliminated and the comment correlation score is decremented appropriately and the overall correlation score is decremented appropriately.
- Identifier filtering: A list of identifiers is created by the user. Any correlation due to an identifier on this list is eliminated and the identifier correlation score is decremented appropriately and the overall correlation score is decremented appropriately.
- General file filtering: A list of file names is created by the user. Any correlation between any file whose name appears on the list and any other file is removed from the results database.
- Specific file filtering: A list of file names with absolute folder paths is created by the user. Any correlation between a specific file on the list and any other file is removed from the results database.
- Folder filtering: A list of folders is created by the user. Any correlation between a file in a folder on the list and any other file is removed from the results database.
- Threshold filtering: The user can change threshold parameters, reducing the number of correlated file pairs that are displayed. The user can set minimum and maximum correlation scores to display and can set a maximum number of correlated files to display for each file in set 1.

After the filtering is performed on the database, new correlation scores are computed between file pairs. This affects the displayed output because ranking can change after filtering. It was found that for large file sets this filtering reduced the manual process of reviewing the results in order to find plagiarized source code files from days to hours or even minutes.

# 3. POST-PROCESS FILTERING

My experience with CodeMatch can be generalized to any kind of information retrieval process.

## 3.1 Information Retrieval Process

Information retrieval starts with an information domain. This information domain can be a well-organized, categorized domain such as a database or it can be a disorganized, uncategorized domain such as the Internet. In all cases, information retrieval has been classified into two types – "exact match" and "best match."

### *3.1.1 Exact match*

The "exact match" type of information retrieval is represented by the Boolean retrieval method used by database queries and Internet search engines. In these cases, Boolean equations of keywords are entered by a user and all objects in the information domain (HTML pages in the case of the Web), that meet the criteria are retrieved for the user. Even the more sophisticated search engines that allow a user to input natural language queries are typically parsing the language to retrieve the keywords and Boolean equations.

### *3.1.2 Best match*

The "best match" category of information retrieval uses vector space and probabilistic retrieval methods that essentially try to understand what information a user wants, sometimes based on past searches or other stored user parameters, then present the information to the user that is deemed closest to what the user desires. An example of this would be the book suggestions that Amazon.com presents to customers based on their search criteria and their past searches. A more detailed description of retrieval models and retrieval classification can be found in Salton & McGill [17].

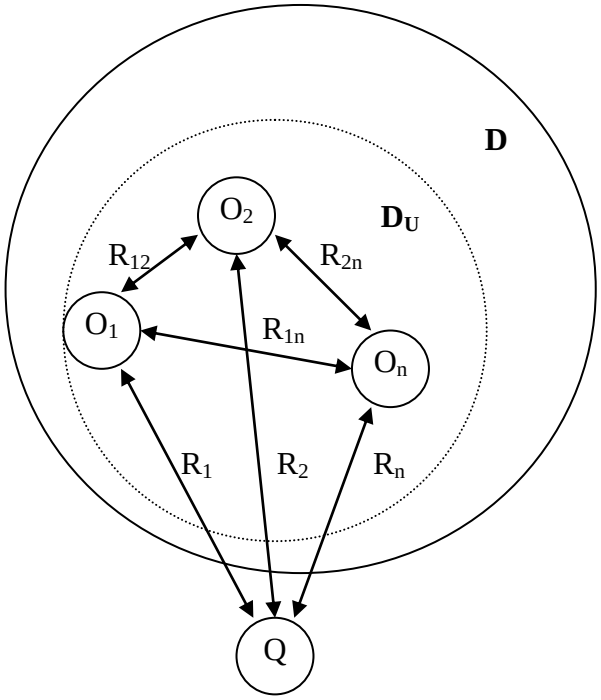


**Figure 2. Information retrieval**

### *3.1.3 Query-object relationships*

A representation of information retrieval is shown in Figure 2 where D is the information domain, and Q is the user's query. $D_U$ is the subset of the domain that meets the user's information need based on the retrieval process. Each arrow from an object to the query represents the relationship $R_i$ between the query and the object. For all retrieval methods, $D_U$ is the set of all objects such that $R_i > 0$.

$$D_U = \{Oi : R_i > 0 \text{ for all } i\} \quad \textbf{(1)}$$

**Equation 1. User information domain**

For a Boolean retrieval method, the $R_i$ is a scalar 1 for all i. In other words, a Boolean retrieval only retrieves objects that exactly match the query.

$$R_i = 1 \text{ for all } i \quad \textbf{(2)}$$

**Equation 2. Query-object relationship for Boolean retrieval**

For a probabilistic retrieval method, $R_i$ equals $P(Q|O_i)$, which is the probability that a user's query is met by retrieved object $O_i$.

$R_i = P(Q|O_i)$ for all i **(3)**

**Equation 3. Query-object relationship for probabilistic retrieval**

For a vector space retrieval method, $R_i$ equals $\rho(Q,Oi)$, which is the correlation between a user's query and object $O_i$. The correlation may be a dot product of vectors, representing the distance between the vectors in a multidimensional space, or it may be the cosine similarity measure, representing the angle between the vectors (see Letsche & Berry [10] for further details). Note that the correlation determined by CodeMatch is a form of vector space retrieval (a future paper will expound on that).

$R_i = \rho(Q,O_i)$ for all i **(4)**

**Equation 4. Query-object relationship for vector space retrieval**

In addition to relationships between a query and the retrieved objects, note that there are relationships between various retrieved objects, represented by the arrows $R_{ik}$ between objects $O_i$ and $O_k$. We make use of this fact for post-process filtering, discussed later.

## 3.2 Information Display Process

Once the information objects are retrieved from the domain, they must be displayed to the user. There are two types of criteria that can be used for this display. "Internal criteria" are criteria derived from the relationships determined during the retrieval process. "External criteria" are criteria determined in a new step unrelated to the retrieval process. Of course, combinations of internal and external criteria can also be used.

### *3.2.1 Internal display ranking criteria*

For best match retrieval methods, the objects can be displayed in order according to their relationship to the query. Objects with higher probabilities or higher correlation values are displayed first. The relationships are used as the criteria for displaying the objects. For exact match retrieval methods, internal criteria do not provide a way to display the results because all retrieved objects have a relationship of 1.

### *3.2.2 External display ranking criteria*

External criteria are often used to display the results. Perhaps the best known example of external display criteria is the PageRank method used by Google (see Brin & Page [2]).

$P_i = (1\text{-}d) + d(\Sigma(P_k/L_k)$ for $1 \le k \le n, k \ne i$ **(5)**

**Equation 5. Google PageRank algorithm**

$P_i$ is the PageRank value of page $O_i$. $L_k$ is the number of external links on page $O_k$ that point to other pages. Pages $O_1$ through $O_n$ are all of the pages that have links to page $O_i$. The parameter d is a "damping factor" that Google claims to typically set to 0.85. The PageRanks effectively form a probability distribution such that the sum of all web page PageRanks will be 1.

The Google ranking method is only one particular method. Other methods include the "Hub-Threshold Kleinberg" algorithm (see Kleinberg [10]). I represent ranking methods generally using the term $P_i$.

### *3.2.3 Display threshold*

Regardless of which kind of ranking criteria is used, there is often also a display threshold. Retrieved objects that have a ranking below the display threshold are not shown to the user. An object with a very low ranking is thought to be irrelevant and its relationship with the query is thought to be random rather than due to any relationship that would be significant to the user.

## 3.3 Post-retrieval Filtering

What I propose is another step after retrieval and display to further refine the results and reduce the number of retrieved objects to one that is reasonable to examine. There are several ways this can be accomplished using combinations of "object filters," "new query filters," "negative query filters," "threshold filters," and "object relationship filters."

### *3.3.1 Object filter*

Object filtering is the process of allowing the user to eliminate individual objects or whole sets of objects from the user information domain $D_U$. This can be done by allowing the user to specify objects to remove or categories of objects to remove. The removal process is dependent on the type of information being retrieved. When the retrieved objects are files, the criteria used to remove objects might be file name, location (path name), size, modification date, or creation date. With regard to CodeMatch, the general file filtering, specific file filtering, and folder filtering are examples of object filtering.

### *3.3.2 New query filter*

New query filtering refers to using a new query on the retrieved user domain $D_U$ to create a new domain $D_U'$ that is a subset of $D_U$. Some search engines provide this kind of filtering by allowing the user to further search the retrieved results with a new Boolean expression of keywords.

### *3.3.3 Negative query filter*

Negative query filtering refers to applying a query to the retrieved information in order to eliminate objects. For instance, suppose the original query is a Boolean query to find all documents with the phrases "software" "source code" and "correlation." A query-based elimination filter would one where the user eliminates all objects within $D_U$ that contain the keyword "correlation." This would be equivalent to an original query to find all documents with the phrases "software" and "source code" but not "correlation." However, there are two reasons that negative query filtering is useful. First, retrieval of objects from the original domain D based on the new query will require more resources (compute power, storage space, network bandwidth) than a negative query filter performed on the much small domain $D_U$. Second, if the query is a best match query rather than an exact match query, the query-based elimination filter can be used to get results that may be difficult for the user to define with a single query to the original domain.

With regard to CodeMatch, the identifier filtering, statement filtering, and comment filtering are forms of negative query filters.

### *3.3.4 Threshold filter*

Threshold filtering involves setting thresholds for displaying the retrieved objects to the user. I define three kinds of threshold filters, "relationship thresholds," "ranking thresholds," and "number thresholds." Combinations of these thresholds are also possible.

#### *3.3.4.1 Relationship threshold*

With a relationship threshold the value used for determining the threshold is the $R_i$ relationship between the query and the

objects. In other words, any object $O_i$ with relationship $R_i$ that is less then threshold T gets eliminated from user domain $D_U$.

*3.3.4.2 Ranking threshold*

The threshold filtering can be based on the information display ranking. For example, the Google PageRank criteria can be used such that any object $O_i$ with rank $P_i$ that is less then threshold T gets eliminated from user domain $D_U$.

*3.3.4.3 Number threshold*

Number filtering is the process of simply reducing the number of files in the user domain $D_U$ to one that is more manageable. It requires that the information retrieval method be a best match method or that the information display process uses a ranking method (otherwise, all retrieved objects have equal relationships to the query and eliminating a specific number of them would have to be arbitrary). Given a number threshold N, if the retrieval method is best match, the objects $O_i$ are ordered from highest to lowest by their relationship $R_i$ until the number of objects displayed is N. If the retrieval method is exact match but the display process uses a ranking method, the objects $O_i$ are ordered from highest to lowest by their ranking $P_i$ until the number of objects displayed is N.

Note that thresholds need not be minimum thresholds. Maximum thresholds and combinations of minimum and maximum thresholds may be appropriate if the user wishes to study various aspects of the retrieved information such as statistical distributions of the information.

With regard to CodeMatch, the threshold filtering is, obviously, a form of a threshold filter.

### *3.3.5 Object relationship filter*

An object relationship filter allows the user to select an object $O_i$ that the user feels is characteristic of an object that belongs in the user information domain $D_U$ or does not belong in the user domain $D_U$. All similar objects are then removed, or all dissimilar objects are removed, depending on whether the filter is a "positive object relationship filter" or a "negative object relationship filter."

*3.3.5.1 Positive object relationship filter*

The user selects an object $O_i$ and specifies a minimum relationship value $R_M$. Object $O_i$ and all objects $O_k$ such that the relationship $R_{ik}$ between objects $O_i$ and $O_k$ is greater than or equal to the minimum relationship value $R_M$ are eliminated from the user information domain $D_U$. In this case, object $O_i$ is selected as an example of an object that the user feels is not relevant.

*3.3.5.2 Negative object relationship filter*

The user selects an object $O_i$ and specifies a minimum relationship value $R_M$. All objects $O_k$ such that the relationship $R_{ik}$ between objects $O_i$ and $O_k$ is less than the minimum relationship value $R_M$ are eliminated from the user information domain $D_U$. In this case, object $O_i$ is selected as an example of an object that the user feels is relevant.

Object relationship filtering allows user to select objects to be included or excluded from the user information domain without understanding the details of why the object is relevant or is not relevant.

## 4. APPLICATIONS

I have now defined various kinds of post-retrieval filtering. This kind of filtering has worked very well for CodeMatch, a program that finds correlation amongst software source code files. There are many other applications to which post-retrieval filtering can be applied that will offer many advantages to a user. Obviously Web searching can be greatly improved. Particularly for users who are not technically savvy, post-retrieval filtering can be used to narrow down search results that may have been produced from a query that was too broad and produced too many results.

Another related area where post-retrieval filtering can be advantageous is e-commerce. Users can find a Web page offering an item for sale. The Web page can be used as a query employing a best match method of retrieval to find similar items for sale at other locations on the Web. At that point, the user can employ post-retrieval filtering to reduce the number of results to a selection of items that the user can examine and decide to purchase in a reasonable amount of time. In particular, object relationship filtering can be employed, allowing the user to filter the results without needing to specify the exact criteria used for the filtering.

This paper refers to iterative filtering, because post-retrieval filtering should be an iterative process. It requires experimentation by the user. Some filters may turn out to eliminate too many results while other filters may not eliminate enough.

## 5. CONCLUSION

This paper has presented an example of post-retrieval filtering from work done on CodeMatch, a commercial tool for finding correlation between software source code files. Post-retrieval filtering for CodeMatch has improved the time to find plagiarized source code by an order of magnitude. The specific filtering employed in CodeMatch was presented and explained.

The concept of post-retrieval filtering was then expanded, generalized, and categorized as applied to all forms of information retrieval. Some examples of applications of post-retrieval filtering for Web searching and e-commerce were presented.

In conclusion, better methods of information retrieval will always be needed and these methods are improving regularly. Better methods of information display are also useful and there is a great demand for it as evidenced by the success of Google, one of whose major innovations was in the area of information display. Automatic filtering of retrieved information is a great goal and research is going on in that area also. However, automatic filtering may never be 100% accurate and manual filtering has many great benefits that have yet to be fully exploited.

## REFERENCES


[1] Belkin, N. J. & Croft, W. B.: "Information filtering and information retrieval: Two sides of the same coin?" *Communications of the ACM*, 35(12), 29-38, 1992.

[2] Brin, S. & Page, L.: "The Anatomy of a Large-Scale Hypertextual Web Search Engine," WWW7 / Computer Networks 30(1-7): 107-117, 1998.

[3] Burd, E. & Bailey, J.: "Evaluating Clone Detection Tools for Use during Preventative Maintenance," *Proc. 2nd IEEE International Workshop on Source Code Analysis and Manipulation (SCAM)* 2002, pp. 36-43, Oct. 2002.

[4] Chen, X., Francia, B., Li, M., Mckinnon, B, & Seker, A.: "Shared Information and Program Plagiarism Detection," *IEEE Transactions on Information Theory*, vol. 50, pp. 1545-1551, 2004.

[5] Ching Yiu, C. & Che Yin, I.: "Image Ranking Schemes Using Link-Structure Analysis Algorithm," *Proceedings International WWW Conference*(11), 2002.

[6] Clough, P.: "Plagiarism in natural and programming languages: an overview of current tools and technologies," Research Memoranda, CS-00-05, Department of Computer Science, University of Sheffield, UK, 2000.

[7] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R.: "Indexing by latent semantic analysis," *Journal of the American Society for Information Science*, 41(6):391-407, 1990.

[8] Faidhi, J. A. W. & Robinson, S. K.: "An empirical approach for detecting program similarity and plagiarism within a university programming environment," *Computer Education* Vol. 11, pp. 11-19, 1987.

[9] Foltz, P. W. & Dumai, S. T.: "Personalized Information Delivery: An Analysis of Information Filtering Methods," *Communications of the ACM*, 35(12), 51-60, 1992.

[10] Kleinberg, J.: "Authoritative sources in a hyperlinked environment" *Journal of the ACM*, 46, 1999

[11] Langville, A. N. & Meyer, C. D.: "Deeper inside PageRank," *Internet Mathematics*, 1(3), 335-400, 2005.

[12] Letsche, T. A. & Berry, M. W., "Large-Scale Information Retrieval with Latent Semantic Indexing", *Information Sciences*, Vol. 100, pp. 105-137, 1997.

[13] Mayrand, J., Leblanc, C., & Merlo, E.M.: "Experiment On The Automatic Detection Of Function Clones In A Software System Using Metrics," *International Conference on Software System Using Metrics*, pp. 244–253, 1996.

[14] McGreevy, M. W.: "A practical guide to interpretation of large collections of incident narratives using the QUORUM method," *NASA TM-112190*, Ames Research Center, Moffett Field, Calif., 1997.

[15] Parker, A. & Hamblen, J: *"Computer Algorithms for Plagiarism Detection," IEEE Transactions on Education* Vol. 32, No. 2., pp. 94-99, May 1989.

[16] Pirolli, P. & Card, S.K., "Information foraging," *Psychological Review*, 106: p. 643-675, 1999.

[17] Salton, G. & McGill, M. J.: *Introduction to Modern Information Retrieval*, McGraw Hill, New York, 1983.

[18] Zeidman, R.: "Software Source Code Correlation," *icis-comsar*, pp. 383-392, 5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse (ICIS-COMSAR'06), July 2006.

**Bob Zeidman** is a Senior Member of the IEEE, President of Zeidman Consulting (www.ZeidmanConsulting.com) a contract research and development firm, and President of Zeidman Technologies (www.zeidman.biz), that produces tools for embedded systems development. Among his publications are technical papers on hardware and software design methods as well as three textbooks -- *Designing with FPGAs and CPLDs*, *Verilog Designer's Library*, and *Introduction to Verilog*. He has taught courses at engineering conferences throughout the world. Bob holds two patents in software synthesis and another patent in hardware emulation. He earned a master's degree in electrical engineering at Stanford University and bachelor's degrees in physics and electrical engineering at Cornell University.