

# A SHORT HISTORY OF SOFTWARE QUALITY MEASUREMENT

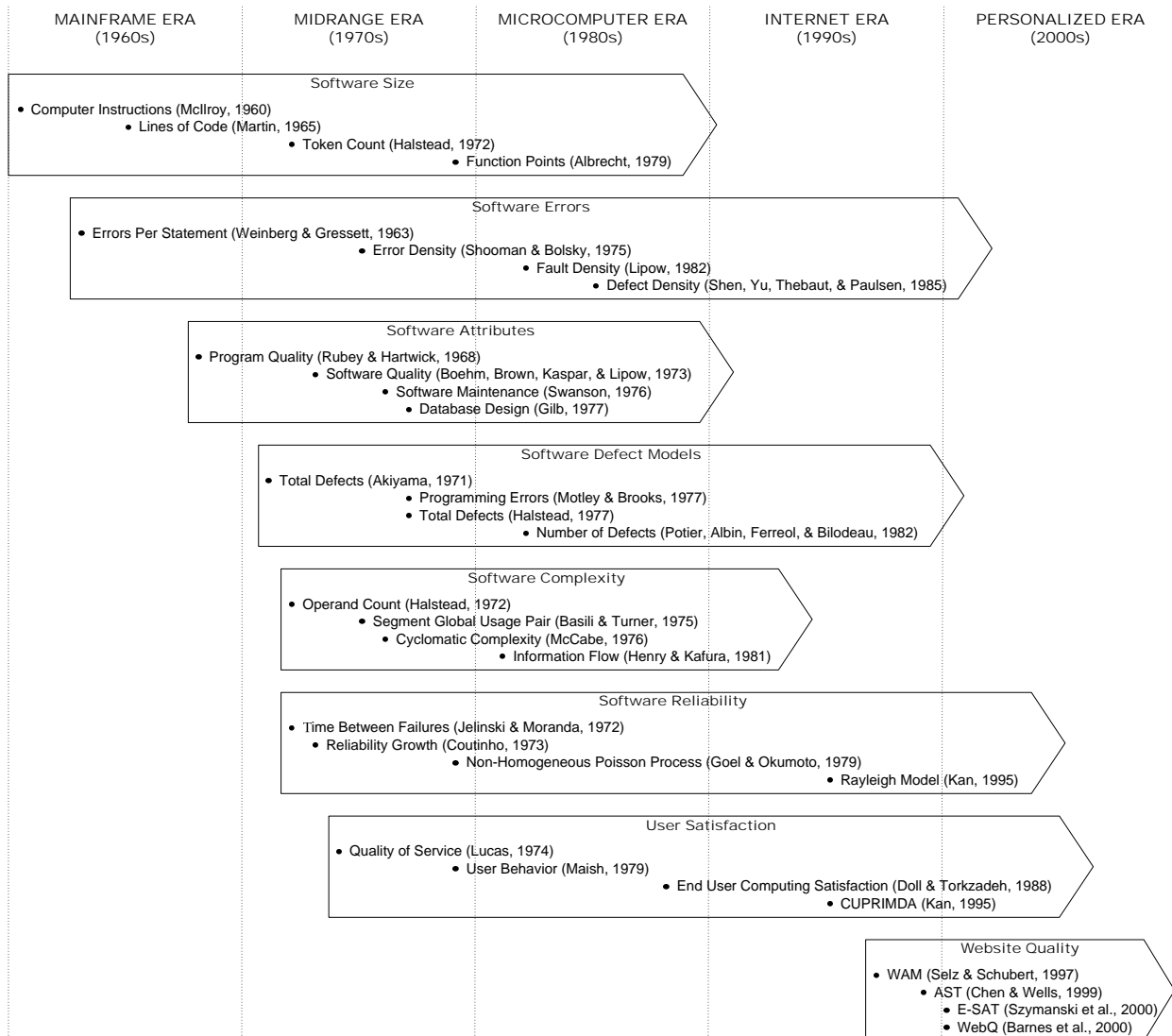


Figure 1. Timeline and history of software quality measures

Software size. One of the earliest known measures used to describe computer programs was software size (McIlroy, 1960). Software size is a measure of the volume, length, quantity, amount, and overall magnitude of a computer program (Conte, Dunsmore, & Shen, 1986). In the mid 1960s, lines of code or LOC was one of the first known measures of software size, which referred to the number of computer instructions or source statements comprising a computer program and usually expressed as thousands of lines of code (Martin, 1965). Almost a decade

later in the mid to late 1970s, more sophisticated measures of software size emerged such as token count, volume, function count, and function points (Conte, Dunsmore, & Shen, 1986). Recognizing that individual lines of code had variable lengths, token count was created to distinguish between unequally sized lines of code, which was technically defined as “basic syntactic units distinguishable by a compiler” (Halstead, 1977). In yet another attempt to accurately gauge the size of an individual line of code, volume was created to measure the actual size of a line of code in bits, otherwise known as binary zeros and ones (Halstead, 1977). Shortly thereafter, function count was created to measure software size in terms of the number of modules or subroutines (Basili & Reiter, 1979). Function points was another major measure of software size, which was based on estimating the number of inputs, outputs, master files, inquiries, and interfaces (Albrecht, 1979). Though software size is not a measure of software quality itself, it formed the basis of many measures or ratios of software quality right on through the modern era (e.g., number of defects, faults, or failures per line of code or function point). Furthermore, some treatises on software metrics consider software size one of the most basic measures of software complexity (Kan, 1995). Thus, a history of software quality measurement may not be complete without introducing an elementary discussion of software size.

Software errors. One of the earliest approaches for measuring software quality was the practice of counting software errors dating back to the 1950s when digital computers emerged. Software errors are human actions resulting in defects, defects sometimes manifest themselves as faults, and faults lead to failures, which are often referred to as software crashes (Kan, 1995). The concept of “errors per statement” first appeared in the early 1960s (Weinberg & Gressett, 1963) and studies of “error proneness” intensified by the close of the decade (Youngs, 1970). The term error density was coined in the mid 1970s, which referred to the simple ratio of errors

to software size (Shooman & Bolsky, 1975). Fault density was also a measure of software quality, which referred to the ratio of anomaly causing faults to software size (Lipow, 1982). The term defect density subsumed the measure of error and fault density in the mid 1980s, which referred to the ratio of software errors to software size (Shen, Yu, Thebaut, & Paulsen, 1985). Many unique types of errors were counted, such as number of requirement, design, coding, testing, and maintenance errors, along with number of changes and number of changed lines of code (Conte, Dunsmore, & Shen, 1986). Even the term problem density emerged in the early 1990s, which referred to the number of problems encountered by customers to measure and track software quality (Kan, 1995). The practice of counting errors, defects, faults, and failures as a means of measuring software quality enjoyed widespread popularity for more than five decades.

Software attributes. Another of the earliest approaches for measuring software quality was the practice of quantifying and assessing attributes or characteristics of computer programs. Software attributes are an “inherent, possibly accidental trait, quality, or property” such as functionality, performance, or usability (Institute of Electrical and Electronics Engineers, 1990). Logicon designed a model to measure software attributes such as correctness, logicity, non-interference, optimizability, intelligibility, modifiability and usability (Rubey & Hartwick, 1968). Next, TRW identified software attributes such as portability, reliability, efficiency, modifiability, testability, human engineering, and understandability (Boehm, Brown, Kaspar, & Lipow, 1973). These early works led to numerous specialized spin-offs such as framework for measuring the attributes of software maintenance (Swanson, 1976) and even a database’s design (Gilb, 1977). Spin-offs continued emerging with an increasing focus on operationalizing these attributes with real software metrics (Cavano & McCall, 1978; Dzida, Herda, & Itzfeldt, 1978; Gaffney, 1981). By the mid 1980s, this practice reached Japan (Sunazuka, Azuma, & Yamagishi, 1985) and a

comprehensive framework emerged replete with a detailed software measures (Arthur, 1985). Use of software attributes to measure software quality was exemplified by the functionality, usability, reliability, performance, and supportability or FURPS model (Grady & Caswell, 1987). Software attributes enjoyed widespread use among practitioners throughout the 1970s and 1980s because of their simplicity, though scientists passed them by in favor of statistical models.

Static defect models. One of the earliest approaches for predicting software quality was the use of statistical models referred to as static reliability or static software defect models. “A static model uses other attributes of the project or program modules to estimate the number of defects in software” (Kan, 1995), ignoring “rate of change” (Conte, Dunsmore, & Shen, 1986). One of the earliest software defect models predicted the number of defects in a computer program as a function of size, decision count, or number of subroutine calls (Akiyama, 1971). Multi-linear models were created with up to 10 inputs for the various types of statements found in software code such as comments, data, and executable instructions (Motley & Brooks, 1977). The theory of software science was extended to include defect models by using volume as an input, which itself was a function of program language and statement length (Halstead, 1977). Research into software defect models continued with more extensions based on software science, cyclomatic complexity, path, and reachability metrics (Potier, Albin, Ferreol, & Bilodeau, 1982). More defect models were created by mixing defects, problems, and software science measures such as vocabulary, length, volume, difficulty, and effort (Shen, Yu, Thebaut, & Paulsen, 1985). Later, IBM developed models for predicting problems, fielded defects, arrival rate of problems, and backlog projection, which were used to design midrange operating systems (Kan, 1995). Static linear or multi-linear statistical models to predict defects continue to be useful tools well into modern times, though older dynamic statistical reliability models are overtaking them.

Software complexity. Appearing in the early 1970s, the study of software complexity became one of the most common approaches for measuring the quality of computer programs. Software complexity is defined as “looking into the internal dynamics of the design and code of software from the program or module level to provide clues about program quality” (Kan, 1995). Software complexity sprang from fervor among research scientists eager to transform computer programming from an art into a mathematically based engineering discipline (Halstead, 1972). Many technological breakthroughs in the two decades prior to the mid 1970s led to the formation of software complexity measures. These included the advent of digital computers in the 1950s, discovery of high level computer programming languages, and the formation of compiler theory. Furthermore, flowcharting was routinely automated, axiomatic theorems were used for designing new computer languages, and analysis of numerical computer algorithms became commonplace. As a result, three major classes of software complexity metrics arose for measuring the quality of software: (a) data structure, (b) logic structure, and (c) composite metrics (Weismann, 1973). One of the first data structure metrics was the count of operands, which measured the number of variables, constants, and labels in a computer program versus measuring logic (Halstead, 1972). The segment-global-usage-pair metric determined complexity by counting references to global variables; a high number of which was considered bad among coders (Basili & Turner, 1975). Another data structure metric was the span between variables, which measured how many logic structure statements existed between variables where a higher number was poor (Elshoff, 1976). A unique data structure metric for measuring software quality was the number of live variables within a procedure or subroutine as a sign of undue complexity (Dunsmore & Gannon, 1979). One data structure metric surviving to modern times is the information flow, or fan in - fan out metric, which measures the number of modules that exchange data (Henry & Kafura, 1981).

Logic structure metrics were cyclomatic complexity or paths (McCabe, 1976), minimum paths (Schneidewind & Hoffmann, 1979) and gotos or knots (Woodward, Hennell, & Hedley, 1979). Also included were nesting (Dunsmore & Gannon, 1980), reachability (Shoorman, 1983), nest depth (Zolnowsky & Simmons, 1981), and decisions (Shen, Yu, Thebaut, & Paulsen, 1985). Composite metrics combined cyclomatic complexity with other attributes of computer programs to achieve an accurate estimate software quality (Myers, 1977; Hansen, 1978; Oviedo, 1980). They also included system complexity (Card & Glass, 1990) and syntactic construct (Lo, 1992). Finally, it's important to note that most complexity metrics are now defunct, though cyclomatic complexity, which arose out of this era, is still used as a measure of software quality today.

Software reliability. Also emerging in the early 1970s, software reliability was created to predict the number of defects or faults in software as a method of measuring software quality. Software reliability is the “probability that the software will execute for a particular period of time without failure, weighted by the cost to the user of each failure encountered” (Myers, 1976). Major types of reliability models include: (a) finite versus infinite failure models (Musa, 1999), (b) static versus dynamic (Conte, Dunsmore, & Shen, 1986), and (c) deterministic versus probabilistic (Pham, 2000). Major types of dynamic reliability models include: life cycle versus reliability growth (Kan, 1995) and failure rate, curve fitting, reliability growth, non-homogeneous Poisson process, and Markov structure (Pham, 2000). One of the first and most basic failure rate models estimated the mean time between failures (Jelinski & Moranda, 1972). A slightly more sophisticated failure rate model was created based on the notion that software became more reliable with each successive code failure repaired (Schick & Wolverton, 1978). The next failure rate model assumed the failure rate was initially constant and then begins to decrease (Moranda, 1979). Multiple failure rate models appeared throughout the 1970s to round

out this family of reliability models (Goel & Okumoto, 1979; Littlewood, 1979; Sukert, 1979). Reliability or “exponential” growth models followed the appearance of failure rate models, which measured the reliability of computer programs during testing as a function of time or the number of tests (Coutinho, 1973; Wall & Ferguson, 1977). Another major family of reliability models is the non-homogeneous Poisson process models, which estimate the mean number of cumulative failures up to a certain point in time (Huang, 1984; Goel & Okumoto, 1979; Musa, Iannino, & Okumoto, 1987; Ohba, 1984; Yamada, Ohba, & Osaki, 1983). Reliability models estimate the number of software failures after development based on failures encountered during testing and operation. Though rarely mentioned, the Rayleigh life cycle reliability model accurately estimates defects inserted and removed throughout the software lifecycle (Kan, 1995). Some researchers believe using software reliability models offers the best hope for transforming computer programming from a craft industry into a true engineering discipline.

User satisfaction. User satisfaction gradually became a measure of software quality during the 1950s, 1960s, and 1970s (Thayer, 1958; Hardin, 1960; Kaufman, 1966; Lucas, 1973). User satisfaction is defined as “the sum of one’s feelings or attitudes toward a variety of factors affecting that situation,” e.g., computer use and adoption by end users (Bailey & Pearson, 1983). Though not the first, one study of user satisfaction analyzed attitudes toward quality of service, management support, user participation, communication, and computer potential (Lucas, 1974). A more complex study of user satisfaction looked at feelings about staff, management support, preparation for its use, access to system, usefulness, ease-of-use, and flexibility (Maish, 1979). Most studies up until 1980 focused on the end user’s satisfaction toward software developers; but one study squarely focused on the end user’s satisfaction with the software itself (Lyons, 1980). One of the first studies to address a variety of software attributes such as software accuracy,

timeliness, precision, reliability, currency, and flexibility appeared (Pearson & Bailey, 1980). Studies throughout the 1980s addressed user satisfaction with both designers and software (Walsh, 1982; Bailey & Pearson, 1983; Ives, Olson, & Baroudi, 1983; Joshi, Perkins, & Bostrom, 1986; Baroudi & Orlikowski, 1988). The late 1980s marked a turning point with studies focusing entirely on user satisfaction with the software itself and attributes such as content, accuracy, format, ease of use, and timeliness of the software (Doll & Torkzadeh, 1988). A study of user satisfaction at IBM was based on reliability, capability, usability, installability, maintainability, performance, and documentation factors (Kekre, Krishnan, & Srinivasan, 1995). Throughout the 1990s, IBM used a family of user satisfaction models called UPRIMD, UPRIMDA, CUPRIMDA, and CUPRIMDSO, which referred variously to factors of capability, usability, performance, reliability, installability, maintainability, documentation, availability, service, and overall satisfaction (Kan, 1995). User satisfaction, now commonly referred to as customer satisfaction, is no doubt related to earlier measures of software attributes, usability or user friendliness of software, and more recently web quality.

Website quality. Appearing in the late 1990s, following the user satisfaction movement, models of website quality appeared as important measures of software quality (Lindroos, 1997). One of the first models of website quality identified background, image size, sound file display, and celebrity endorsement as important factors of software quality (Dreze & Zufryden, 1997). The web assessment method or WAM quickly followed with quality factors of external bundling, generic services, customer specific services, and emotional experience (Selz & Schubert, 1997). In what promised to be the most prominent web quality model, attitude toward the site or AST had quality factors of entertainment, informativeness, and entertainment (Chen & Wells, 1999). The next major model was the e-satisfaction model with its five factors of convenience, product

offerings, product information, website design, and financial security (Szymanski & Hise, 2000). The website quality model or WebQual for business school portals was based on factors of ease-of-use, experience, information, and communication and integration (Barnes & Vidgen, 2000). An adaptation of the service quality or ServQual model, WebQual 2.0 measured quality factors such as tangibles, reliability, responsiveness, assurance, and empathy (Barnes & Vidgen, 2001). The electronic commerce user consumer satisfaction index or ECUSI consisted of 10 factors such as product information, consumer service, purchase result and delivery, site design, purchasing process, product merchandising, delivery time and charge, payment methods, ease-of-use, and additional information services (Cho & Park, 2001). Based on nine factors, the website quality or SiteQual model consisted of aesthetic design, competitive value, ease-of-use, clarity of ordering, corporate and brand equity, security, processing speed, product uniqueness, and product quality assurance (Yoo & Donthu, 2001). In what promised to be exclusively for websites, the Internet retail service quality or IRSQ model was based on nine factors of performance, access, security, sensation, information, satisfaction, word of mouth, likelihood of future purchases, and likelihood of complaining (Janda, Trocchia, & Gwinner, 2002). In a rather complex approach, the expectation-disconfirmation effects on web-customer satisfaction or EDEWS model consists of three broad factors (e.g., information quality, system quality, and web satisfaction) and nine subfactors (McKinney, Yoon, & Zahedi, 2002). In one of the smallest and most reliable website quality models to-date, the electronic commerce retail quality or EtailQ model, consists of only four major factors (e.g., fulfillment and reliability, website design, privacy and security, and customer service) and only 14 instrument items (Wolfenbarger & Gilly, 2003). Based on techniques for measuring software quality dating back to the late 1960s, more data has been collected and validated using models of website quality than any other measure.

## REFERENCES

- Akiyama, F. (1971). An example of software system debugging. *Proceedings of the International Federation for Information Processing Congress, Ljubljana, Yugoslavia*, 353-379.
- Albrecht, A. J. (1979). Measuring application development productivity. *Proceedings of the IBM Applications Development Joint SHARE/GUIDE Symposium, Monterrey, California, USA*, 83-92.
- Arthur, L. J. (1985). *Measuring programmer productivity and software quality*. New York, NY: John Wiley & Sons.
- Bailey, J. E., & Pearson, S. W. (1983). Development of a tool for measuring and analyzing computer user satisfaction. *Management Science*, 29(5), 530-545.
- Barnes, S. J., & Vidgen, R. T. (2000). Webqual: An exploration of web site quality. *Proceedings of the Eighth European Conference on Information Systems, Vienna, Austria*, 298-305.
- Barnes, S. J., & Vidgen, R. T. (2001). An evaluation of cyber bookshops: The webqual method. *International Journal of Electronic Commerce*, 6(1), 11-30.
- Baroudi, J. J., & Orlikowski, W. J. (1988). A short form measure of user information satisfaction: A psychometric evaluation and notes on use. *Journal of Management Information Systems*, 4(4), 44-59.
- Basili, V. R., & Reiter, R. W. (1979). An investigation of human factors in software development. *IEEE Computer*, 12(12), 21-38.
- Basili, V. R., & Turner, J. (1975). Iterative enhancement: A practical technique for software development. *IEEE Transactions on Software Engineering*, 1(4), 390-396.
- Boehm, B. W., Brown, J. R., Kaspar, H., & Lipow, M. (1973). *Characteristics of software quality* (TRW-SS-73-09). Redondo Beach, CA: TRW Corporation.
- Card, D. N., & Glass, R. L. (1990). *Measuring software design quality*. Englewood Cliffs, NJ: Prentice Hall.
- Cavano, J. P., & McCall, J. A. (1978). A framework for the measurement of software quality. *Proceedings of the Software Quality Assurance Workshop on Functional and Performance Issues, San Diego, California, USA*, 133-139.
- Chen, Q., & Wells, W. D. (1999). Attitude toward the site. *Journal of Advertising Research*, 39(5), 27-37.
- Cho, N., & Park, S. (2001). Development of electronic commerce user consumer satisfaction index (ECUSI) for internet shopping. *Industrial Management and Data Systems*, 101(8/9), 400-405.

- Conte, S. D., Dunsmore, H. E., & Shen, V. Y. (1986). *Software engineering metrics and models*. Menlo Park, CA: Benjamin Cummings.
- Coutinho, J. S. (1973). Software reliability growth. *Proceedings of the IEEE Symposium on Computer Software Reliability, New York, New York, USA*, 58-64.
- Doll, W. J., & Torkzadeh, G. (1988). The measurement of end user computing satisfaction. *MIS Quarterly*, 12(2), 258-274.
- Dreze, X., & Zufryden, F. (1997). Testing web site design and promotional content. *Journal of Advertising Research*, 37(2), 77-91.
- Dunsmore, H. E., & Gannon, J. D. (1979). Data referencing: An empirical investigation. *IEEE Computer*, 12(12), 50-59.
- Dunsmore, H. E., & Gannon, J. D. (1980). Analysis of the effects of programming factors on programming effort. *Journal of Systems and Software*, 1(2), 141-153.
- Dzida, W., Herda, S., & Itzfeldt, W. D. (1978). User perceived quality of interactive systems. *Proceedings of the Third International Conference on Software Engineering, Atlanta, Georgia, USA*, 188-195.
- Elshoff, J. L. (1976). An analysis of some commercial PL/1 programs. *IEEE Transactions on Software Engineering*, 2(2), 113-120.
- Gaffney, J. E. (1981). Metrics in software quality assurance. *Proceedings of the ACM SIGMETRICS Workshop/Symposium on Measurement and Evaluation of Software Quality, Las Vegas, Nevada, USA*, 126-130.
- Gilb, T. (1977). *Software metrics*. Cambridge, MA: Winthrop Publishers.
- Goel, A. L., & Okumoto, K. (1979). Time dependent error detection rate model for software and other performance measures. *IEEE Transactions on Reliability*, 28(3), 206-211.
- Grady, R. B., & Caswell, R. B. (1987). *Software metrics: Establishing a company wide program*. Englewood Cliffs, NJ: Prentice Hall.
- Halstead, M. H. (1972). Natural laws controlling algorithm structure? *ACM SIGPLAN Notices*, 7(2), 19-26.
- Halstead, M. H. (1977). *Elements of software science*. New York, NY: Elsevier North Holland.
- Hansen, W. J. (1978). Measurement of program complexity by the pair (cyclomatic number, operator count). *ACM SIGPLAN Notices*, 13(3), 29-33.
- Hardin, K. (1960). Computer automation, work environment, and employee satisfaction: A case study. *Industrial and Labor Relations Review*, 13(4), 559-567.

- Henry, S., & Kafura, D. (1981), Software structure metrics based on information flow. *IEEE Transactions on Software Engineering*, 7(5), 510-518.
- Huang, X. Z. (1984). The hypergeometric distribution model for predicting the reliability of software. *Microelectronics and Reliability*, 24(1), 11-20.
- Institute of Electrical and Electronics Engineers. (1990). *IEEE standard glossary of software engineering terminology* (IEEE Std 610.12-1990). New York, NY: Author.
- Ives, B., Olson, M. H., & Baroudi, J. J. (1983). The measurement of user information satisfaction. *Communications of the ACM*, 26(10), 785-793.
- Janda, S., Trocchia, P. J., & Gwinner, K. P. (2002). Consumer perceptions of internet retail service quality. *International Journal of Service Industry Management*, 13(5), 412-433.
- Jelinski, Z., & Moranda, P. B. (1972). Software reliability research. In W. Freiberger (Ed.), *Statistical computer performance evaluation* (pp. 465-484). New York, NY: Academic Press.
- Joshi, K., Perkins, W. C., & Bostrom, R. P. (1986). Some new factors influencing user information satisfaction: Implications for systems professionals. *Proceedings of the 22nd Annual Computer Personnel Research Conference, Calgary, Canada*, 27-42.
- Kan, S. H. (1995). *Metrics and models in software quality engineering*. Reading, MA: Addison-Wesley.
- Kaufman, S. (1966). The IBM information retrieval center (ITIRC): System techniques and applications. *Proceedings of the 21st National Conference for the Association for Computing Machinery, New York, New York, USA*, 505-512.
- Kekre, S., Krishnan, M. S., & Srinivasan, K. (1995). Drivers of customer satisfaction in software products: Implications for design and service support. *Management Science*, 41(9), 1456-1470.
- Lindroos, K. (1997). Use quality and the world wide web. *Information and Software Technology*, 39(12), 827-836.
- Lipow, M. (1982). Number of faults per line of code. *IEEE Transactions on Software Engineering*, 8(4), 437-439.
- Littlewood, B. (1979). Software reliability model for modular program structure. *IEEE Transactions on Reliability*, 28(3), 241-246.
- Lo, B. (1992). *Syntactical construct based APAR projection* (Technical Report). San Jose, CA: IBM Santa Teresa Research Laboratory.
- Lucas, H. C. (1973). User reactions and the management of information services. *Management Informatics*, 2(4), 165-162.

- Lucas, H. C. (1974). Measuring employee reactions to computer operations. *Sloan Management Review*, 15(3), 59-67.
- Lyons, M. L. (1980). Measuring user satisfaction: The semantic differential technique. *Proceedings of the 17th Annual Conference on Computer Personnel Research, Miami, Florida, USA*, 79-87.
- Maish, A. M. (1979). A user's behavior toward his MIS. *MIS Quarterly*, 3(1), 39-52.
- Martin, J. (1965). *Programming real-time computer systems*. Englewood Cliffs, NJ: Prentice Hall.
- McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, 2(4), 308-320.
- McIlroy, M. D. (1960). Macro instruction extensions of compiler languages. *Communications of the ACM*, 3(4), 214-220.
- McKinney, V., Yoon, K., & Zahedi, F. (2002). The measurement of web customer satisfaction: An expectation and disconfirmation approach. *Information Systems Research*, 13(3), 296-315.
- Moranda, P. B. (1979). An error detection model for application during software development. *IEEE Transactions on Reliability*, 28(5), 325-329.
- Motley, R. W., & Brooks, W. D. (1977). *Statistical prediction of programming errors (RADCTR-77-175)*. Griffis AFB, NY: Rome Air Development Center.
- Musa, J. D. (1999). *Software reliability engineering*. New York, NY: McGraw Hill.
- Musa, J. D., Iannino, A., & Okumoto, K. (1987). *Software reliability: Measurement, prediction, and application*. New York, NY: McGraw Hill.
- Myers, G. J. (1976). *Software reliability: Principles and practices*. New York, NY: John Wiley & Sons.
- Myers, G. J. (1977). An extension to the cyclomatic measure of program complexity. *SIGPLAN Notices*, 12(10), 61-64.
- Ohba, M. (1984). Software reliability analysis models. *IBM Journal of Research and Development*, 21(4), 428-443.
- Oviedo, E. I. (1980). Control flow, data flow, and program complexity. *Proceedings of the Fourth International IEEE Computer Software and Applications Conference (COMPSAC 1980), Chicago, Illinois, USA*, 146-152.
- Pearson, S. W., & Bailey, J. E. (1980). Measurement of computer user satisfaction. *ACM SIGMETRICS Performance Evaluation Review*, 9(1), 9-68.

- Pham, H. (2000). *Software reliability*. Singapore: Springer Verlag.
- Potier, D., Albin, J. L., Ferreol, R., & Bilodeau, A. (1982). Experiments with computer software complexity and reliability. *Proceedings of the Sixth International Conference on Software Engineering, Tokyo, Japan*, 94-103.
- Rubey, R. J., & Hartwick, R. D. (1968). Quantitative measurement of program quality. *Proceedings of the 23rd ACM National Conference, Washington, DC, USA*, 671-677.
- Schick, G. J., & Wolverton, R. W. (1978). An analysis of competing software reliability analysis models. *IEEE Transactions on Software Engineering*, 4(2), 104-120.
- Schneidewind, N. F., & Hoffmann, H. (1979). An experiment in software error data collection and analysis. *IEEE Transactions on Software Engineering*, 5(3), 276-286.
- Selz, D., & Schubert, P. (1997). Web assessment: A model for the evaluation and the assessment of successful electronic commerce applications. *Electronic Markets*, 7(3), 46-48.
- Shen, V. Y., Yu, T. J., Thebaut, S. M., & Paulsen, L. R. (1985). Identifying error prone software: An empirical study. *IEEE Transactions on Software Engineering*, 11(4), 317-324.
- Shooman, M. L. (1983). *Software engineering*. New York, NY: McGraw Hill.
- Shooman, M. L., & Bolsky, M. I. (1975). Types, distribution, and test and correction times for programming errors. *Proceedings of the International Conference on Reliable Software, Los Angeles, California, USA*, 347-357.
- Sukert, A. N. (1979). Empirical validation of three software error prediction models. *IEEE Transactions on Reliability*, 28(3), 199-205.
- Sunazuka, T., Azuma, M., & Yamagishi, N. (1985). Software quality assessment technology. *Proceedings of the Eighth International Conference on Software Engineering, London, England*, 142-148.
- Swanson, E. B. (1976). The dimensions of maintenance. *Proceedings of the Second International Conference on Software Engineering, San Francisco, California, USA*, 492-497.
- Szymanski, D. M., & Hise, R. T. (2000). E-satisfaction: An initial examination. *Journal of Retailing*, 76(3), 309-322.
- Thayer, C. H. (1958). Automation and the problems of management. *Vital Speeches of the Day*, 25(4), 121-125.
- Wall, J. K., & Ferguson, P. A. (1977). Pragmatic software reliability prediction. *Proceedings of the Annual Reliability and Maintainability Symposium, Piscataway, New Jersey, USA*, 485-488.

- Walsh, M. D. (1982). Evaluating user satisfaction. *Proceedings of the 10th Annual ACM SIGUCCS Conference on User Services, Chicago, Illinois, USA*, 87-95.
- Weinberg, G. M., & Gressett, G. L. (1963). An experiment in automatic verification of programs. *Communications of the ACM*, 6(10), 610-613.
- Weissman, L. (1973). Psychological complexity of computer programs. *ACM SIGPLAN Notices*, 8(6), 92-95.
- Wolfenbarger, M., & Gilly, M. C. (2003). Etailq: Dimensionalizing, measuring, and predicting etail quality. *Journal of Retailing*, 79(3), 183-198.
- Woodward, M. R., Hennell, M. A., & Hedley, D. (1979). A measure of control flow complexity in program text. *IEEE Transactions on Software Engineering*, 5(1), 45-50.
- Yamada, S., Ohba, M., & Osaki, S. (1983). S shaped reliability growth modeling for software error prediction. *IEEE Transactions on Reliability*, 32(5), 475-478.
- Yoo, B., & Donthu, N. (2001). Developing a scale to measure the perceived quality of an internet shopping site (sitequal). *Quarterly Journal of Electronic Commerce*, 2(1), 31-45.
- Youngs, E. A. (1970). *Error proneness in programming*. Unpublished doctoral dissertation. University of North Carolina at Chapel Hill, Chapel Hill, NC, United States.
- Zolnowski, J. C., & Simmons, D. B. (1981). Taking the measure of program complexity. *Proceedings of the AFIPS National Computer Conference, Chicago, Illinois, USA*, 329-336.