CS504 - Software Engineering - I FAQs By www.virtualians.pk

Question: Professional Journals related to Software Engineering?

Answer:

Software Engineering: The Production of Quality Software by Shari Pfleeger, 2nd Edition, Macmillan, 1991, ISBN 0-02-395115-X. hsrender@happy.colorado.edu: Like #2&#3, had the best explanations of what I want to cover (different engineering lifecycles, methods, and tools).

Software Engineering: A Practitioner's Approach by Roger Pressman, 4th Edition, McGraw-Hill, 1996, ISBN 0070521824

hsrender@happy.colorado.edu: (on 2nd edition): Like #1&#3, had the best explanations of what I want to cover (different engineering lifecycles, methods, and tools).

robb@iotek.uucp (Robb Swanson): The definitive book on the subject as far as I'm concerned.

johnson@aplcen.apl.jhu.edu (Michelle Johnson): A good text book as well as reference.

Software Systems Engineering by Andrew Sage and James D. Palmer.

hsrender@happy.colorado.edu: Like #1&#2, had the best explanations of what I want to cover (different engineering lifecycles, methods, and tools).

Fundamentals of Software Engineering by Ghezzi, Jayazeri and Mandrioli, Prentice-Hall, 1991

hsrender@happy.colorado.edu: Like #5, good, and covered the issue of specifications and verification better, but at the expense of other aspects of the development process. I may use one of them for a graduate course in software engineering.

nancy@murphy.ICS.UCI.EDU (Nancy Leveson): Better than Sommerville, although I like much of Sommerville.

Software Engineering with Abstractions by Valdis Berzins and Luqi, Addison Wesley, 1991, 624 pages.

hsrender@happy.colorado.edu: Like #4, good, and covered the issue of specifications and verification better, but at the expense of other aspects of the development process. I may use one of them for a graduate course in software engineering.

straub@cs.UMD.EDU (Pablo A. Straub): Both this and #9 have a good emphasis on using formal techniques (i.e., doing engineering properly), but they do not disregard informal methods; chapters are roughly organized around the traditional lifecycle. #5 is longer and can be used in a two-term sequence or for graduate students (it's possible to use it in a one-term undergrad course by covering only part of the material). One thing I like is that management and validation is given in all chapters, so that these activities are integrated into the development process. Emphasizes the use of formally specified abstractions. Uses the authors' specification language (Spec) to develop a project in Ada.

Software Engineering by Ian Sommerville, Addison-Wesley, ISBN 0-201-17568-1

hsrender@happy.colorado.edu: Our current text, and my basic problem with it is the vague way it covers many of the topics.

Software Engineering with Student Project Guidance by Barbara Mynatt

hsrender@happy.colorado.edu: Like #8, not bad, but fairly low-level and doesn't cover many tools and techniques I consider valuable.

Software Engineering by Roger Jones

hsrender@happy.colorado.edu: Like #7, not bad, but fairly low-level and doesn't cover many tools and techniques I consider valuable.

Software Engineering: Planning for Change by David Alex Lamb, Prentice-Hall, 1988, 298 pages.

straub@cs.UMD.EDU (Pablo A. Straub): Both this and #5 have a good emphasis on using formal techniques (i.e., doing engineering properly), but they do not disregard informal methods; chapters are roughly organized around the traditional lifecycle. #9 has the advantage of being shorter, yet covering most relevant topics (lifecycle phases, formal specs, v&v, configurations, management, etc.). It is very appropriate for an undergrad course. It emphasizes that maintenance is a given and should be taken into account (hence the title). Several specification techniques are covered and used to develop a project in Pascal.

A Practical Handbook for Software Development by N.D. Birrell and M.A. Ould, Cambridge University Press, 1985/88. ISBN 0-521-34792-0 (Paper cover); ISBN 0-521-25462-0 (Hard cover).

ewoods@hemel.bull.co.uk (Eoin Woods):

Fundamentals of Computing for Software Engineers by Eric S. Chan & Murat M. Tanik, Van Nostrand Reinhold.

kayaalp@csvax.seas.smu.edu (Mehmet M. Kayaalp MD):

Classic and Object-Oriented Software Engineering, 3rd Edition, by Stephen R. Schach, Richard D. Irwin, Inc. (ISBN 0-256-18298-1), 1996. Advertised as senior/first year graduate level, emphasizing the object-oriented paradigm, metrics, CASE tools, testing, and maintenance.

Practical Software Engineering by Stephen R. Schach, Aksen Associates and Richard D. Irwin Inc. (ISBN 0-256-11455-2), 1992. Advertised as sophomore through senior level, emphasizing teams, maintenance, reuse, CASE tools.

Question: What is Software Engineering?

Answer:

IEEE Standard Computer Dictionary, 610, ISBN 1-55937-079-3, 1990: The application of a systematic, disciplined, quantifiable approach to development, operation, and maintenance of software; that is, the application of engineering to software.

Textbooks tend not to give definitions, but instead spend their introductory chapters explaining characteristics of the discipline. Interesting phrases include:

Ian Sommerville, Software Engineering, 5th edition, Addison-Wesley, 1996. The specification, development, management, and evolution of software systems. Not constrained by materials governed by physical laws or manufacturing processes. Theories, methods, and tools needed to develop software. Evolving models of the real world.

Stephen R. Schach, Software Engineering, 2nd Edition, Richard D.Irwin, Inc. and Aksen Associates, 1993. A discipline whose aim is the production of quality software, delivered on time, within budget, and satisfying users' needs.

Shari Lawrence Pfleeger, Software Engineering: the Production of Quality Software, 2nd Edition, Macmillan, 1991, ISBN 0-02-395115-X. Designing and developing high-quality software. Application of computer science techniques to a variety of problems. We are problem-solvers rather that theoreticians.

Question: What's a CASE Tool?

Answer: CASE stands for Computer Aided Software Engineering; it can be used to mean any computer-based tool for software planning, development, and evolution. Various people regularly call the following 'CASE': Structured Analysis (SA), Structured Design (SD), Editors, Compilers, Debuggers, Edit-Compile-Debug environments, Code Generators, Documentation Generators, Configuration Management, Release Management, Project Management, Scheduling, Tracking, Requirements Tracing, Change Management (CM), Defect Tracking, Structured Discourse, Documentation editing, Collaboration tools, Access Control, Integrated Project Support Environments (IPSEs), Intertool message systems, Reverse Engineering, Metric Analyzers.

Question: What's a 'function point'?

Answer:

Function points and feature points are methods of estimating the "amount of functionality" required for a program, and are thus used to estimate project completion time. The basic idea involves counting inputs, outputs, and other features of a description of functionality. Bruno Peeters has collected a biblography on function points.

If interested, for a fee you can join:

International Function Point Users Group

5008-28 Pine Creek Drive

Blendonview Office Park

Westerville, Ohio 43081-4899

614-895-7130

Home page

Question: What's the 'spiral model'?

Answer: (1) Barry Boehm, "A Spiral Model of Software Development and Enhancement", ACM SIGSOFT Software Engineering Notes, August 1986. (2) Barry Boehm "A Spiral Model of Software Development and Enhancement" IEEE Computer, vol.21, #5, May 1988, pp 61-72. Basically, the idea is evolutionary development, using the waterfall model for each step; it's intended to help manage risks. Don't define in detail the entire system at first. The developers should only define the highest priority features. Define and implement those, then get feedback from users/customers (such feedback distinguishes "evolutionary" from "incremental" development). With this knowledge, they should then go back to define and implement more features in smaller chunks.

Question: Are lines-of-code (LOC) a useful productivity measure?

Answer: Not unless you are very careful. Capers Jones' book has a detailed and insightful discussion of Lines of Code, including anomalies, and shows how to use it sensibly (eg in a single job shop, with a single language, and a standard company coding style). It is easy to cook up anomalies where LOC gives different numbers for code written in different styles, but pathological cases should get caught in code inspections. References:

T. Capers Jones, Programming Productivity, McGraw-Hill, New York, 1986

Capers Jones, Applied Software Measurement: Assuring Productivity and Quality, McGraw-Hill, Inc., 1991, 494 pages ISBN 0-07-032813-7

The appendices of the latter give rules for counting procedural source code, as well as rules for counting function points and feature points. The following study, cited in Boehm's Software Engineering Economics, claims that anomalies that seriously "fool" the LOC metric show up rarely in real code.

R. Nelson Software Date Collection and Analysis at RADC, Rome Air Development Center, Rome, NY. 1978.

Question: Should software professionals be licenced/certified?

Answer: This is a very controversial and political question. Generally, certification is something voluntary, while licencing is regulated by governments. Certification generally means some agency warrants you meet its standards; licencing generally means that to claim to practice a certain profession requires a government licence, often administered through a professional organization. In theory both are supposed to help judge if someone is capable of doing certain jobs.

Licencing isn't currently required for computing professionals; some people would like to see some jobs require it, as with established branches of engineering. Others don't like government intervention, and/or believe many people who wouldn't get licenced are perfectly competent.

Computing professionals in the USA have had a certification program for years, administered by the Institute for Certification of Computer Professionals (708-299-4227), a meta-organization with representatives from ACM, IEEE-CS, ADAPSO, ICCA, IACE, AIM, DPMA, AISP, COMMON, ASM, CIPS, and AWC. There are three certificates aimed at different broad types of practitioner, and many areas of specialization. To keep a certificate requires at least 40 hours of continuing education each year; credit can also be obtained for self-study, teaching, publication, etc.

Question: What is the SEI maturity model?

Answer: Maturity is not an easy concept to get down to a single paragraph, but consider this. Premise: The quality of a software system is largely governed by the quality of the process used to develop and maintain the software. Basics: The first step in improving the existing situation is to get management buy-in and management action to clean up the software management processes (walk the talk, as TQMers frequently say). Integration: The second step is to get everyone working together as a team. Measurement: The third step is to establish objective ways of understanding status and predict where things are going in your process. Continuous improvement: Understand that this is building a foundation for continually getting better.

Question: Where can I get copies of standards??

Answer:

ISO, ANSI, and IEEE standards are usually sold to raise some of the funds that the various national and international standards bodies (who usually own the copyright) need to keep afloat; thus they are not normally available electronically. Also, the organizations are concerned that electronic copies would make it too easy for people to disseminate doctored versions of the standards.

Some IEEE standards are available by annual electronic subscription; see http://standards.ieee.org/catalog/olis/

For ITU (formerly CCITT) standards, see the ITU gopher server,

Question: What is 'cleanroom'?

Answer: 'Cleanroom' is a software process based on mathematical verification of components and statistical system-level testing. Cleanroom Software Engineering, Inc. keeps a more extensive definition, including a bibliography.

Question: What is requirement elicitation?

Answer: Requirements elicitation is concerned with where software requirements come from and how the software engineer can collect them. It is the first stage in building an understanding of the problem the software is required to solve. It is fundamentally a human activity, and is where the stakeholders are identified and relationships established between the development team and the customer. It is variously termed “requirements capture,” “requirements discovery,” and “requirements acquisition