Resources for The Development System for CSE403

People, Processes and Tools

Other relevant materials written by jb:

Improving the Development System Model (Adapted from an article in IEEE Computer) – Guesting in James Bach's *Computer Realities* column, this discusses how skilled project managers tune the development system to the problem at hand, even when they don't admit it.

The Development System for Emerging Technologies and Emerging Markets – Tools Eastern Europe, 2002 – Invited to keynote. This addresses changes in the development system, supporting IT systems that directly implement business value chains, and extend those chains outside the organization to suppliers, partners and customers.

A Reflection on a Workshop called PSL – This is a discussion of experiential learning. This essay illustrates some system effects, and suggests that developing systems is a high-performance activity. Find it here: www.geraldmweinberg.com/READING.html

Readings on Systems Thinking, and creating a "Diagram of Effects"

Peter Senge & company – *The Fifth Discipline*. Read about the beer game. Then read up on controlling dynamic systems in any engineering discipline (mechanical, electrical, chemical, civil). Then read about the beer game again.

Christopher Alexander – This architect was referenced by software designers before the patterns people arrived. Start with: *A Pattern Language: Towns, Buildings, Construction*.

Why Things Bite Back: Technology and the Revenge of Unintended Consequences – Edward Tenner wrote this study of system failures in technology – all systems effects.

Normal Accidents by Charles Perrow. Sometimes frustrating because he doesn't have all the tools to describe precisely what he's talking about. But he's talking about system effects, and how they emerge. His description of what makes systems go awry (or not): invisibility, coupling, rapid response, and so on is a good checklist for your development processes. If you've got invisibility, coupling, etc. you project will likely be unpleasant.

Synergetics – R. Buckminster Fuller. His thinking was dominantly "systems driven." Synergetics is his text on this thinking style, from his point of view. Fuller is often hard to read. I've found that decomposing his prose as prose-poems helps.

Measuring and Managing Performance in Organizations – Robert Austin. He never uses a systems diagram. But all of his "dysfunctional" measurement stories are about system relationships that were not taken into account in designing the measurement program. I strongly recommend that before you change a development project or organization read this book, then rethink the change.

Jerry Weinberg, *Quality Software Management, Vol 1: Systems Thinking*, is a study of systems thinking applied to software engineering. Powerful point of view. Jerry also wrote two texts on systems thinking: *General Principles of Systems Design* (with Dani Weinberg) and *An Introduction to General Systems Thinking*. They are both wonderful. I've had the references he sites on my pile of "stuff to read" for quite a while, but while I have great interest, I haven't felt the compelling need. These books are sufficient.

Real Examples of modifying the Development System

CenterRun – Build / CM tool. Addresses fan out and dependency matrix stuff for multi-component systems. Explicitly changes the "development system" by automating stuff that hasn't generally been automated before.

Ken's Estes expanded *TinderBox* to address package / component type build issues. Find it on the Mozilla site at www.mozilla.org/projects/tinderbox/

In requirements management space, compare *Calibre/RM* (from *Starbase*) with *QSS-Doors*. One optimized heavily for traceability, change management, and analysis once the requirements are fixed. One emphasizes the context & conversation of creating requirements.

You might investigate, for an example of defect tracking tuned heavily to support SPI, IBM's work with *orthogonal defect classification* schemes. Unfortunately they move that stuff around quite a bit.

Selected Bibliography

Fred Brooks *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition (2nd Edition),* This is a collection of essays grounded in the development of OS-360. I use it as a litmus test. If I'm talking to someone in the field and they don't at least recognize the name, I know pretty much to expect.

Roger Pressman *Software Engineering: A Practitioner's Approach* and *Software Engineering: A Beginner's Guide.* Pressman consistently does a good job with summary / introduction type texts. Not a bad place to go for a survey of the methods in the domain called "Software Engineering". More readable than most.

Iain Sommerville *Software Engineering (6th Edition).* Sommerville is another text / survey of the domain book. His writing is a bit denser than Pressman, but he's sometimes a bit more thorough.

Richard Thayer, edited or co-edited an excellent series on software engineering topics for the IEEE. These are generally collections of the original papers that put forward a practice, or a model of development. In my opinion, outstanding stuff, although less directly prescriptive than a lot of textbooks. Included are:

- *Software Requirements Engineering*
- *Software Engineering* (M. Dorfman, lead editor)
- *Software Process Improvement*
- *System and Software Requirements Engineering*
- *Tutorial: Software Engineering Project Management* – Strongly recommended, especially in light of the interest of some of the class in "Project Management."
- *Standards, Guidelines, and Examples on System and Software Requirements Engineering*
- *Software Engineering: A European Perspective*
- *Software Engineering Project Management* – Also Strongly recommended, especially in light of the interest of some of the class in "Project Management"

Tom Gilb wrote *Principles of Software Engineering Management*, which includes particular emphasis on iterative development and non-functional requirements. Note that the European SW & methodology standards tend to do a better job with NF requirements, and that you were mostly spared this problem with your system. Gilb's ideas about organizing iterative development are packaged in his "EVO" methodology.

D. L. Parnas wrote some of the basic stuff on product architecture, and software engineering. Much of his work is available in reprint, now in *Software Fundamentals: Collected Papers by David L. Parnas.* There is an admirable clarity to his though, similar in that sense to Nick Wirth, and Edsger Dijkstra. Parnas still teaches, while Dijkstra recently passed away. Nick Wirth I don't know about.

Robert Glass writes extensively on projects that have gone haywire. I particularly like his *Real Time* Software although it is not so much about projects; it is about development systems that weren't up to the task. Another interesting read is *Software Runaways.*

Dwayne Phillips wrote a wonderful book on software engineering project management. *The Software Project Manager's Handbook: Principles that Work at Work*. Dwayne's book makes an excellent entry point.

Jim Highsmith is a recent very influential entry into the "tunable development system" mindset. His book, *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems,* is justifiably well known.

Alistair Cockburn is a pragmatic practitioner. I really like his *Agile Software Development*, which is about "agility" of the approach to development, not one approach that is all things agile. The attributes that make up "agility" in physical space are good, strong analogies for what we'd want in our development systems.

Larry Constantine has published several collections of essays, and more "textbook-y" stuff as well. I particularly like his: *The Peopleware Papers: Notes on the Human Side of Software*

Of course, Jerry Weinberg, explicitly approaches the management of software as a study in systems. His four volume set: *Quality Software Management* is to my mind, definitive in this regard.

Periodicals, Sites and People

Crosstalk – Free journal of defense software engineering. Good, pragmatic reading for all software engineers, once you get past the Jargon.

STQE – STQE magazine, and associated web site. Clearing house for practitioners.

Johanna Rothman – Good, pragmatic site and an outstanding consultant on software engineering management, and projects.

Coyote Valley Software – Brian Lawrence is quite a cool guy. His Coyote Valley site contains some templates and other material about requirements that is quite powerful.

Steve McConnell's / Construx – I have some issues with McConnell's books, and his approach to presenting software engineering. He's got the Micro$oft "we invented it all" bias, and tends to be a bit strident. That said, his resources aren't bad, and he's introduced software engineering disciplines to a lot of people who otherwise wouldn't have known about them.

ACM – Association for Computing Machinery.

IEEE – Institute for Electrical, and Electronic Engineers

Computer Related Risks – A digest of computer related risks, moderated by Peter G. Neumann. They're mostly unintended systems effects – side effects.

Uncommon References on Software and Systems

Politics and the English Language – George Orwell. Common meaning of words leads to clarity. Obfuscation leads to problems. How many development and project problems are due to mixed meanings? What do you do about it?

Alice in Wonderland and *Through the Looking Glass* – Lewis Caroll. "A word means exactly what I want it to mean." And other gems.

Mending Wall – Robert Frost. "Good fences make good neighbors." Really an extended metaphor for the value of strong interfaces in cooperative systems.

True Professionalism – David Maister

The Age of Unreason – Charles Handy

The Hitchiker's Guide to the Galaxy – Most software developers just can't get the hang of Tuesdays. This is another clue test I use. If someone "doesn't get it" I worry about them.

The Goal – Elyhud Goldratt. When I read this book, I realized that I had always thought of software development as a production system. Comes from my engineering training, I suspect.

Methodologies Crib-Sheet

While “XP” and “Agile” methodologies are getting all the press right now, there have been many methodologies and methods of development. I think there are ideas of value in all of them, although they are focused on the “programming” part and tend to make some silent assumptions about the scope of the problem at hand.

SLC – “Systems Life Cycle” IBM’s classic “waterfall” methodology from the 70s. I have found that while the organization of activities is different in time, you can’t really get away from doing, in one way or another, all the stuff in this and related inventories of tasks.

Cleanroom Development – The antithesis of XP’s idea of co-locating developers and testers, this method proposes a “Chinese wall” between the two groups. Interestingly cleanroom too provides marked improvements in productivity, and customer satisfaction, despite being more or less the “anti-XP.”

Mil-STD-12207, & related. – The US military provided a waterfall-ish methodology in the early 80’s as a required methodology for its development contracts (Published as MID-

STD-2167 & 2167a). This methodology evolved into “12207”. The “standard” specifies the lifecycle, and deliverables. Notable because:

- It has converged with European IT methodologies
- It is “systems engineering” heavy, compared to IT or “programming” methodologies.

XP – A recent collection of techniques for developing software, based on a project at Chrysler, which incidentally, didn’t work too well. XP seems to be applicable in some circumstances. Alistair Cockburn has written convincingly on when and where this approach seems to work.

Agile – A generic name for tunable methodologies, popularized lately by Cockburn and Highsmith. Obviously, I think they’ve got the right idea, and have mostly named something that clever people did all along.

Scrum – An iterative, user-involved methodology, for developing IT systems. Strong emphasis on the mechanics of *how* the collaborative next iteration planning takes place.

JAD – “Joint Application Development” The name of a technique for eliciting requirements that also became the name for an entire methodology driven by these requirements and that technique.

“EVO” – Tom Gilb’s evolutionary model, built around addressing risks, mainly to non-functional requirements, in an iterative fashion.

RAD - “Rapid Application Development” Another iterative, collaborative methodology.

Spiral – The name initially applied to Barry Boehm’s iterative methodology. He argues with later adopters of “spiral” in that his methodology explicitly dealt with identified system and program risks, at about 1 / iteration. Most “spiral” methodologies do not.