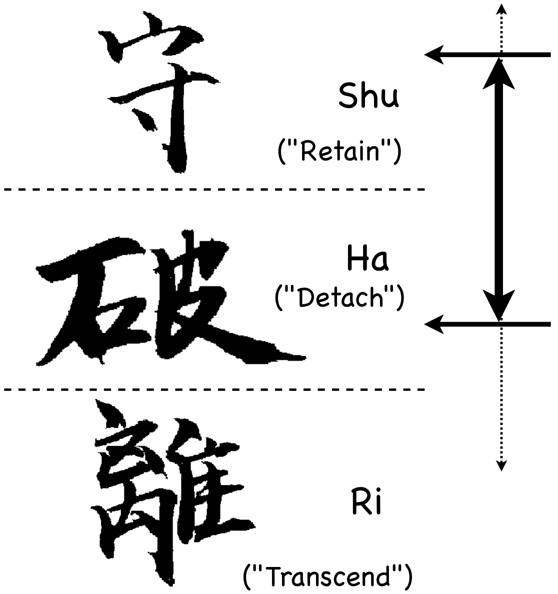
Technical Management Of Software Development

with some lessons from Open-Source

Alex Martelli

What Levels I Address



Shu ("Retain")

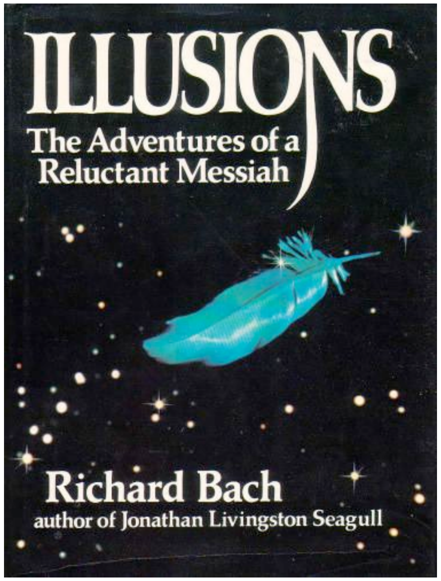
Ha ("Detach")

Ri ("Transcend")

+: let's keep this INTERACTIVE!

Shu: where you're "learning by rote", ONE technique, action by action
Ha: where you're comparing/contrasting techniques, picking the right one case by case
Ri: where you leave "book learning" behind, create your own techniques

Why Do I Teach This...


ILLUSIONS
The Adventures of a Reluctant Messiah
Richard Bach
author of Jonathan Livingston Seagull

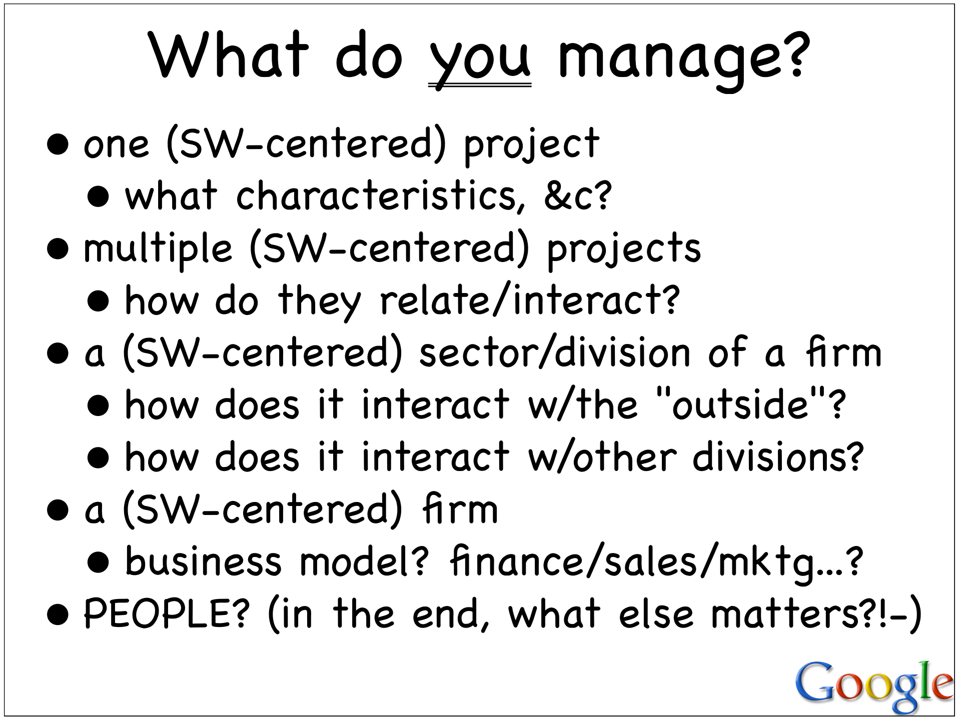
You teach best what you most need to learn.

You're probably familiar with the works of R. Bach, but if you aren't I recommend them highly (and I don't read much fiction...).

Why are you here?

- presumably to learn something interesting, improve what you do, and thus CHANGE it
- so, one meta-issue: how do you change?
- the "system view": you can't "change just one thing" ('cuz everything touches all)
- the pragmatic view: you'd better try to!
  - "big bang" changes are hard and risky
  - AND they don't let you judge easily
  - INCREMENTAL AND ITERATIVE change: it's not smart just for software stuph!-)

I'm very keen on _incremental and iterative_ change -- *one step at a time*. If you change 4 things at once and it works, how do you KNOW which ones really helped? if they don't work, how do you know which ones soured things? If you must try two things A and B, try A first; if it doesn't work, drop it and try B; THEN try both together to see if there's good or bad synergy... well, this doesn't scale (you need 2**N steps to try N things;-), so you need some mixed strategy -- but "big bang" is STILL a disaster!-)



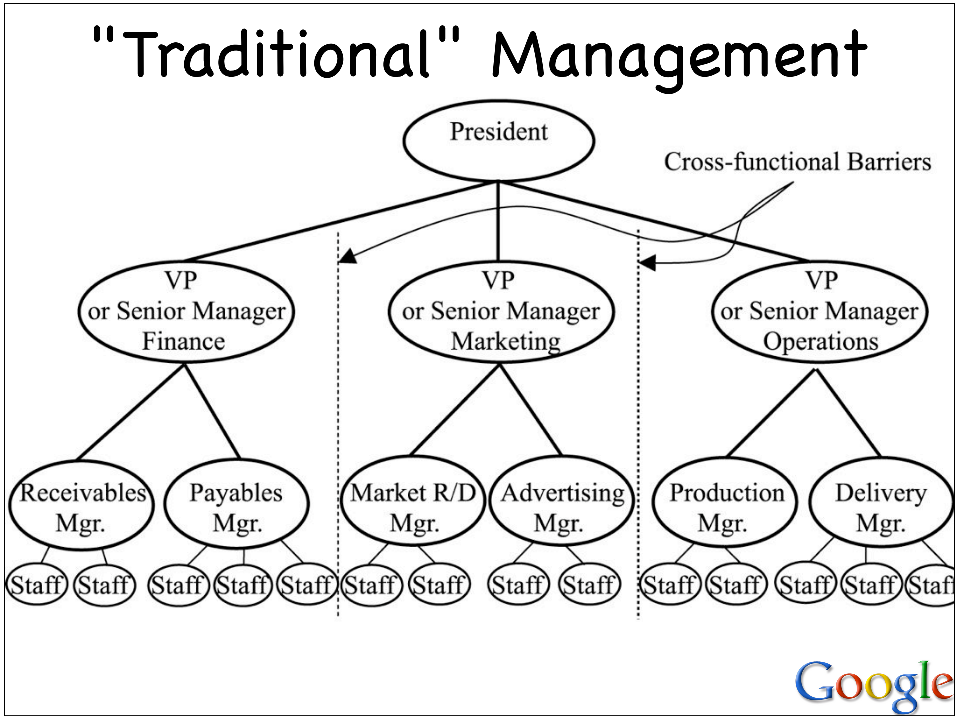
What do you manage?

- one (SW-centered) project
  - what characteristics, &c?
- multiple (SW-centered) projects
  - how do they relate/interact?
- a (SW-centered) sector/division of a firm
  - how does it interact w/the "outside"?
  - how does it interact w/other divisions?
- a (SW-centered) firm
  - business model? finance/sales/mktg...?
- PEOPLE? (in the end, what else matters?!-)

Of course, there are many other possibilities (and "a firm" can be a 3-person startup or a 30,000-person biggie, etc;-). But if you do manage, you DO manage PEOPLE -- the red thread running through my whole talk.

Who IS a Manager?

I actually picked only managers I respect and appreciate for this slide (well, not the PHB, I guess, but he does give me MANY laughs:-). The lower-left corner knight is Swedish King Gustavus Adolphus. The football guys on the right are Italy's Word Champion team from 2006 -- the one holding the cup is Cannavaro, the playing captain; in the top left is Lippi, the non-playing manager/trainer of the winning team, holding the same cup later. Iacocca, Mayer, Jobs, Cox, Hopper, hopefully need no intro:-)



"Traditional" Management

President

Cross-functional Barriers

VP or Senior Manager Finance

VP or Senior Manager Marketing

VP or Senior Manager Operations

Receivables Mgr.

Payables Mgr.

Market R/D Mgr.

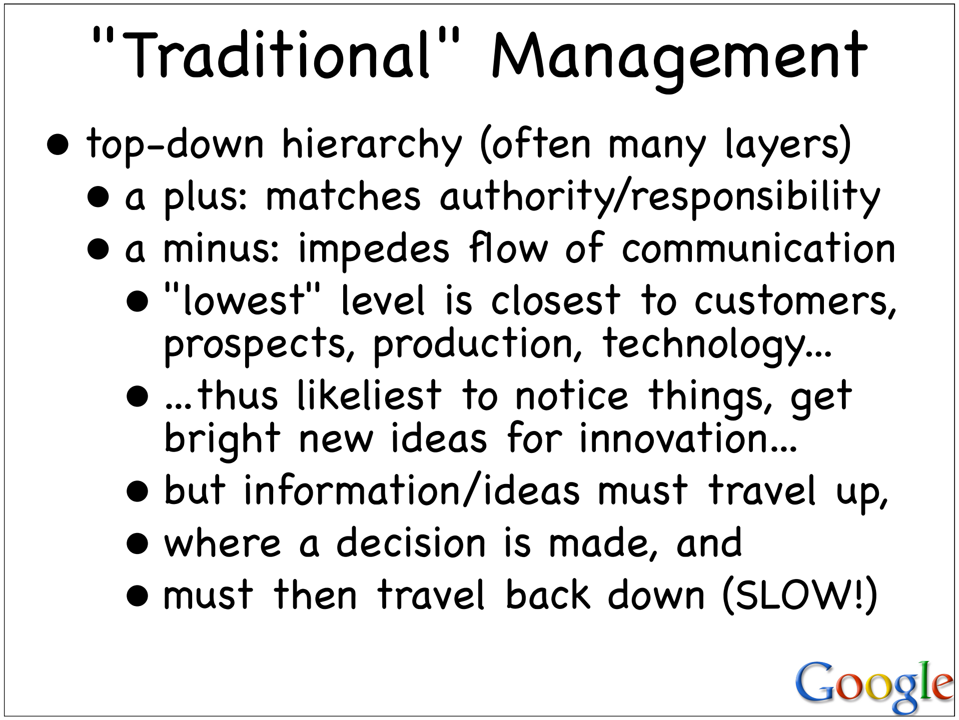
Advertising Mgr.

Production Mgr.

Delivery Mgr.

Staff Staff Staff Staff Staff Staff Staff Staff Staff Staff Staff Staff Staff Staf

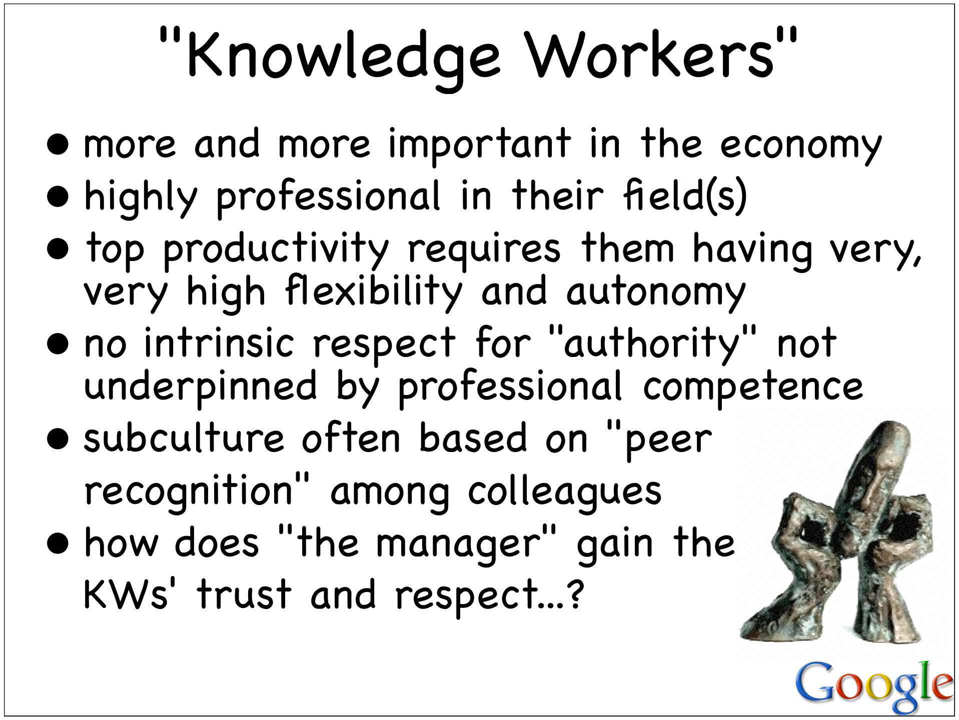
Most real-life traditional mgmt hierarchies are deeper and less elementary than this.



"Traditional" Management

- top-down hierarchy (often many layers)
  - a plus: matches authority/responsibility
  - a minus: impedes flow of communication
    - "lowest" level is closest to customers, prospects, production, technology...
    - ...thus likeliest to notice things, get bright new ideas for innovation...
    - but information/ideas must travel up,
    - where a decision is made, and
    - must then travel back down (SLOW!)

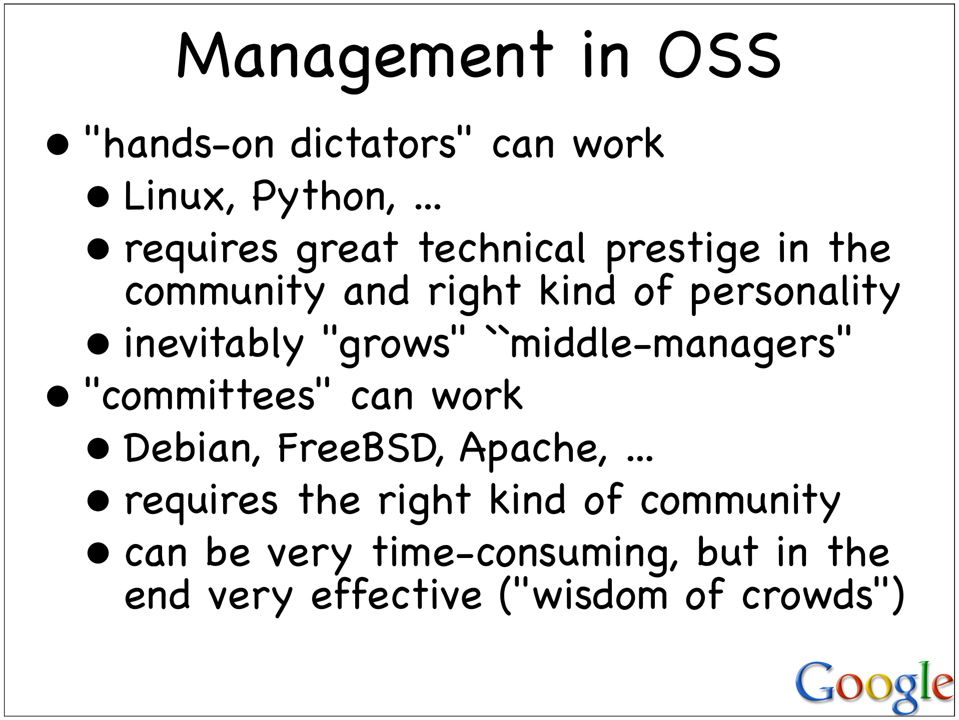
As time goes by, change and innovation are speedier, more disruptive, and more crucial to the firm's success -- so, traditional hierarchies become less and less appropriate (punctilious execution remains very important in many fields, but invariably it must be conjugated with innovation and with riding the endless frothy waves of disruptive-changes...!).



"Knowledge Workers"

- more and more important in the economy
- highly professional in their field(s)
- top productivity requires them having very, very high flexibility and autonomy
- no intrinsic respect for "authority" not underpinned by professional competence
- subculture often based on "peer recognition" among colleagues
- how does "the manager" gain the KWs' trust and respect...?

It's a new term, but the concept is very old (though it accounts for a growing fraction of the economy as time goes by). Given key workers who live in a subculture where technical competence is the main determinant of status, how does the manager gain their trust and respect? (This is known in the trade as "foreshadowing":-).



Management in OSS

- "hands-on dictators" can work
  - Linux, Python, ...
  - requires great technical prestige in the community and right kind of personality
  - inevitably "grows" ``middle-managers"
- "committees" can work
  - Debian, FreeBSD, Apache, ...
  - requires the right kind of community
  - can be very time-consuming, but in the end very effective ("wisdom of crowds")

An effective "dictator" in fact ends up delegating A LOT -- unilateral, anti-consensus decisions are rare (though admittedly pretty crucial when they DO happen:-).

Managing Risk

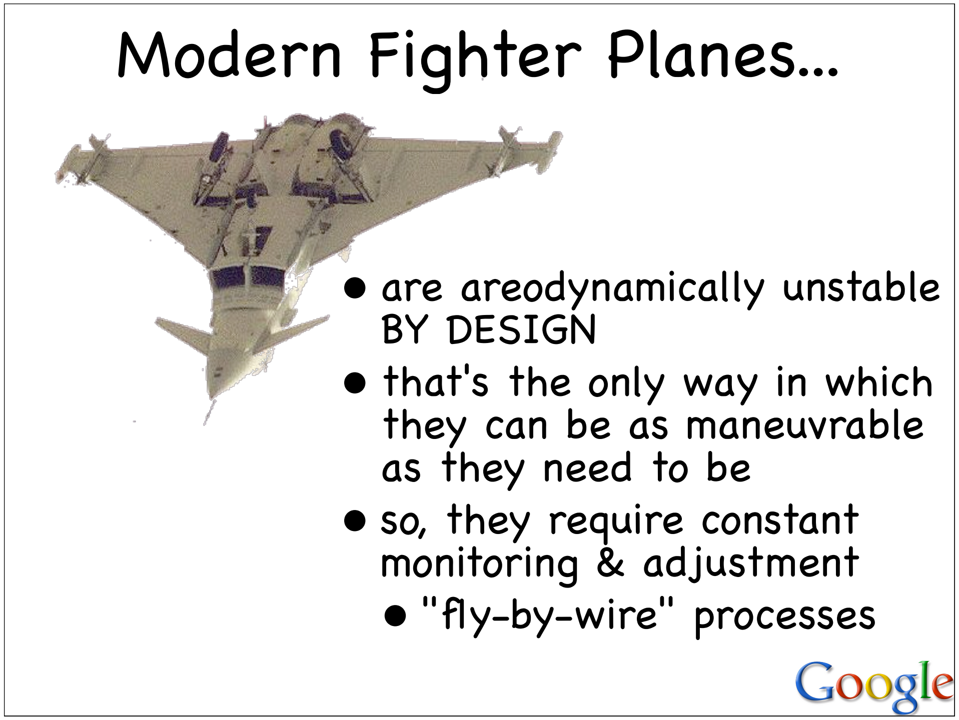
- "actively attack risks, otherwise they will actively attack you"
- risk 1: building the wrong thing
- risk 2: building the thing wrong
- to fight risks, they have to be VISIBLE (use TRANSPARENCY)
- cheat: fight them when they're little and can't yet fight back (ANTICIPATE risks)
- just one downside: will this really help your career? (do you truly care?)

We often reward "heroic effort" -- and not the foresight and planning that make the heroism unnecessary and offer much better ROI. Bosses and customers used to being shielded from problems and risks may freak if you use transparency to display all risks and problems with your projects, even though this puts them in control and maximizes chances of success and quality. Do you care about being rewarded and praised, or are you out for Self-Actualization and Transcendence? (See Maslow Pyramid later)

How Much Control?

- if a project starts in "chaos" (zero "control"), we easily observe that adding a little control increases project efficiency
- so, if a little control is good, more must be better, right...?
- wrong! there can easily be too much
  - the "sweet spot" for the "just right" amount of control varies by project
    - how "predictable" is the project?
    - how innovative (thus unpredictable)?
    - how changeable requirements/needs?

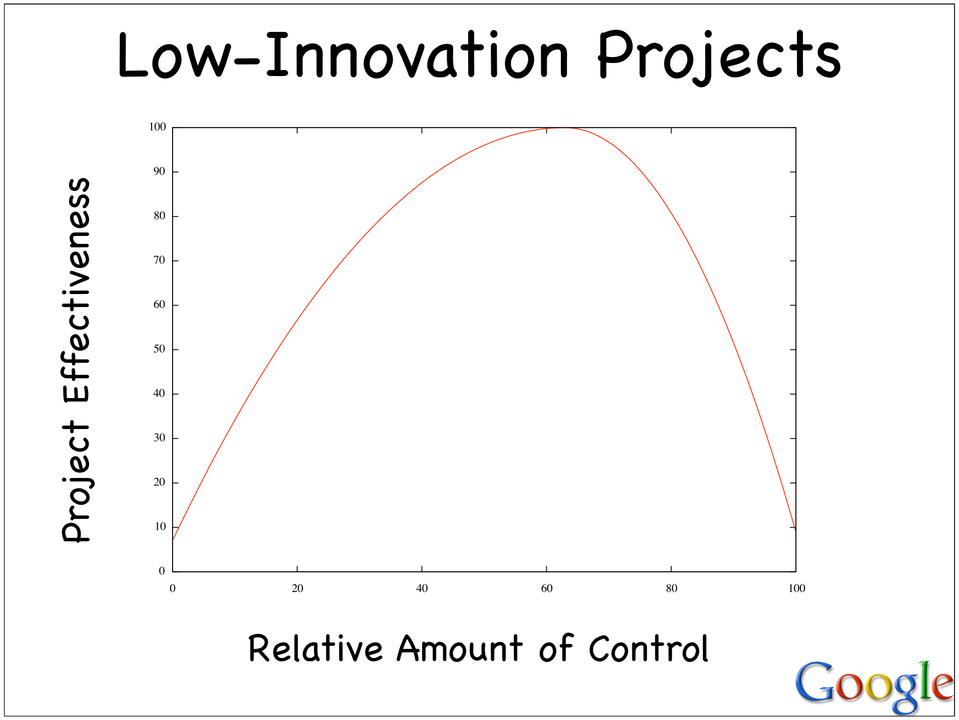
This truth is "fractal" -- applies (in different but similar ways) at all levels, from projects to huge firms.



Modern Fighter Planes...

- are areodynamically unstable BY DESIGN
- that's the only way in which they can be as maneuvrable as they need to be
- so, they require constant monitoring & adjustment
  - "fly-by-wire" processes

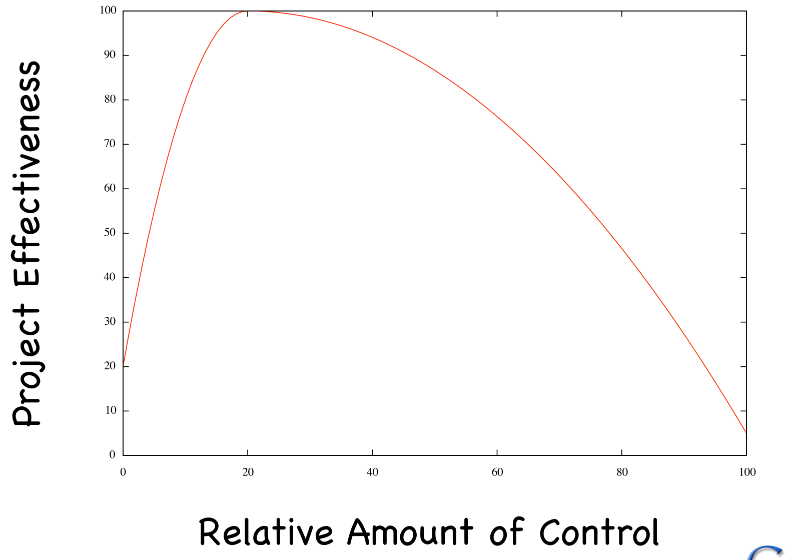
AKA, "the price of [maneuverability and fast response to change] is constant vigilance".



Low-Innovation Projects

Again, not just "projects" -- conceptually applies to any endeavor and to whole organizations.

High-Innovation Projects

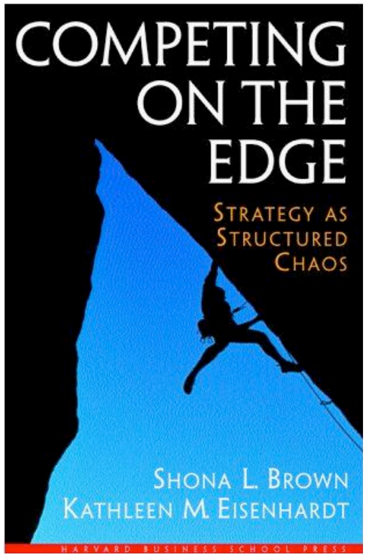


6σ vs Innovation at 3M

- 6σ reduced costs, increased profits...
- ...BUT innovation may be flagging at 3M!
- 6σ: control, measure, predict, plan, execute
- but innovation cannot be made repeatable, controlled, predicted, nor sensibly measured before the fact...

Just one example (and far from an open-and-shut case!), but rather recent and well-publicized.

Not Just For Projects...

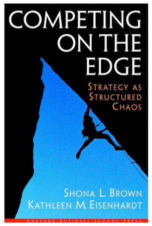


...traditional approaches to strategy often collapse in the face of rapidly and unpredictably changing industries ... because they over-emphasize the degree to which it is possible to predict ...

... change is the striking feature of contemporary business ... the key strategic challenge is managing that change.

Brown was a partner at McKinsey (leader of Global Strategy Practice) when she wrote this book -- soon after she joined Google, where she's now Senior Vice President for Business Operations. Eisenhardt is Professor of Strategy and Organization at Stanford.

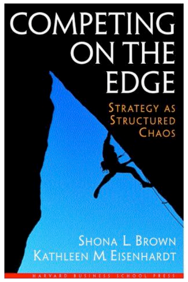
Managing Change



- progressively better choices...:
  - REACTING to change
    - monitor the landscape assiduously
    - keep the organization flexible
  - ANTICIPATING change
    - move proactively with it
  - LEADING change
    - innovate yourself, before others do it!
- ...and many more bad choices (whine about it, resist it, undermine it, go into denial, ...)

(the list of bad choices for managing change is my own idea, not necessarily the authors':-)

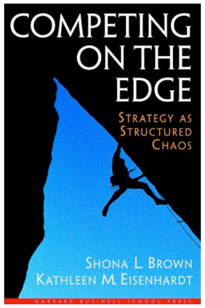
Competing on the Edge is...:



- unpredictable
  - it's about surprise, not plans!-)
- uncontrolled
  - many moving on their own
- inefficient
  - WILL make mistakes -- & fix
- proactive
  - definitely NOT passive or reactive!-)
- continuous
  - it's a rhythm of moves over time
- diverse
  - and diversity makes it robust

...and you're always risking falling off either side of the edge... fun, fun, fun!-)

Time-Pacing



- event-pacing is reactive
- time-pacing structures chaos
  - just a bit:-)
  - stops from changing too often
  - gets you into "tempo" (rhythm)
  - forces periodical "stop what you're doing, assess effectiveness, adjust strategies"
  - (occasional even-pacing still needed:-)
- warning: may easily create a stressful environment -- act to alleviate that!

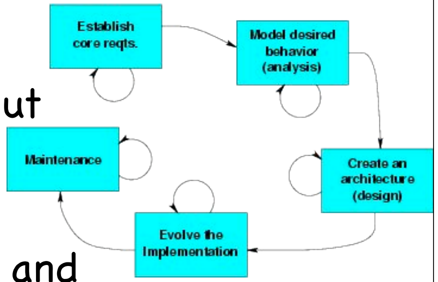
Setting fixed-time, periodic targets helps: that's unintuitive but profoundly true -- OSS experiences validating this include Ubuntu and FreeBSD.

And, Back to Tactics...:

- Waterfall: aim, aim, aim, aim, aim, FIRE!
- Chaos: fire!, fire!, fire!, fire!, OOPS, sorry!...
- Agile: aim, fire, adjust; aim, fire, adjust; aim, fire, adjust; aim, fire, adjust; ...
  - iterative and incremental development
  - "release early, release often"
  - Agile methods try to extract the aspects that work and apply them with discipline
  - ...and pick coding and testing techniques well-suited to the reality at hand

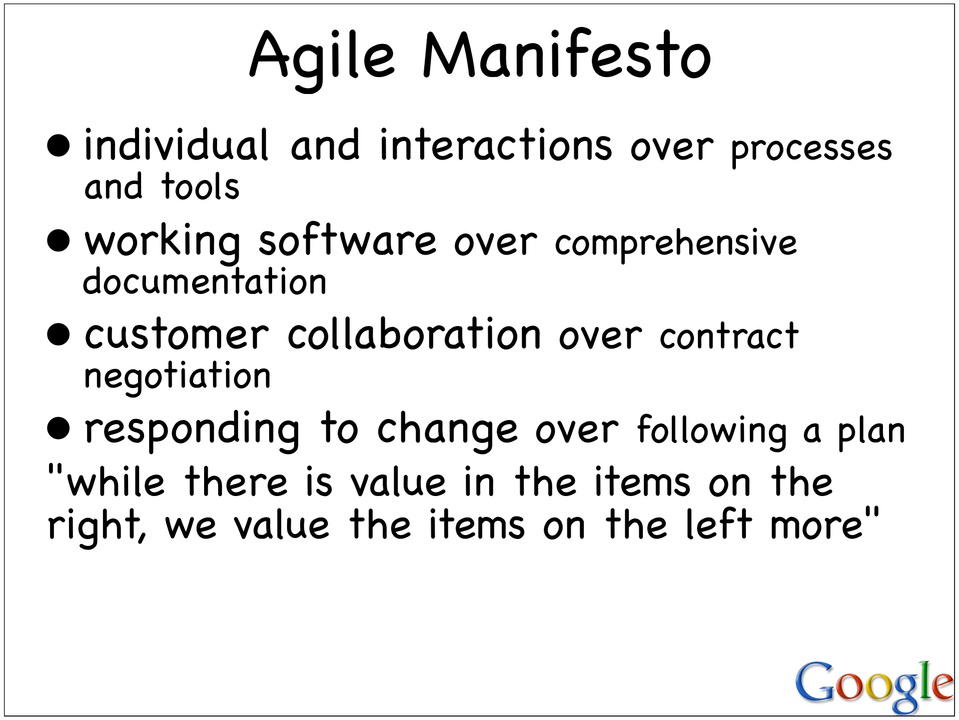
yes, there _are_ degrees of "agility" (but no: Chaos isn't one such degree -- it's a "falling of the edge":-).

Macro & Micro Processes


Establish core reqts.
Model desired behavior (analysis)
Create an architecture (design)
Evolve the implementation
Maintenance

- macro-level has some "flow", risk-focused (but also iteration, etc); it's quite manager-centric
- micro-level is iterative and incremental (hopefully in a spiral!-) -- it's quite developer-centric

The concept (of dual macro/micro processes) was extended and refined all the way to RUP (Rational's Unified Process) -- it never ceased being controversial though (it's fired upon from both sides, which is often a good sign:-).

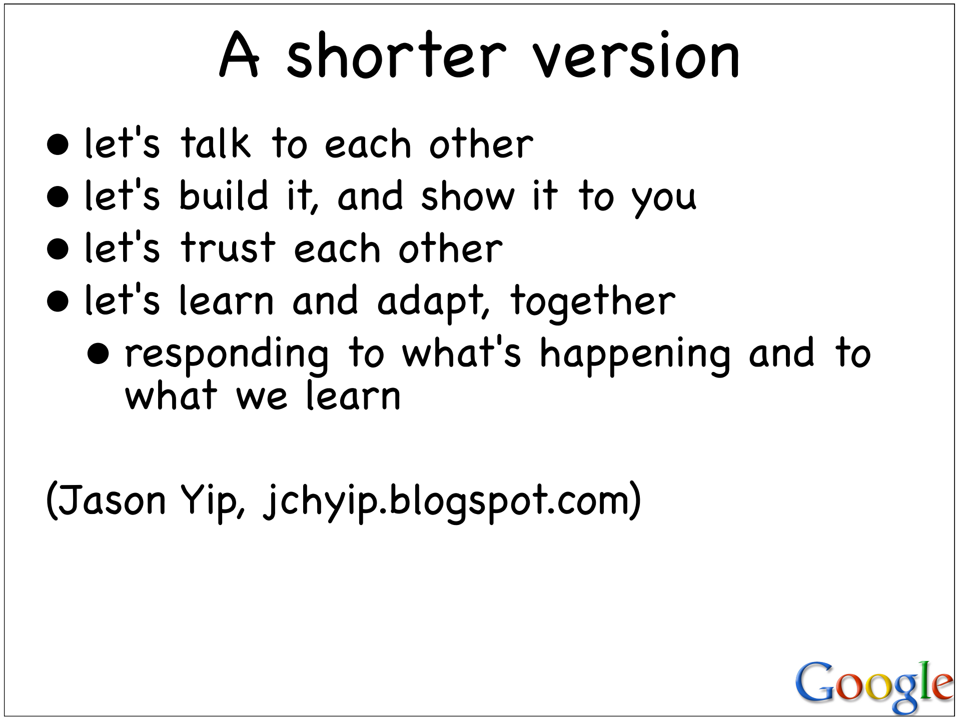


Agile Manifesto

- individual and interactions over processes and tools
- working software over comprehensive documentation
- customer collaboration over contract negotiation
- responding to change over following a plan

"while there is value in the items on the right, we value the items on the left more"

Craftily worded so it's hard even for a dyed-in-the-wool waterfaller to disagree:-)

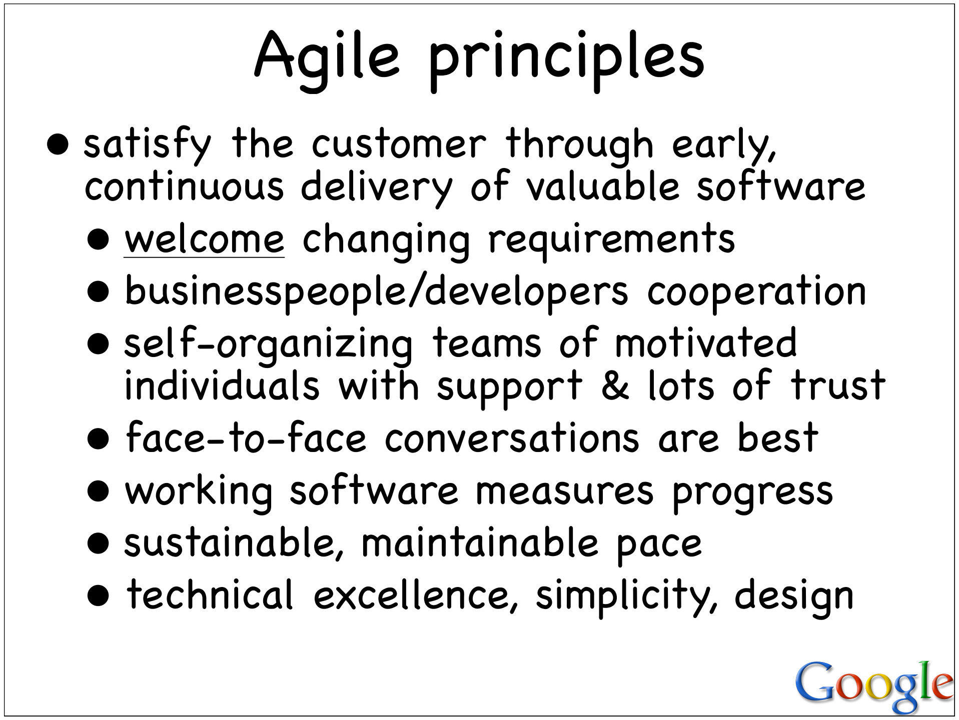


A shorter version

- let's talk to each other
- let's build it, and show it to you
- let's trust each other
- let's learn and adapt, together
  - responding to what's happening and to what we learn

(Jason Yip, jchyip.blogspot.com)

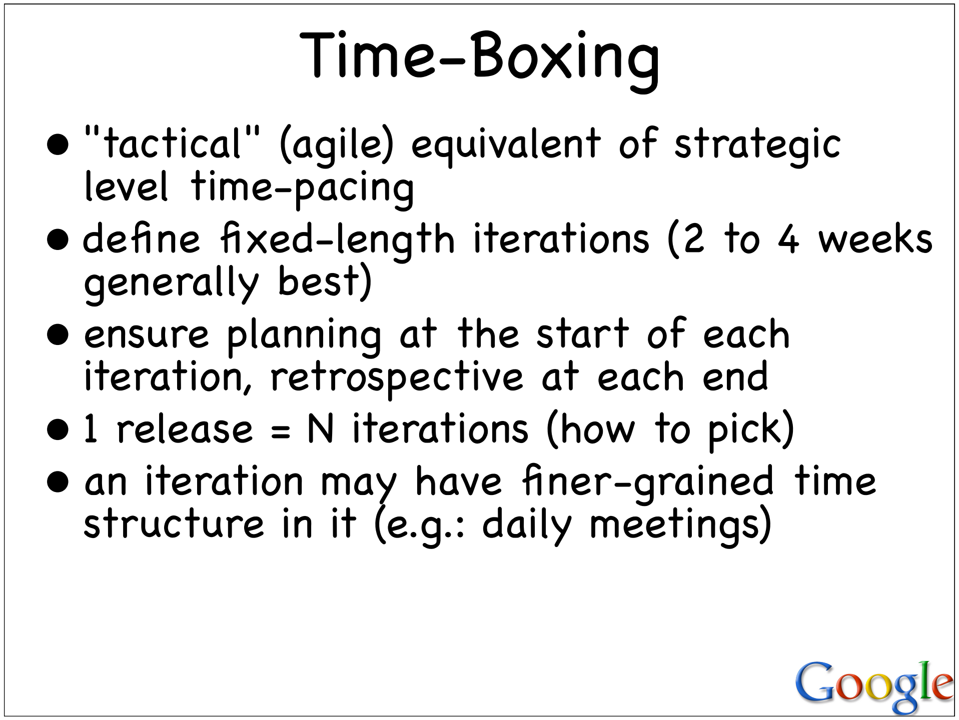
very worthy blog -- search it for these sentences for more detail



Agile principles

- satisfy the customer through early, continuous delivery of valuable software
  - welcome changing requirements
  - businesspeople/developers cooperation
  - self-organizing teams of motivated individuals with support & lots of trust
  - face-to-face conversations are best
  - working software measures progress
  - sustainable, maintainable pace
  - technical excellence, simplicity, design

do you see where the principles match with the points in the Manifesto?



Time-Boxing

- "tactical" (agile) equivalent of strategic level time-pacing
- define fixed-length iterations (2 to 4 weeks generally best)
- ensure planning at the start of each iteration, retrospective at each end
- 1 release = N iterations (how to pick)
- an iteration may have finer-grained time structure in it (e.g.: daily meetings)

in fact, time-whatevering can help at many levels -- in short, it helps achieve *sustainable rhythm*, and humans thrive on that (vs stasis or chaotic flapping). The "sustainable" part of it DOES need nurturing and proactive support, though!-)

Whole-team Ownership

- no "my code"/"your code": it's all OUR code
- enhances team's "bus number", flexibility
- how do you get there?
  - uniform team style (& helpful tools)
  - strong, growing test-suite (automated!!!)
  - enabling "refactor mercilessly" culture
  - mandatory code reviews (or pair programming... -- or, ideally, both)
- the "problem" of specialists
  - "grow" them into "semi-generalists"

Of all active practices, maybe the one farthest from "classic" -- and yet most crucial and productive.

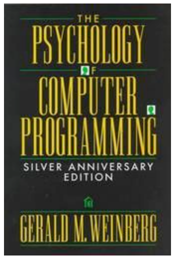
Pair Programming

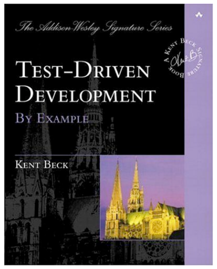
- a highly contentious practice -- but very solid if local cultural conventions can possibly stand it! (perf reviews, too...)
- not necessarily a substitute for mandatory code reviews (the pair programs "as one" in some ways, chiefly in "big picture" issues)
- nevertheless presents many advantages (bus number, intrinsic training, favorable impact on team ownership, style, ...)

You can definitely do Agile without PP -- but if you can use PP (maybe only some of the time) it can help A LOT.

Test-Driven Development

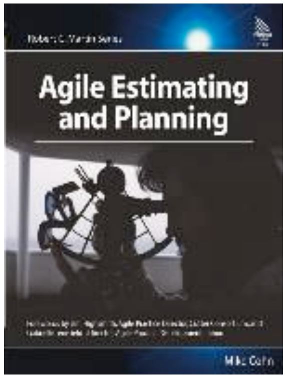
- "prepare the tests in advance of testing, and, if possible, in advance of coding" (a quote from _Weinberg, 1971_...!-)
- a thorough development of this idea into a methodology, with practical examples and explanations:

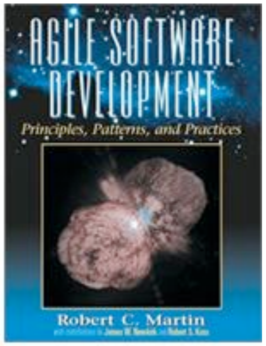

THE
PSYCHOLOGY OF COMPUTER PROGRAMMING
SILVER ANNIVERSARY EDITION
GERALD M. WEINBERG



"Pure" TDD is more often talked about than practiced, but a large dose of it can enhance your development A LOT. In particular, ALWAYS write automated tests that reproduce a bug BEFORE you work to find and fix the bug (you get "regression testing" nearly for free, this way).

Good Agile Intro Books

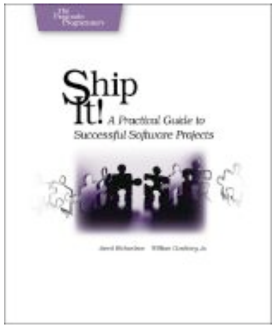


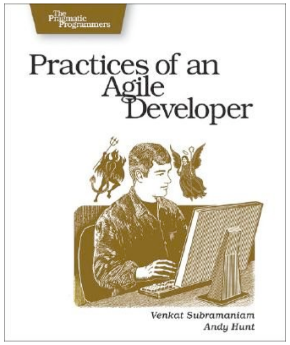

AGILE SOFTWARE DEVELOPMENT
Principles, Patterns, and Practices
Robert C. Martin

Larman: good, wide, general survey
Cohn: focus on plans, schedules, estimations
Martin: great mix for developer types

I'd recommend Martin's book to anybody who develops software -- and any manager of development should ensure they know enough technical aspects to follow that book, too! Larman is really meant for managers, not developers. Cohn is the best book I know about planning, scheduling and estimating software development (mostly for project managers, but really all audiences).

Pragmatic & for Developers



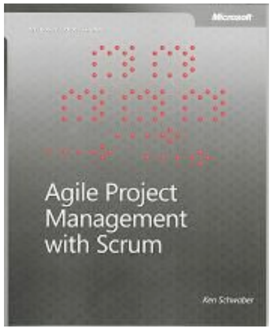


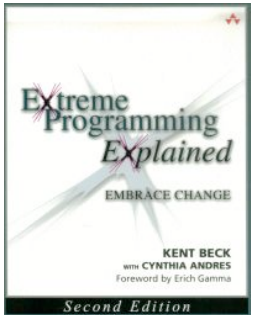
Richardson-Gwaltney: not just for "Agile" projects; very "hands-on", very practical

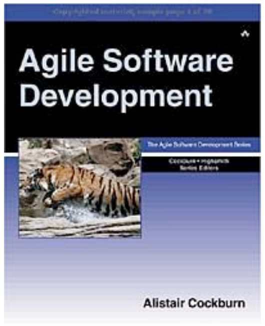
Subramaniam-Hunt: "devil/angel" takes on 45 agile practices, entertaining AND very useful

Both books meant mostly for developers, but won't hurt managers one bit ("Ship It!" very useful for project managers in particular, "Practices" for anybody wanting "a taste of agile development" even if without much technical background)

More Advanced Agile Books







Schwaber: focus on the Scrum process

Beck: focus on Extreme Programming ideas

Cockburn: wide, deep, profound survey

Scrum (mostly with a project-manager orientation) and XP (mostly with a developer-orientation) are probably the two most widespread Agile methods. Cockburn's book is the deepest and most profound survey on Agile that I've ever read: I recommend rereading it every year or two, you'll keep getting more and more out of it as you grow with experience (thus, a true classic).

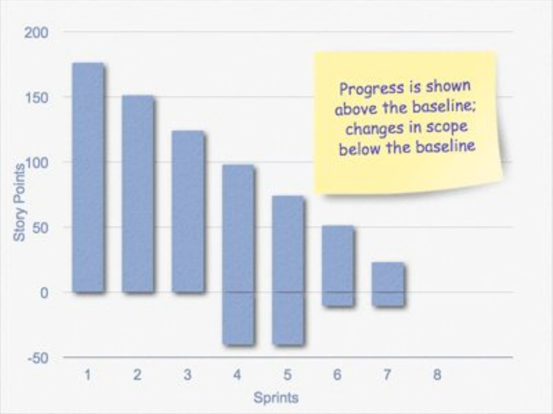
Scheduling and Planning

- start LOW-tech (a spreadsheet, or whiteboard+stickies, or index cards...)
- pick very fine-grained tasks (to combat developers', and your own, optimism!-)
- schedule vacations, holidays, training, sick days (proactively fight to avoid burnout!) -- what can't be scheduled deterministically can be scheduled "statistically" (adv: schedule RANGES, NOT "point estimates" w/unknown uncertainties)
- heed Cohn's advice: estimate size, derive duration (&, size in arbitrary units * velocity)

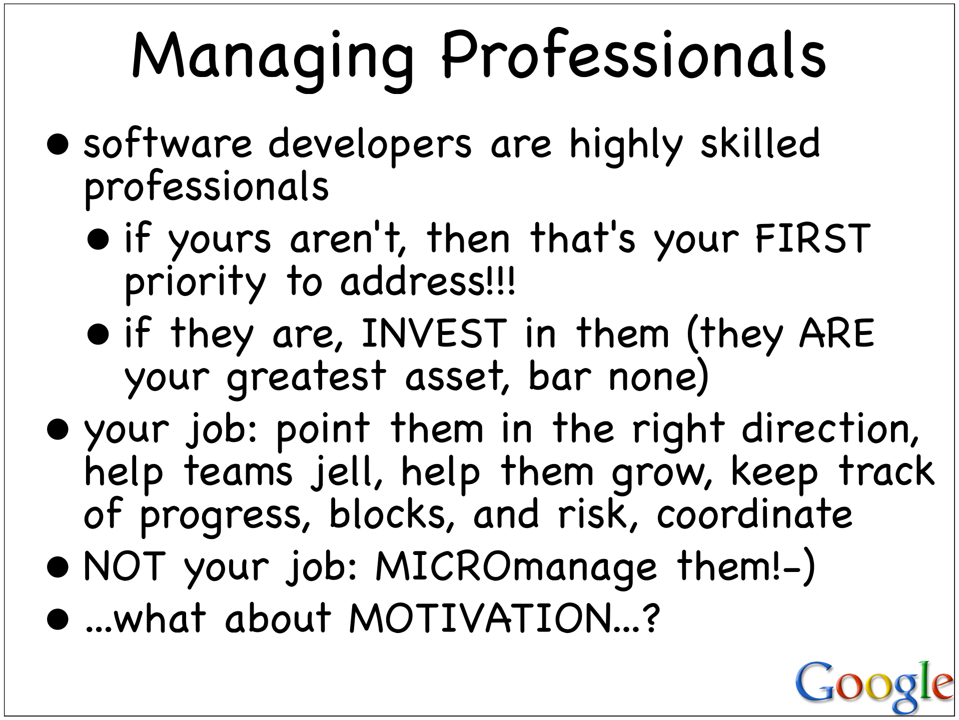
Cohn has many, many more useful pieces of advice -- but DO keep the two bits about fine-grained tasks and "schedule everything" as top priorities!-)

The Burndown Chart

- shows per-iteration progress AND changes in scope in one bird's-eye view



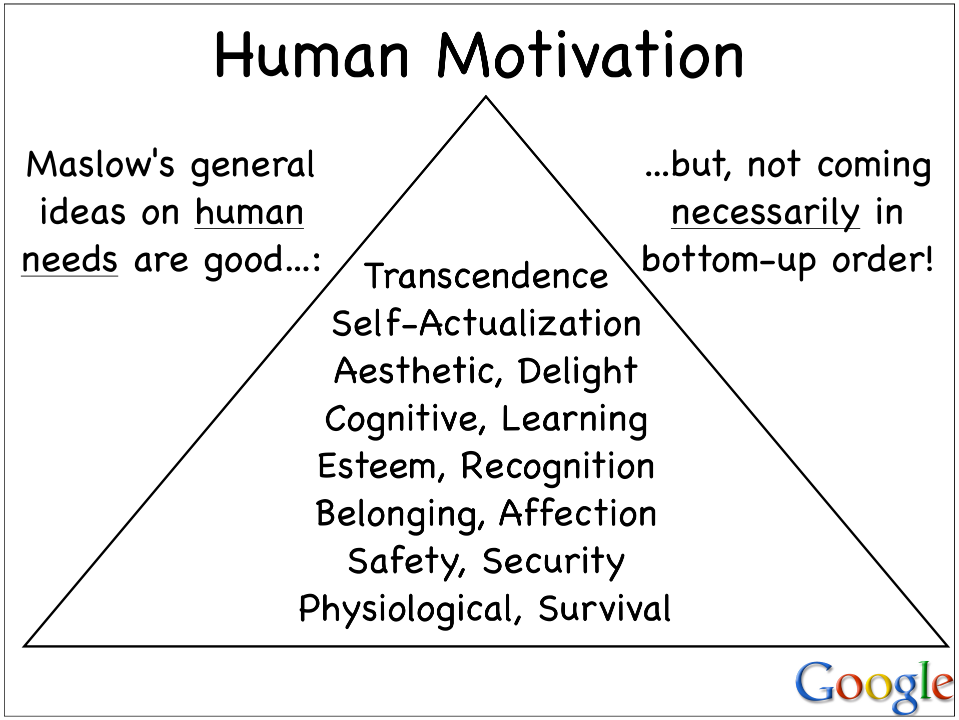
There are many form of "burn-down charts" (plotting "remaining efforts in project's backlog" versus time): I like the ones, like the one I show here, where added requirements show "the bottom falling away" (not the developers' fault...:-).



Managing Professionals

- software developers are highly skilled professionals
  - if yours aren't, then that's your FIRST priority to address!!!
  - if they are, INVEST in them (they ARE your greatest asset, bar none)
- your job: point them in the right direction, help teams jell, help them grow, keep track of progress, blocks, and risk, coordinate
- NOT your job: MICROmanage them!-)
- ...what about MOTIVATION...?

Hiring well is VERY difficult (I've heard good things about Joel's latest book, "Smart and Get Things Done", which is right on the subject, but haven't read it myself yet at the time of writing). Removing from the team/organization people that are wrong for it is, or SHOULD be, even harder (you ARE messing with another person's life -- if that doesn't make you shudder, I'm worried), but nevertheless crucial for the success of the project, team, firm, etc.

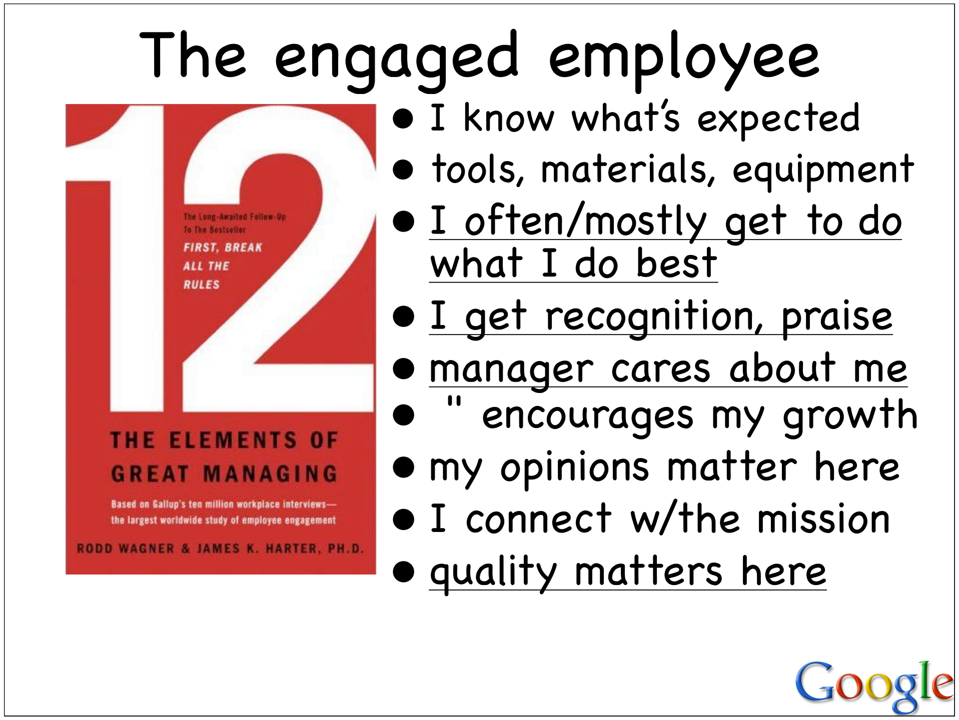


Human Motivation

Maslow's general ideas on <u>human</u> <u>needs</u> are good...:

...but, not coming <u>necessarily</u> in bottom-up order!

Most places quote the simpler, older 5-level version, but I'm very keen on this one (from Maslow's later works). "Transcendence"s main manifestation is "helping others *for the sake* of helping others" (not for reciprocity, belonging, to get praise, etc), and I concur it IS the highest peak of human achievement, beyond even self-actualization (that may be part of what makes it so hard for many to accept at face value when it does show up...). As for order, it's obviously NOT linear -- e.g., soldiers often risk their lives (survival and safety levels) for the sake of their "unit" (belonging, recognition), etc.

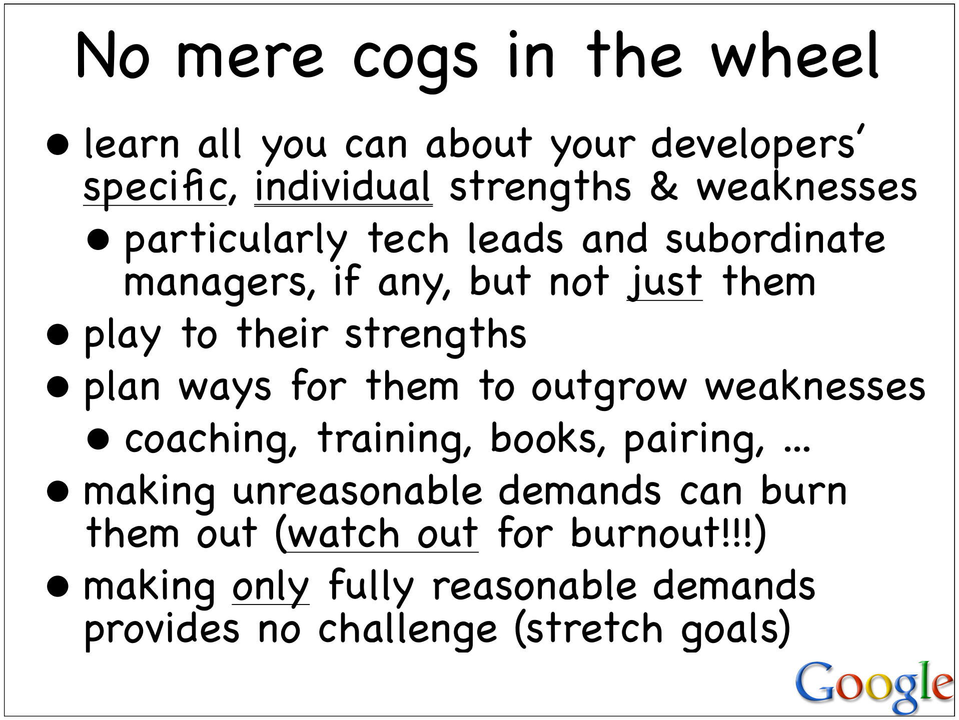


The engaged employee

12
The Long-Awaited Follow-Up To The Bestseller
FIRST, BREAK ALL THE RULES
THE ELEMENTS OF GREAT MANAGING
Based on Gallup's ten million workplace interviews—the largest worldwide study of employee engagement
RODD WAGNER & JAMES K. HARTER, PH.D.

- I know what's expected
- tools, materials, equipment
- I often/mostly get to do what I do best
- I get recognition, praise
- manager cares about me
- " encourages my growth
- my opinions matter here
- I connect w/the mission
- quality matters here

The exact wording of the 12 principles is claimed as copyright by the book's authors, so I've paraphrased the ideas behind them (ideas can't be copyrighted, thanks be!) --actually, only the nine out of 12 that I think matter most in this context (the authors correctly point out how difficult it is to fit considerations of _compensation_ in there!). Exercise: match each element to the relevant points in Maslow's Pyramid.



No mere cogs in the wheel

- learn all you can about your developers' specific, individual strengths & weaknesses
  - particularly tech leads and subordinate managers, if any, but not just them
- play to their strengths
- plan ways for them to outgrow weaknesses
  - coaching, training, books, pairing, ...
- making unreasonable demands can burn them out (watch out for burnout!!!)
- making only fully reasonable demands provides no challenge (stretch goals)

Managing professionals as if they were "interchangeable parts" is the single worst source of management mistakes. Professionals (good ones, especially) thrive on challenge -- but you have to balance that (stretch goals) with avoiding their burning out.

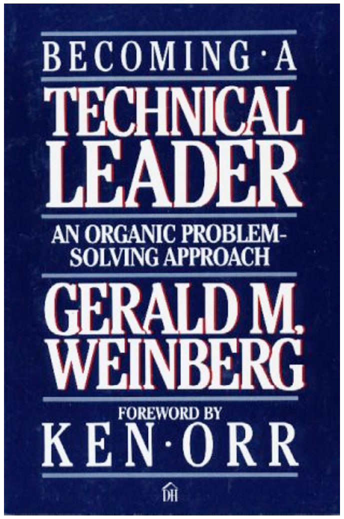
Can Just Anybody Do It?

- probably not -- it takes some "talent"!
  - there are techniques you can learn,
  - and experience and reflection will help,
  - but some things, you have, or you don't
- you must <u>really</u> CARE about your people
  - and ALSO care about the team's <u>results</u>
- you must be able to deal with interrupts
  - time management techniques are good, but no panacea -- interrupts WILL occur
- you must inspire, feel, and nurture TRUST

This is contentious -- Weinberg, e.g., claims any technical person can become a better technical leader than they are right now, if the motivation for so doing is "pure" (see next slide). I think there are some "inborn" prereqs, but it's hard to prove either way.

Becoming a Leader

- "is not something that <u>happens</u> to you, but something you <u>do</u>"
- "all leading is leading change": Motivation, Organization, Ideas
- "always be sincere (whether you mean it or not)"



BECOMING·A
TECHNICAL
LEADER
AN ORGANIC PROBLEM-SOLVING APPROACH
GERALD M. WEINBERG
FOREWORD BY
KEN·ORR

The third point is obviously a funny quip -- but there's depth behind it, just like there is behind every funny anecdote and homily in this excellent book. A book that can easily be appreciated at any level of experience but will give you more and more in proportion to what you have to start with, and what you put into reading, studying, and applying it in your life.

Trust...

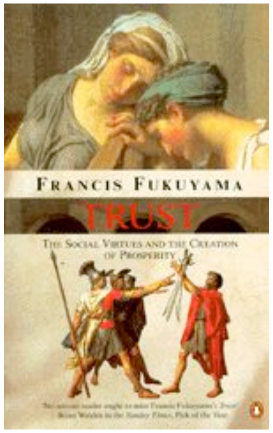
Trust, and the consequent ability to delegate, are common to all area of "knowledge work" (or so I firmly believe, though my first-hand personal experience is limited to electronics and computers); here I illustrate the issue with a brief snippet from an immortal movie, "All the President's Men".

Trust begins at home

- to be worthy of others' trust, you must first be worthy of your own!-)
  - This above all: to thine own self be true,
    And it must follow, as the night the day,
    Thou canst not then be false to any man
  - don't tell yourself little white lies...!
  - "above all, don't fool yourself, don't say it was a dream, your ears deceived you: don't degrade yourself with empty hopes like these" (C. Cavavy, "The god forsakes Anthony")

This kind of consideration applies more widely, as Weinberg points out well -- e.g., a prereq for respecting others is to respect yourself. I focus on trust because it's so incredibly important -- and "being worthy of self-trust" (the courage to tell yourself the truth and live by your principles) sometimes a huge challenge to all managers (and others).

Trust Builds Firms, Economies



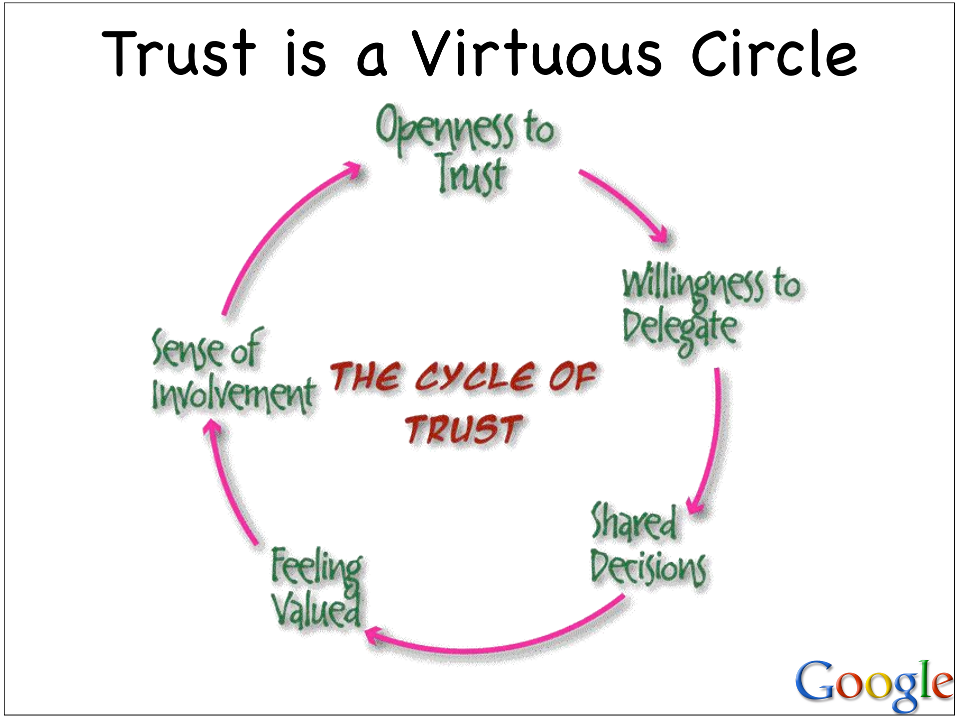
The satisfaction we derive from being connected to others in the workplace grows out of a fundamental human desire for recognition.

Fukuyama's take on trust is chiefly that of the historian, though I've picked a quote that ties back to the issue of motivation. You really should read the book, but the key idea is that the ability to trust strangers is present to different degrees in different cultures, and is a prereq for the modern form of capitalism (as opposed to purely family-based, thus smaller, enterprises).

Trust...

- is mutual, and built up over time
- you must earn & deserve developers' trust
  - technical ability & "technical currency"
  - true, not faked, interest in them as individuals, within and outside work
- they must earn & deserve yours
  - tech skills, integrity, goal-focusing
  - but: always start "trusting by default"!
    - helpful prereq: hire VERY selectively!-)

Pointing out the most important issues in the two-way exchange of trust between manager and developers -- and the (I hope:-) non-obvious connection back to hiring. You don't hire just for technical excellence -- character and personality are ALSO key.



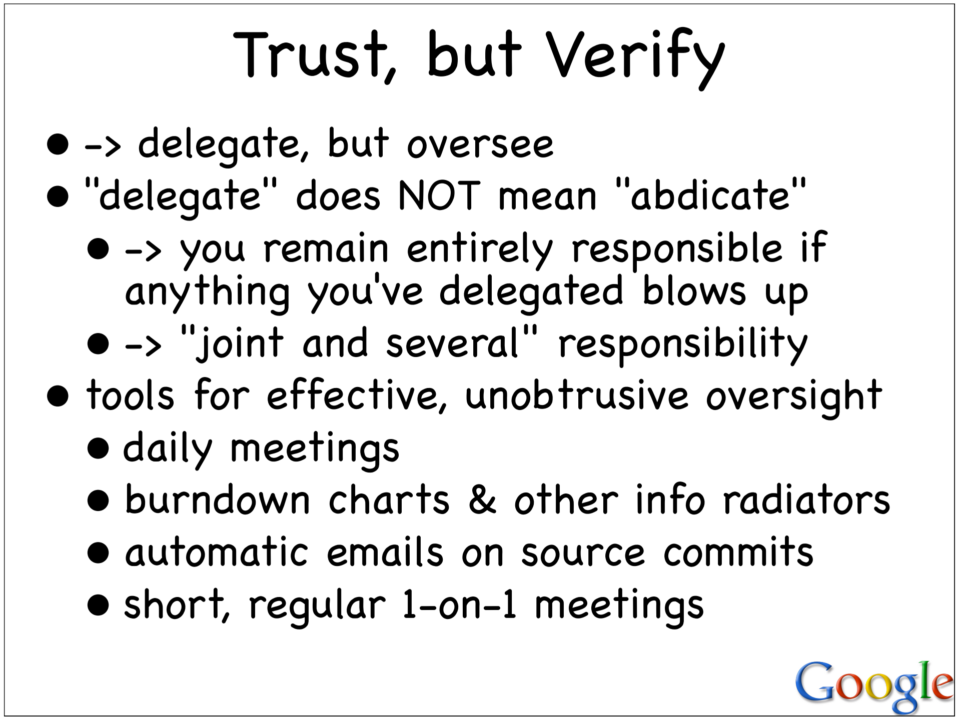
Trust is a Virtuous Circle

Trust can build on itself, gradually and progressively (of course, a breach of trust can similarly snowball into very justified MIStrust; perhaps the most common example in our context is the manager that tries to grab credit for himself, or shunt blame to developers, rather than properly taking his responsibilities and giving credit to his people).

More about Trust...

"A fascinating, provocative, and important book" —Douglas R. Hofstadter, author of Gödel, Escher, Bach
THE Evolution OF Cooperation
ROBERT AXELROD
Foreword by STEPHEN R. COVEY
Author of The 7 Habits of Highly Effective People
THE SPEED OF TRUST
The One Thing That Changes Everything
"Red-hot relevant!"
STEPHEN M. R. COVEY
WITH REBECCA R. MERRILL
The Origins of Virtue
Human Instincts and the Evolution of Cooperation
MATT RIDLEY
author of The Red Queen

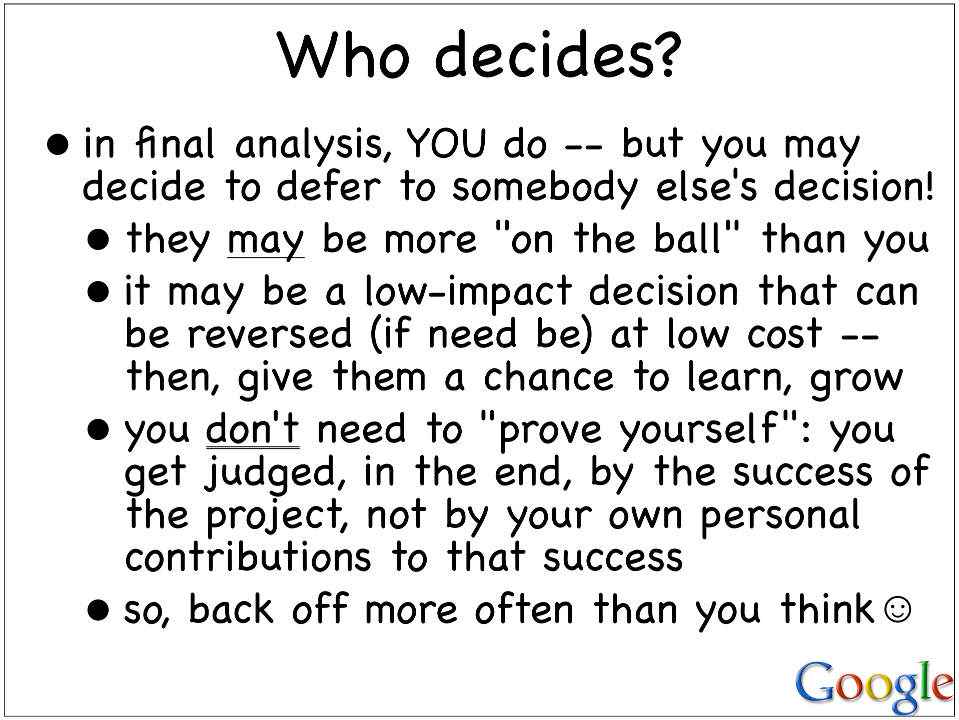
Axelrod works on simple but interesting mathematical models and simulations of "prisoner dilemma" (and has further published more work on the subject). Covey's is a typical, high-quality "self-help-style" book (like his dad's:-). Ridley starts from a biology (particularly, genetics) perspective, but in this book tries to tie in threads from economics (including game theory), sociology, anthropology, ... You shouldn't read JUST about management theory and software development issues, after all...!-)



Trust, but Verify

- -> delegate, but oversee
- "delegate" does NOT mean "abdicate"
  - -> you remain entirely responsible if anything you've delegated blows up
  - -> "joint and several" responsibility
- tools for effective, unobtrusive oversight
  - daily meetings
  - burndown charts & other info radiators
  - automatic emails on source commits
  - short, regular 1-on-1 meetings

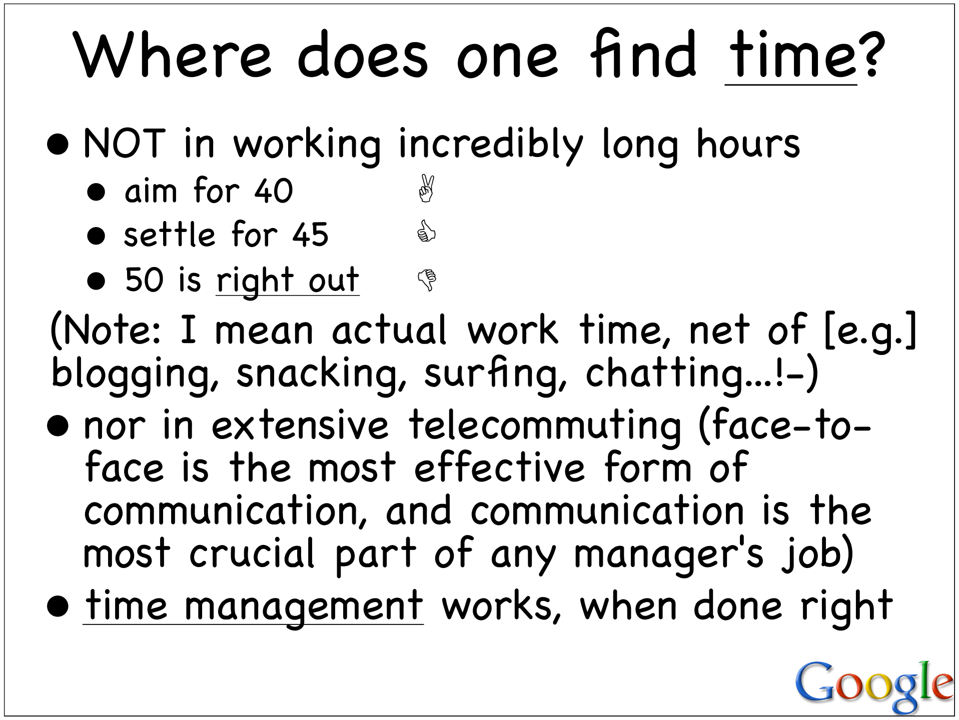
Yeah, the title's a quote out of context:-). But the key is: you keep responsibility for all you delegate; the best mechanisms to let you keep an eye on things are just the same that help *project transparency* all around, plus 1-on-1 meetings (which you should have anyway) to find out about things that escape the "transparency" mechanisms (mostly "people issues"). A formal mentor relationship *outside* the reporting chain is also extremely useful (doesn't help YOU, but helps the FIRM...).



Who decides?

- in final analysis, YOU do -- but you may decide to defer to somebody else's decision!
  - they may be more "on the ball" than you
  - it may be a low-impact decision that can be reversed (if need be) at low cost -- then, give them a chance to learn, grow
  - you don't need to "prove yourself": you get judged, in the end, by the success of the project, not by your own personal contributions to that success
  - so, back off more often than you think ☺

Don't hog decisions, don't shun them -- stay dynamically balanced on the edge. Let them make mistakes (we learn best from our own mistakes), whenever you can afford to... and once in a while the "mistake" will prove to be brilliance (then be ready to applaud and give credit!!!) To manage well, you need a good enough sense of self-worth that you don't feel the need to assert or prove yourself to confirm your self-worth -- you CAN and WILL let others have their day!



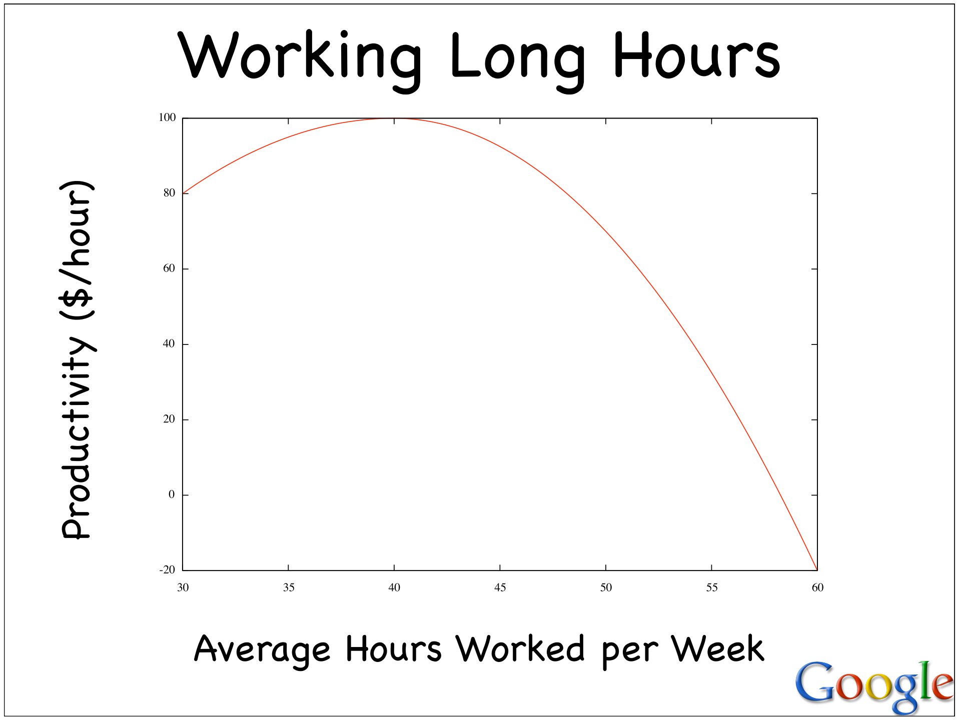
Where does one find time?

- NOT in working incredibly long hours
  - aim for 40 ✌
  - settle for 45 👍
  - 50 is right out 👎

(Note: I mean actual work time, net of [e.g.] blogging, snacking, surfing, chatting...!-)

- nor in extensive telecommuting (face-to-face is the most effective form of communication, and communication is the most crucial part of any manager's job)
- time management works, when done right

There's a warped sense of pride in "working" 60-hours weeks -- but it's a horrid idea (even if you're used to it, you're unlikely to be producing at the top of your potential). Telecommuting's a very sweet dream... but face-to-face still IS best! So, you need a set of techniques, tricks and tips known classically as "time management" (or more modernly as "getting things done").



Working Long Hours

Of course, this is merely indicative (e.g., I know that at my best I'm worth more than $100/hr!-), but I hope it leaves the right impression -- there is SOME number of work hours per week where your productivity is maximal, and if you way overdo it your productivity will go negative -- by working in a far-too-tired state you'll do DAMAGE (negative value). Actually, it's been measured that the productivity per-hour of French workers (lowest hr/year in G8) is higher than in the US (highest), though the total production of _France_ as a whole suffers (Sarkozy promises to make them work more hours/year -- and yet he WON!-)

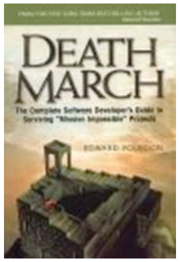
Booch Against Overtime

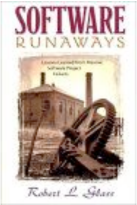
- if the schedule is slipping, do not try to make up time by forcing your developers to work even harder: it will exacerbate the problem. Acknowledge reality: relax schedule constraints, reduce functionality, or both. To do neither is folly.
- you've misplanned/misscheduled if 60+ hours/wk are the norm: overtime as common practice is unsustainable AND an indication of severe mgmt malpractice.

OCCASIONAL weeks at 60, 70, even 80 hours of actual work are inevitable -- emergencies, urgencies, and so on. But *keep them occasional*, for yourself as much as for your reports. Remember: it's a marathon, not a 100-meters dash!

When pace's unsustainable

- productivity loss -- even production loss
- no (time) "reserves" for emergencies
  - if you're running flat out, and an emergency requires a sprint, now what?
  - a wise general keeps a reserve...!
- things get worse as it drags on (and on)

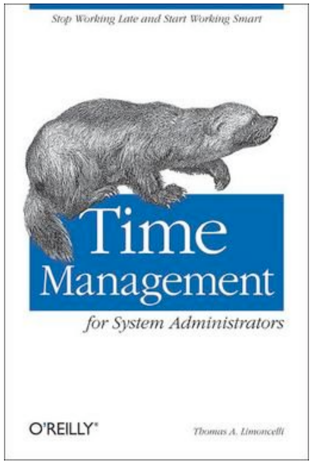




one key issue: are you rewarding RESULTS or EFFORT? "You'll get what you measure"!-)

The "key issue" about performance measurement and rewards is truly key. Again, it's too easy to reward "heroic efforts" rather than good strong planning which saves the needs for such efforts and obtains better RESULTS.

A Better Approach

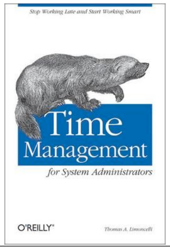

Stop Working Late and Start Working Smart
Time Management
for System Administrators
O'REILLY
Thomas A. Limoncelli

- it's not just for SAs:
  - at least 80/90% is good for developers and managers
  - esp. w/operational duties
- brief, useful summary talk:
- http://video.google.com/videoplay?docid=7278397109952382318
- key ideas: focus & interrupts, single TODO list & its handling, building routines, prioritization

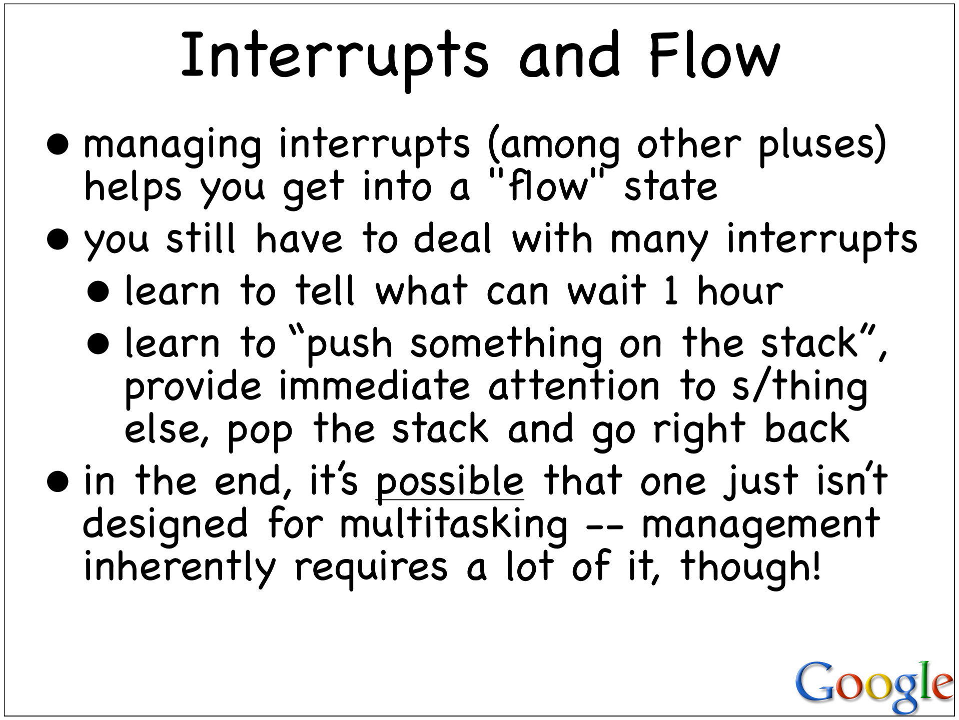
Tom's first-person experience is as a (great!) system administrator, so, quite rightly, that's what he writes about -- but his observations and tips, for the most part, do generaize to all of us.

Limoncelli on Interrupts

- they're inevitable, so we must manage them
  - "interrupt-driven" work's VERY inefficient
- main strategy:"delegate", "record", "do"
  - direct interrupts away from you (to the proper target for them) -- you're NOT all powerful, nor responsible for EVERYthing
  - acknowledge it, write it down, DO LATER
  - if <2 minutes, DO NOW
- "mutual interrupt shield" (and other "flappers":-)


Stop Working Late and Start Working Smart
Time Management
for System Administrators
O'REILLY
Thomas A. Limoncelli

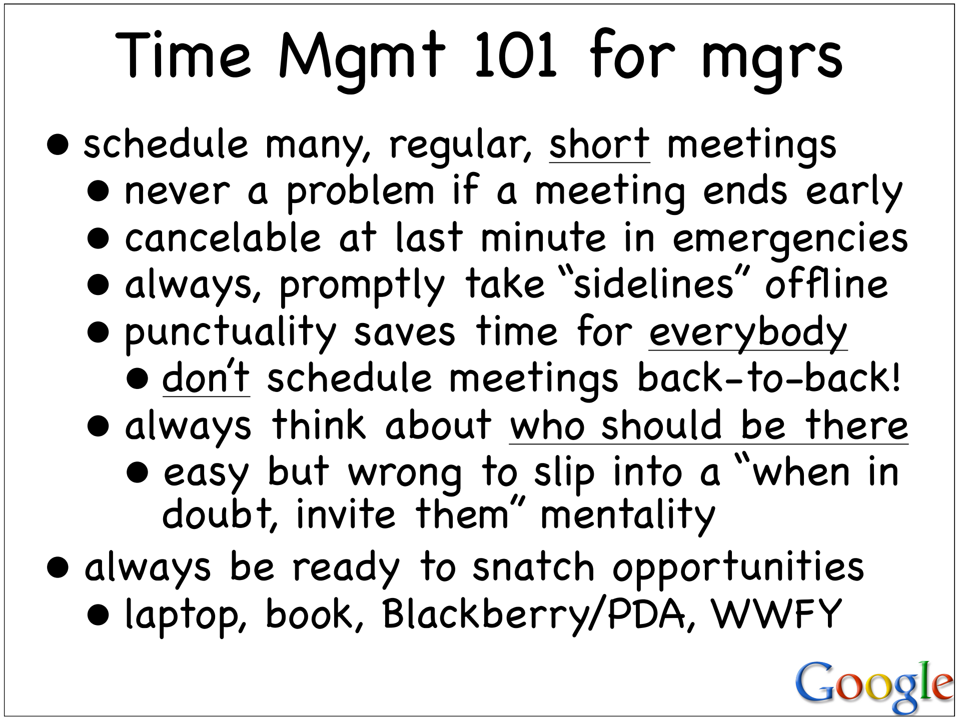
"Deflecting" and otherwise managing interrupts is a technique I single out because it's quite as important for managers as it is for sysadmins. The "mutual interrupt shield" (unless you're high up enough to warrant an administrative assistant, and, even then...) is a particularly useful, although non-obvious, technique for managers.



Interrupts and Flow

- managing interrupts (among other pluses) helps you get into a "flow" state
- you still have to deal with many interrupts
  - learn to tell what can wait 1 hour
  - learn to “push something on the stack”, provide immediate attention to s/thing else, pop the stack and go right back
- in the end, it’s possible that one just isn’t designed for multitasking -- management inherently requires a lot of it, though!

"Getting into the flow" can make your productivity 2 or 3 times better as long as you can maintain it -- and that's one crucial advantage of good interrupts-management (there are many others, mind you -- prioritization and proper sequencing, etc). But, in the conditions of a typical manager, I've observed people who just can't deal properly with the high rate of interrupts, even though from many viewpoints they'd make good managers -- they should change back to the IC track, maybe as "tech leads" if feasible.



Time Mgmt 101 for mgrs

- schedule many, regular, short meetings
  - never a problem if a meeting ends early
  - cancelable at last minute in emergencies
  - always, promptly take “sidelines” offline
  - punctuality saves time for everybody
    - don’t schedule meetings back-to-back!
  - always think about who should be there
    - easy but wrong to slip into a “when in doubt, invite them” mentality
- always be ready to snatch opportunities
  - laptop, book, Blackberry/PDA, WWFY

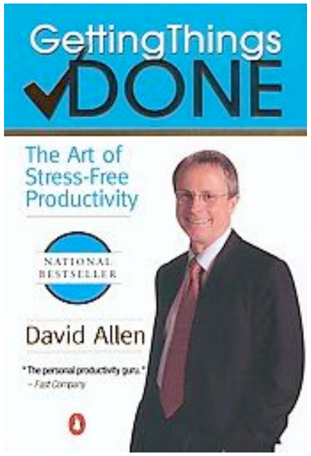
Here are a couple of elementary but crucial tips specifically for managers (the "snatch opportunities" one applies more widely;-).

Time Mgmt 102 for mgrs

- consider each piece of work specifically
  - does it really need to be done at all?
  - if so, am I the best person to do it?
  - &, when should it optimally be done?
- don't let emergencies emerge!
  - a stitch in time saves 9.4247779677
- schedule ~50% of your "discretionary" time each week for not-(yet!-)-urgent "fillers"
  - a wise general strives to keep a reserve
  - can be rescheduled for emerging work
  - don't wait until they are urgent!

These are a bit more advanced, but no less crucial. Care to guess what that weird number IS, btw?-)

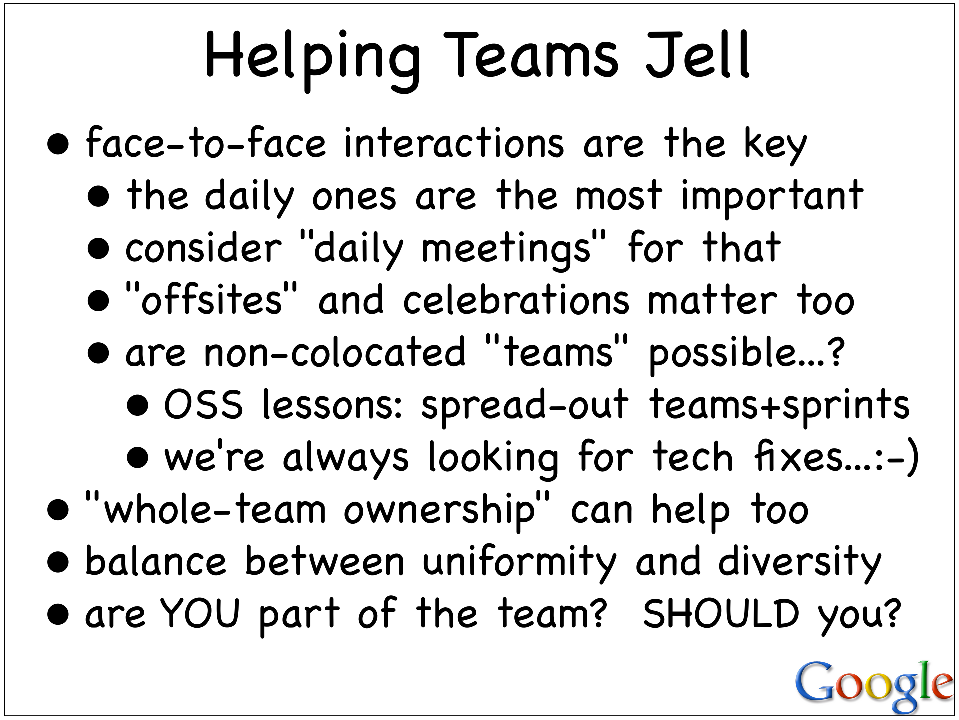
"Manage Actions, not Time"


Getting Things DONE
The Art of Stress-Free Productivity
NATIONAL BESTSELLER
David Allen
"The personal productivity guru." —Fast Company

- highly non-specific (manager oriented, but not hi-tech)
- has many enthusiasts, a real "movement" around it
- http://www.davidco.com/
- key ideas: mind like water, single in-basket, highly structured flow, action steps, "two minutes rule"

doesn't quite work for me, but, works well for many!

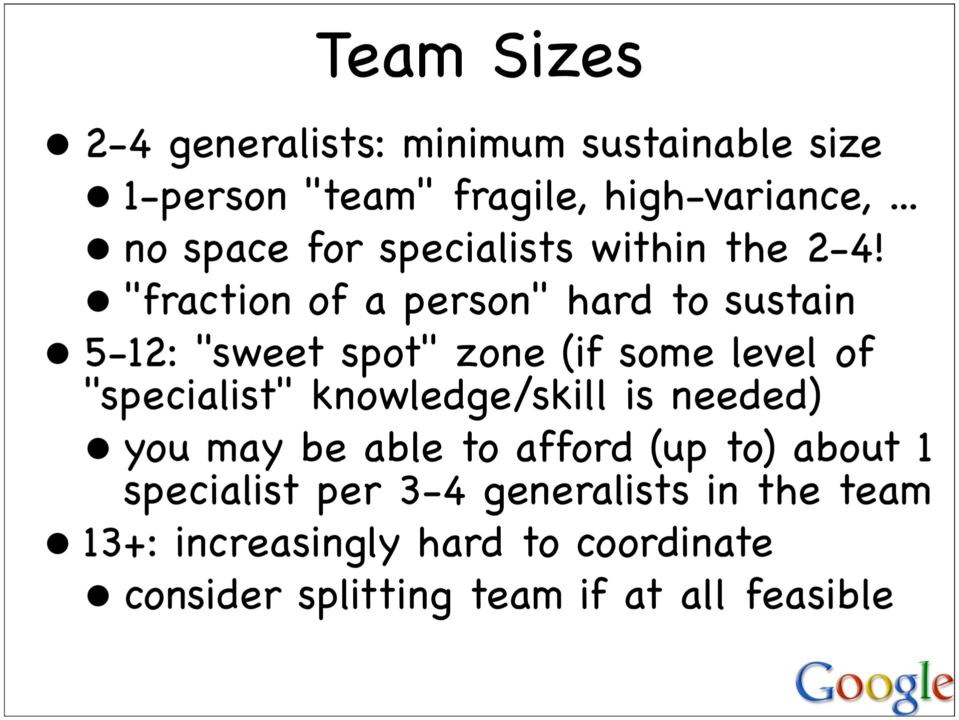
I'm not really into the GTD movement, myself; but I've seen it work great for many, so I feel I must at least mention it!



Helping Teams Jell

- face-to-face interactions are the key
  - the daily ones are the most important
  - consider "daily meetings" for that
  - "offsites" and celebrations matter too
  - are non-colocated "teams" possible...?
    - OSS lessons: spread-out teams+sprints
    - we're always looking for tech fixes...:-)
- "whole-team ownership" can help too
- balance between uniformity and diversity
- are YOU part of the team? SHOULD you?

Teams are what really matters most, and many aspects can help a team "jell" and become MUCH more productive -- chiefly the right kind of face to face interactions, but issues of "ownership" and uniformity vs diversity also play key roles. A crucial question (that I'm still leaving open -- more foreshadowing!-) is whether the manager CAN, and SHOULD, be _part_ of the team s/he manages.



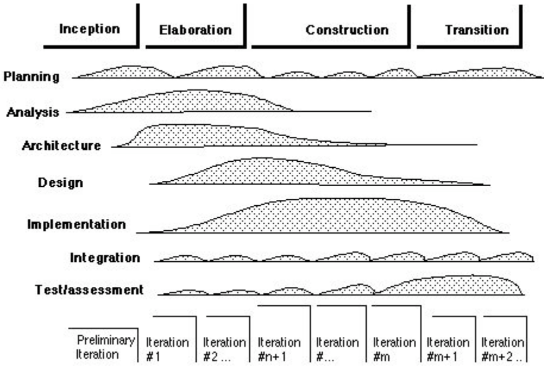
Team Sizes

- 2-4 generalists: minimum sustainable size
  - 1-person "team" fragile, high-variance, ...
  - no space for specialists within the 2-4!
  - "fraction of a person" hard to sustain
- 5-12: "sweet spot" zone (if some level of "specialist" knowledge/skill is needed)
  - you may be able to afford (up to) about 1 specialist per 3-4 generalists in the team
- 13+: increasingly hard to coordinate
  - consider splitting team if at all feasible

I've seen alleged "teams" of 40 or more people -- but "alleged" is key here;-). More often these days you see single people assigned as the whole "team" for small sub-projects, but that's far from optimal either. If I could have my pick, I'd have 4 generalists when I can do w/o specialists, or about 2 specialists and 7 generalists when the specialists are crucial to a given project (and use already-mentioned techniques, such as PP, to "grow" the specialists AND the generalists at the same time). Splitting one person across multiple teams has its own issues (though sometimes it's the least of evils, typically for a specialist and many smallish teams).

Vary team over time?

- a semiclassic "optimized" approach (in RUP)



- however... what does that do to the TEAM?

RUP is Rational's Unified Process. Systematically changing staffing levels and composition over a projects' macro-terms lifecycle seems an obvious approach... but you SHOULD consider what it WILL to do team's cohesion and "jelling". I'm NOT advocating "stationary" teams -- projects SHOULD have a beginning and an end, and so (on a longer timescale) generally should team's lifecycles -- but DO consider the people (and their ineffable dynamics as a jelled team!), NOT just the single task!!!

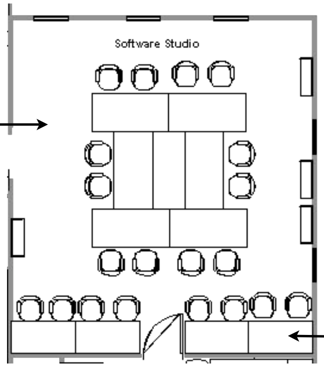
How many can YOU manage?

- ...and manage WELL...?
  - varies by oversight needs, location, ...
- and, what else do you do besides that?-)
- direct reports: 6, a breeze; 12, good; 24, stretching it a bit; 48 "is right out"
  - +: count indirect reports as a fraction

In one team or several, there IS a limit to the number of people you can manage effectively, depending on how much hand-holding (or shielding, oversight, etc) your people need, whether you have other things to do besides people-managing, etc. If you're a second- or higher-level manager, don't forget to count SOME fraction of your indirect reports -- they WILL at least occasionally need your direct attention and involvement (you DON'T want to ossify the organization in order to avoid that, believe me!)

A Team's "Geometry"

- e.g, "caves and common" arrangement:



"Commons" (team) work area

Private Workstations ("Caves")

There are many, MANY other possibilities -- alas, I'm not an architect, but I want to point out that this issue needs care too.

The HyperProductive Dev'l

- estimated to be 4-10 times as productive as "normal" good developers (some sources quote estimates up to 30)
- these apply to only SOME phases of the development lifecycle (typically: low-level design, coding, debugging, optimizing)
- how do you fit him/her best into the team?
  - can s/he help teammates grow?
  - or, will they just slow him/her down?
  - does s/he have, or want to grow, any leadership qualities?

A nice problem to have, in a sense;-). But, problem (or, "opportunity":-) it sure is. How do you let the HPD fully express his or her incredible productivity while NOT damaging the team and other members thereof, and indeed HELPING them grow...? It's a subtle and complex question, depending more on "human" factors than technical ones. Good solutions, depending on those factor, may range from making the HPD an exception to the rule of "no 1-person teams", to deploying him as a roving consultant across many teams for his/her areas of strength -- but often integration with one team will also be possible.

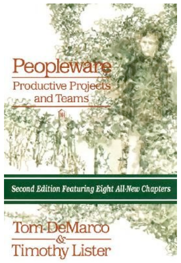
Development Tools

- what needs to be standardized?
  - language, lib, style, standards for testing, release engineering, communication
  - source-code control, issue-tracking, build scripts (ideally: continuous integration), testing (AUTOMATED, at ALL levels!)
- many other tools need not be uniform
  - editors/debuggers/IDEs, OS to be used for development, mail/&c clients, ...
  - let a thousand flowers bloom (wherever that is sensibly feasible!-)

People want to "mark their space" -- let each developer happily use their favorite editor, debugger, IDE, etc. But you must stand firm on other issues: the key SHARED tools are versioning system, issue-tracking system, continuous integration and automated testing system (the 3 had better be well harmonized and integrated, too:-). Also, there needs to be consensus on programming language (and style, libraries, frameworks, ...), and standards for testing levels, release/deployment activities, docs and other forms of communication -- all indispensable issues to afford "whole team ownership", an absolutely crucial practice.

Can you be IN the team?

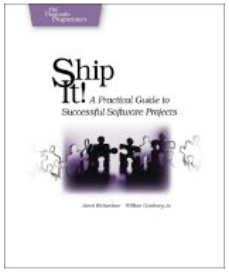
"Once you have four or more people in your group, you can't perform technical work and still be a great manager." (Wk 6)


Peopleware
Productive Projects and Teams
Second Edition Featuring Eight All-New Chapters
Tom DeMarco
&
Timothy Lister

"Managers are not usually part of the teams that they manage ... leadership just doesn't have much place here." (Ch 23)

Two excellent books (which I both heartily recommend) that strongly endorse a traditional idea: the thesis that managers can never do technical work, be part of their teams, exercise real "leadership"...

Maybe you can, & should...



"Tech leads split their time between development tasks and management tasks, not working exclusively in either realm." (T.15: Let a tech lead)

ONE possible way to manage software development is to "get your hands dirty" with it (definitely not the ONLY one...!-)

...and another excellent book arguing otherwise. So maybe, while not the ONLY way, ONE way to manage knowledge workers IS to be (part-time) one of them...?

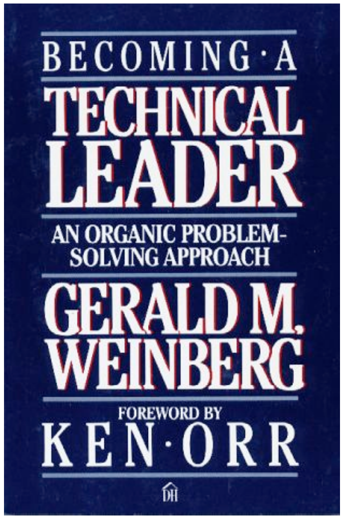
Shd you be a coxswain...?

- "with" rowers (coxswain doesn't row, just steers & directs!-) are sometimes faster...
- ...but not ALWAYS, sometimes coxless's best (key issue: how many rowers?)

Rowing lets us argue both ways -- for large-enough teams of rowers, a non-rowing coxswain appears to be a plus; but for smaller boats, the "without" times appear better;-)

At times you MUST pitch in

- "When time and labor are running short, stop working on ['big' things] and just pitch in [...] some would-be leaders have such an inflated image of themselves that they cannot stoop to mere implementation" (!)
- ...but, why wait for the "running short" stage?-)



Weinberg is talking about generating ideas vs executing on them -- but I think this generalizes well... except that you shouldn't wait for emergencies where "time and labor are running short" before you DO pitch in -- that's very much the wrong approach.

One way to manage SW

- say that manager M is a technical peer of the developers (design, code, debugging...)
- M can nurture mutual trust, interaction and respect by and for the developers by deploying him/herself as a "wildcard technical resource"
  - not for the "fun" tasks, but, rather,
  - for urgent ones requiring an extra pair of ~~hands~~ brain hemispheres right now,
  - be they fun or (preferably!-) chores
- many objections should come to mind here...

So here's my core thesis, which I've been pushing for years -- one approach to let a technical manager "get their hands dirty" in SOME of the team's professional work.

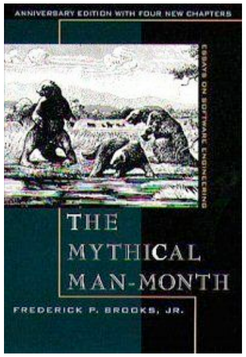
Wait, but, what about...

If you're following critically, you should have one or more of the following objections...:

1. what about Brooks' Law?
2. shouldn't a manager always delegate?
3. must be neglecting "real" mgmt work!
4. it's a waste of technical talent.
5. ...supply your own objections...:-)

Of course, many objections usually get raised here (I'm selecting a subset compared to what I do when I teach about this very specific idea exclusively, rather than about management in general as I do here:-).

Brooks' Law


ANNIVERSARY EDITION WITH FOUR NEW CHAPTERS
ESSAYS ON SOFTWARE ENGINEERING
THE MYTHICAL MAN-MONTH
FREDERICK P. BROOKS, JR.

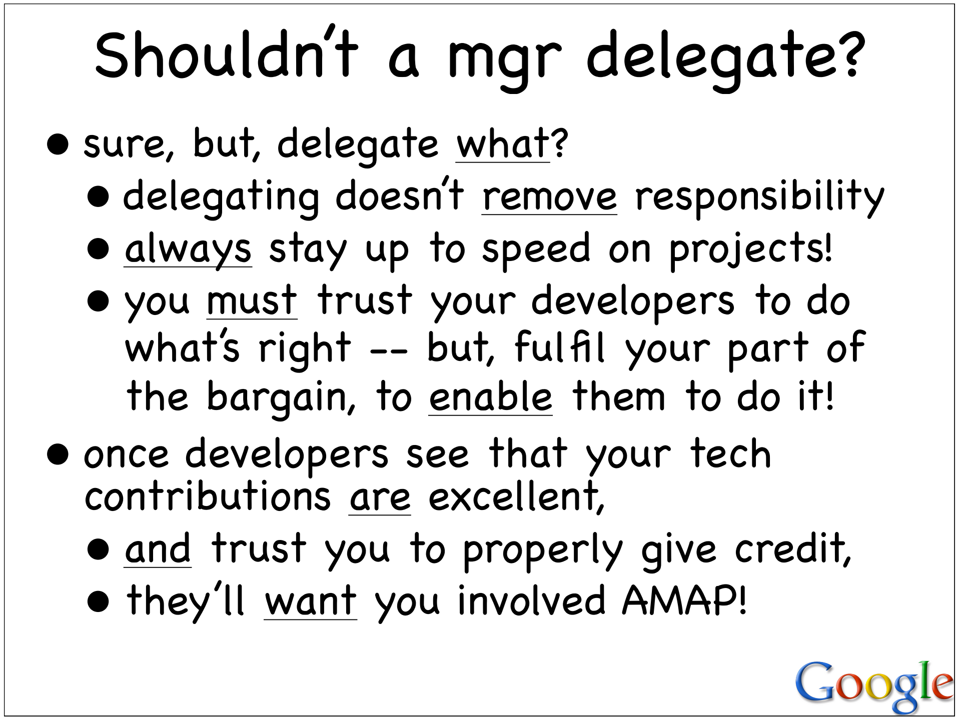
"Adding programmers to a late software project makes it later"

- Yes, but: everybody always omits the immediately-preceding qualification: "Oversimplifying outrageously, we state"...!-)
- also: just don't <u>let</u> it become late!-)

Based on extra time for extant programmers to bring new ones up to speed + extra communication overhead

If a manager is always up-to-speed, & always communicating: no extra overhead ☛ no Brooks' Law

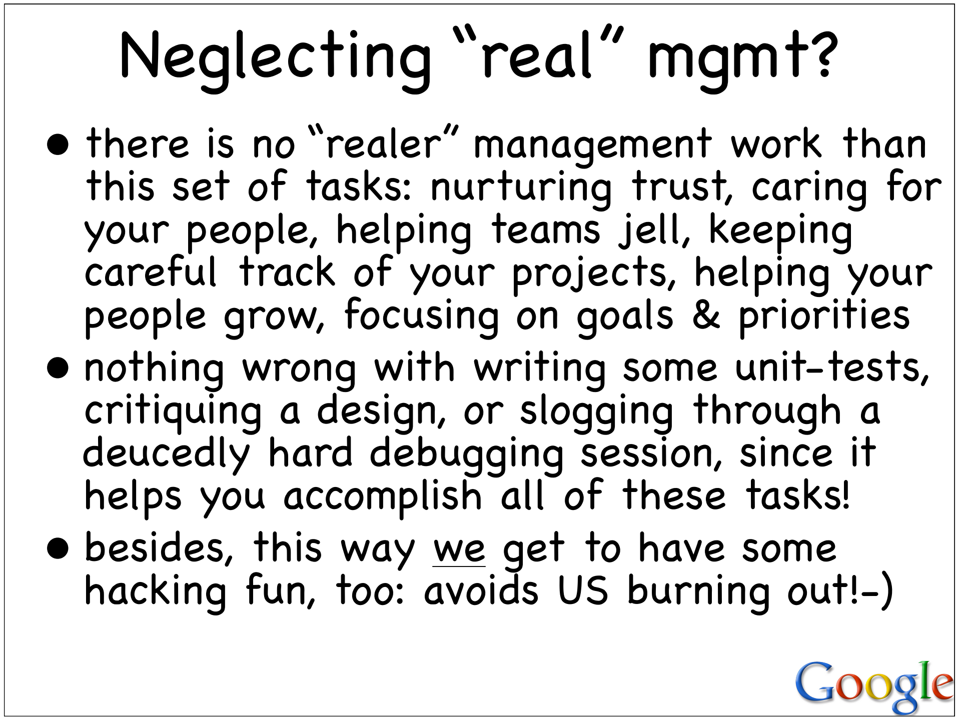
I LOVE Brooks -- but his Law is often misquoted, misused, and misapplied.



Shouldn’t a mgr delegate?

- sure, but, delegate what?
  - delegating doesn’t remove responsibility
  - always stay up to speed on projects!
  - you must trust your developers to do what’s right -- but, fulfil your part of the bargain, to enable them to do it!
- once developers see that your tech contributions are excellent,
  - and trust you to properly give credit,
  - they’ll want you involved AMAP!

Delegation is not abdication, as I've already covered in this tutorial.



Neglecting “real” mgmt?

- there is no “realer” management work than this set of tasks: nurturing trust, caring for your people, helping teams jell, keeping careful track of your projects, helping your people grow, focusing on goals & priorities
- nothing wrong with writing some unit-tests, critiquing a design, or slogging through a deucedly hard debugging session, since it helps you accomplish all of these tasks!
- besides, this way we get to have some hacking fun, too: avoids US burning out!-)

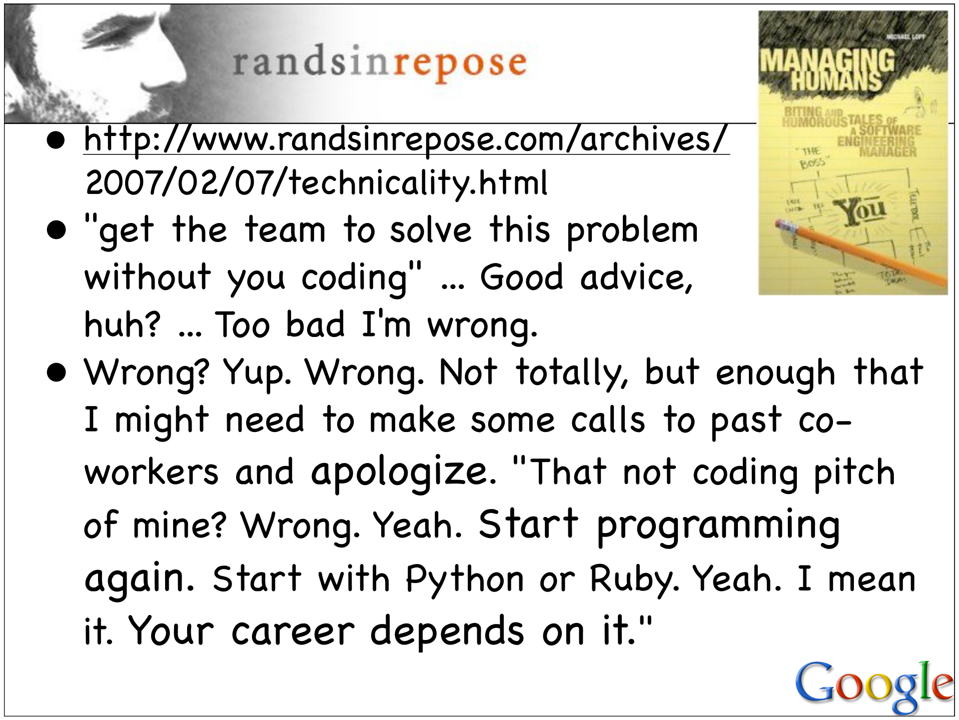
"Real management" is anything and everything real (successful) managers do, that helps their projects!-) Balance is needed (as it always is "out here on the edge":-) -- don't hog the fun tasks, don't override your tech leads and make their jobs irrelevant, don't take credit for technical accomplishments, etc, etc -- but part of the balance is getting the challenge and fun to deal directly with hard technical tasks, at least once in a while!-)

Waste of tech talent?

- it's not wasting, but leveraging it!
- there ARE places where management is only for those who have nothing more to contribute technically... but not SUCCESSFUL ones!-)
- "but isn't leverage high only in design"?
  - no way!
  - "the devil is in the details"
  - and where's a devil to be fought, that's where the best exorcists are needed!-)

75

That's kind of the flip objection to the last one -- that great techies should never move on to management to avoid "wasting" their tech talent (or, in a more attenuated form: that they should strictly focus on "upstream" activities, not coding, debugging, docs, deployment, testing, ...). Successful organizations in the hi-tech field MUST have good ways to balance leadership and technical contributions (dual ladders AND ways for people to "straddle" both ladders, hop back and forth between them, etc:-).


randsinrepose
MICHAEL LOPP
MANAGING HUMANS
BITING AND HUMOROUS TALES OF A SOFTWARE ENGINEERING MANAGER
THE BOSS
YOU

- http://www.randsinrepose.com/archives/2007/02/07/technicality.html
- "get the team to solve this problem without you coding" ... Good advice, huh? ... Too bad I'm wrong.
- Wrong? Yup. Wrong. Not totally, but enough that I might need to make some calls to past co-workers and apologize. "That not coding pitch of mine? Wrong. Yeah. Start programming again. Start with Python or Ruby. Yeah. I mean it. Your career depends on it."

76

I was particularly happy, a good while after I'd started my solo crusade to allow tech mgrs SOME hands-on tech contributions, to see one of my favorite bloggers on technical management issues pick up on exactly the same theme. Lopp's book is good, too (if you can stand books taken/inspired from blogs -- I'm OK w/them). And his ability to state he was wrong, and apologize, comes VERY close to PROVING he's a great manager (and all-around human being), not just a great writer about it!-)

Ars longa, vita brevis...

more books than one can make time for!-)

The Limits of Software
People, Projects, and Perspectives
ROBERT N. BRITCHER
DREAMING IN CODE
SCOTT ROSENBERG
Project Management
O'REILLY
Scott Berkun
WORKING EFFECTIVELY WITH LEGACY CODE
Michael C. Feathers
MINTZBERG ON MANAGEMENT
INSIDE OUR STRANGE WORLD OF ORGANIZATIONS
HENRY MINTZBERG
Fit for Developing Software
Framework for Integrated Tests
Rick Mugridge
Ward Cunningham
Google

Mintzberg (many books) is at the same time a Solon of management teaching, and a trenchant critic of current pratice in the field. Berkun has many good tips and some interesting deep reflections on project management. FIT (and Fitnesse) are great ways to develop test-driven SW (at acceptance-test level, roughly). Feathers is great if you need to deal with untested, badly documented legacy SW (don't we all!). Britcher is pessimistic but worth reading (and critiquing). "Dreaming in Code" is the best case study yet on "why do bad things happen to good project" -- Rosenberg has no prejudged idea, gives you lots of info and lets YOU judge.

Q?

A!

I hope the whole tutorial was VERY interactive, but, this is the right place to ask any questions you omitted to ask DURING it (I'll also be around through OSCON and available for more discussion -- I LOVE these subjects!-).

Thanks for attending this tutorial – and for all that, I feel sure, YOU taught ME during Q&A and other interactions!!! Alex