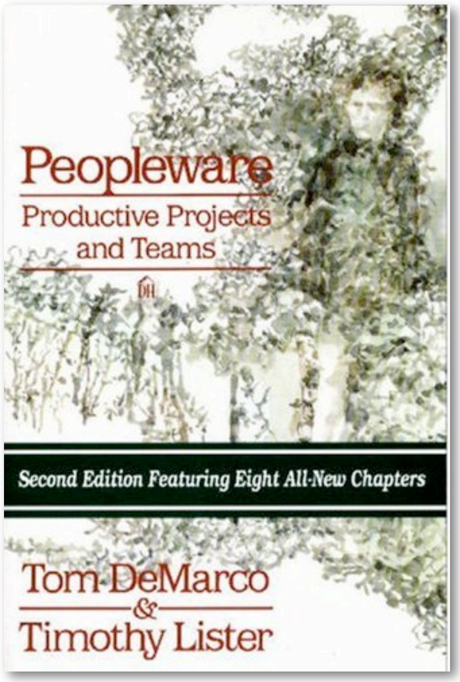
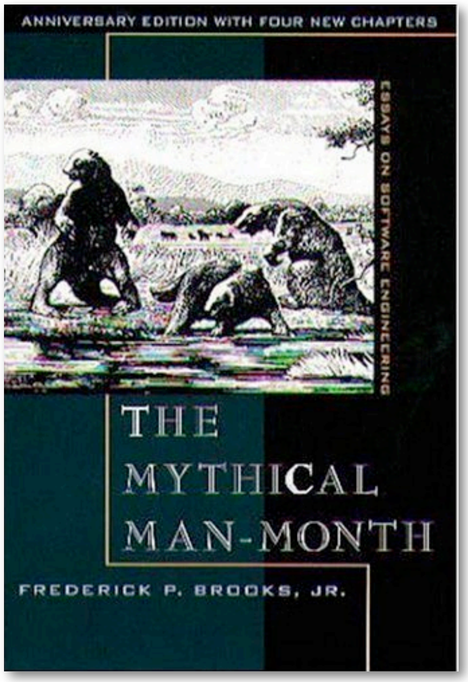
AGILE (and whatnot)

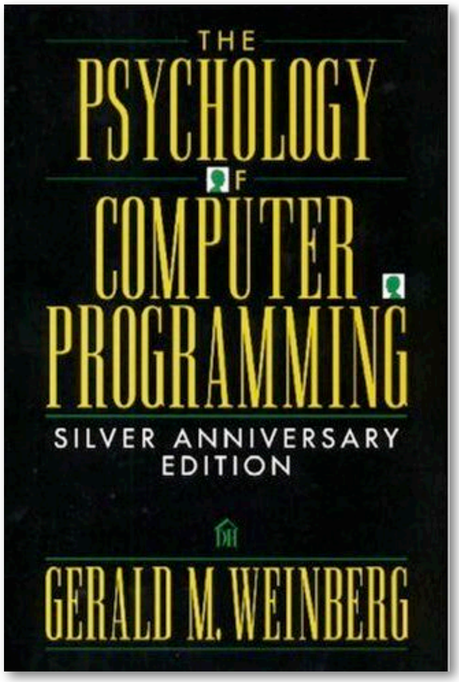
Tobias Wrigstad
Thorbiörn Fritzon
Beatrice Åkerblom
Henrik Bergström
Johan Östlund

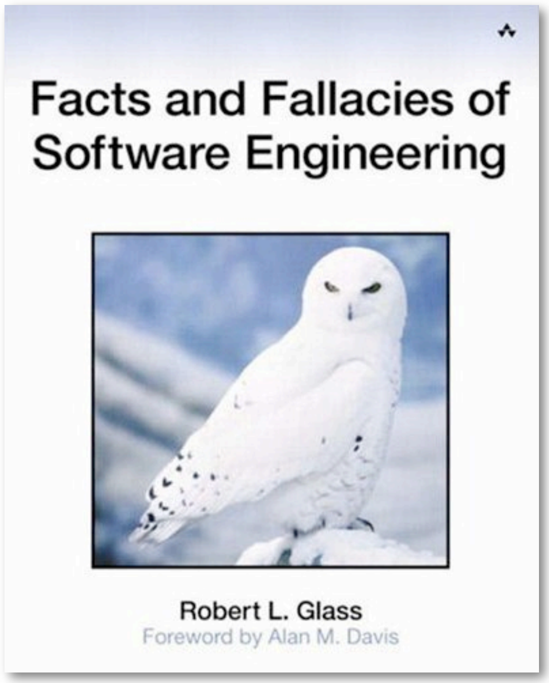
My Main Inspiration for This Lecture

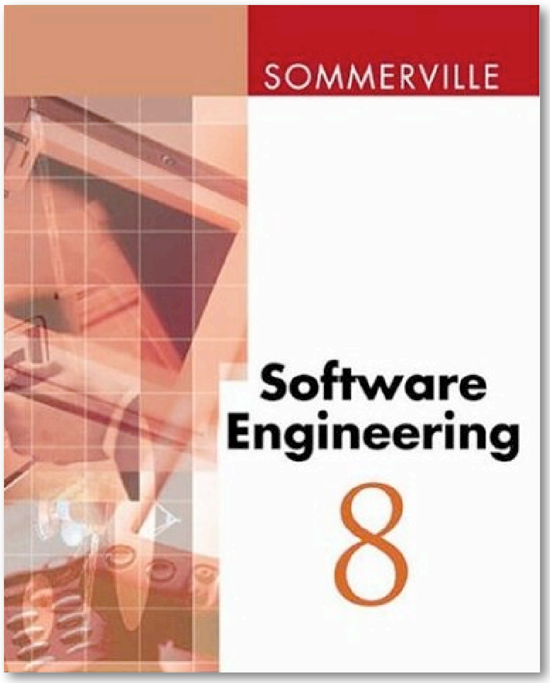
My Main Inspiration for This Lecture


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


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


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


Facts and Fallacies of Software Engineering
Robert L. Glass
Foreword by Alan M. Davis



+ many articles (see back)

Human Factors in Software Development

Tobias Wrigstad
tobias@dsv.su.se

Human Factors in Software Development

Tobias Wrigstad
tobias@dsv.su.se

Computer science
is the only profession
in which a single mind
is obliged to span the
distance from a bit to
a few hundred
megabytes

—Steve McDonnel

Why care about the human factor?

All programmers are not equal

What makes a good programmer?

Is software engineering about making all programmers equal?

“80% of software work is intellectual. A fair amount of it is creative. A little of it s clerical.”

Robert Glass [1]

People are Not Machines

- People's performance will differ over time
  - in relation to others
  - and their own performance
- Code aesthetics matter
- Personal chemistry matters
- Management and motivation matters
- Working environments matter
- Cultural factors matter

The Coding War Game

- Survey by Tom DeMarco and Tim Lister
  - Programmers from different companies compete
  - Everyone works alone, but is judged in pair with another programmer from the same company
  - Work is conducted during regular office hours, in the programmers' offices

War Game Results

- Language choice—matters less than who is at the machine (exception: assembly language)
- Years of experience matters not (exception: <6 months exp. with language of choice)
- Correlation between zero defects and finishing early (about 33% had zero defects)
- No correlation between salary and result in the war game
- *Correlation between partner and performance*
  - Suggests organisation matters
  - Or good programmers cluster
  - Or both
- Productivity ratio of 1:10 between organisations

Quality, Schedules & Humans

Sad Facts of Life [14]

- ❖ Managers tend to view quality as just another product attribute
- ❖ Workers view quality quite differently
  - ~ Quality is tied to self-esteem
- ❖ "Workers tend to have a lowest level of acceptable quality that is far above what the market is willing to pay for" [14]
  - ~ Generally equal to highest quality achieved in the past

"Quality, far beyond that expected by the customer, is a means to higher productivity"

“Quality is Free”

—Philip Crosby, 1979

Parkinson's Law

- Not built on any tangible evidence
  - "Proof by repeated assertion" [14]
- Key factor behind unrealistic planning
  - Faulty belief that time shortage counters Parkinson's Law

“Organisational busy work tends to fill the working day”

—TDM & TLR [14]

Metrics about Metrics

Productivity by Estimation

- Studies of 103 projects by Lawrence and Jeffrey in 1985
  - Investigating the truth behind the folklore belief that "*programmers are more productive when they work towards their own estimates*"

Metrics about Metrics

Productivity by Estimation

Prep'd by	*Avg. Prod.*	*#Projects*

- Studies of 103 projects by Lawrence and Jeffrey in 1985
  - Investigating the truth behind the folklore belief that "*programmers are more productive when they work towards their own estimates*"

Metrics about Metrics

Productivity by Estimation

Prep'd by	*Avg. Prod.*	*#Projects*
Programmer	8.0	19

- Studies of 103 projects by Lawrence and Jeffrey in 1985
  - Investigating the truth behind the folklore belief that "*programmers are more productive when they work towards their own estimates*"

Metrics about Metrics

Productivity by Estimation

Prep'd by	*Avg. Prod.*	*#Projects*
Programmer	8.0	19
Supervisor	6.6	23

- Studies of 103 projects by Lawrence and Jeffrey in 1985
  - Investigating the truth behind the folklore belief that "*programmers are more productive when they work towards their own estimates*"

Metrics about Metrics

Productivity by Estimation

Prep'd by	*Avg. Prod.*	*#Projects*
Programmer	8.0	19
Supervisor	6.6	23
Both	7.8	16

- Studies of 103 projects by Lawrence and Jeffrey in 1985
  - Investigating the truth behind the folklore belief that "*programmers are more productive when they work towards their own estimates*"

Metrics about Metrics

Productivity by Estimation

Prep'd by	*Avg. Prod.*	*#Projects*
Programmer	8.0	19
Supervisor	6.6	23
Both	7.8	16
Systems analyst	9.5	21

- Studies of 103 projects by Lawrence and Jeffrey in 1985
  - Investigating the truth behind the folklore belief that "*programmers are more productive when they work towards their own estimates*"

Metrics about Metrics

Productivity by Estimation

Prep'd by	*Avg. Prod.*	*#Projects*
Programmer	8.0	19
Supervisor	6.6	23
Both	7.8	16
Systems analyst	9.5	21
No estimate	12.0	24

- Studies of 103 projects by Lawrence and Jeffrey in 1985
  - Investigating the truth behind the folklore belief that "*programmers are more productive when they work towards their own estimates*"

Intermission

Code Reading

Readable Code

- Most programs are modified more than once during their life-time
- Generally, a line of code is read many more times that it is written/updated
- This suggest writing readable code is important
  - Better quality
  - Easier to maintain and update
  - Studies suggest writing clear code makes your code better
- Is there anything more to it than just coding conventions?

Motivating Readability: Inspections

- Normally carried out as careful, line-by-line review of the program source
- More than 60% of all errors in a program can be detected using informal program inspections [20]
- Static code reviewing is more effective and less expensive than defect testing in discovering program faults [21,22]
- Studies of telecom software [23] show that about 90-125 LOC per hour can be inspected
- Readable code have fewer faults and can be inspected faster

Testing Your Own Code

- "When a man that's just bought a Ford is reading car advertisements, he will spend most time reading about Fords"
- The human eye has an almost infinite capacity for *not seeing* what it does not want to see
- Individual ownership or group ownership?
  - ~ Litmus test: "code you've written" or "your code"?
- The pair programming solution

Coding Conventions

- Joel Spolsky [2] about working on the Microsoft Excel team
  - About 50 developers and 50 testers
  - Never used coding standards and didn’t have any problems
  - It was possible to see who had written a piece of code by looking at the style
- What about your coding conventions?

A Program to be Read [3]

```
XXX: PROCEDURE OPTIONS(MAIN);
     DECLARE B(1000) FIXED(7,2),
             C FIXED(11,2),
             (I, J) FIXED BINARY;
     C=0;
     DO I = 1 TO 10;
         GET LIST((B(J) DO J TO
                                 1000));
         DO J = 1 TO 1000;
             C = C + B(J);
             END;
         END;
     PUT LIST(‘SUM IS ‘, C);
     END XXX;
```

A Program to be Read

```
XXX: PROCEDURE OPTIONS(MAIN);
     DECLARE A(10000) FIXED(7,2),
             C FIXED(11,2),
             J FIXED BINARY;
     C=0;
     GET LIST((A(J) DO J = 1 TO
                           10000));
     DO J = 1 TO 10000;
         C = C + B(J);
         END;
     PUT LIST(‘SUM IS ‘, C);
     END XXX;
```

A Program to be Read

```
XXX: PROCEDURE;
     DECLARE A(10000) FIXED(7,2),
             J FIXED BINARY;
     GET LIST((A(J) DO J = 1 TO
                            10000));
     PUT LIST(‘SUM IS ‘, SUM(A));
     END XXX;
```

A Program to be Read

```
XXX: PROCEDURE;
     DECLARE A(10000) FIXED(7,2),
     GET LIST(A);
     PUT LIST(‘SUM IS ‘, SUM(A));
     END XXX;
```

Remarks

- Will a newly employed programmer realise the reasons for writing this program in the first way
- How will she view the original programmers?
- This might lead to an unhealthy attitude towards the senior programmers writing that code
  - They are stupid
  - I'm a better programmer

Reading Guidelines [4]

Reading Guidelines [4]

1. Work in pairs, thinking out loud to one another.

Reading Guidelines [4]

1. Work in pairs, thinking out loud to one another.

2. Argue. If your partner says “this means *X*”, and you either don't understand why or you have another opinion, demand an explanation.

Reading Guidelines [4]

1. Work in pairs, thinking out loud to one another.

2. Argue. If your partner says “this means *X*”, and you either don't understand why or you have another opinion, demand an explanation.

3. Sometimes, when dealing with a chunk of text, it's easier to figure out the middle *after* you understand what's on both ends. Therefore, if a fragment of text has you stumped, try skipping over it and seeing if you can come back to it later. (But you still have to come back to it eventually.)

Reading Guidelines [4]

1. Work in pairs, thinking out loud to one another.

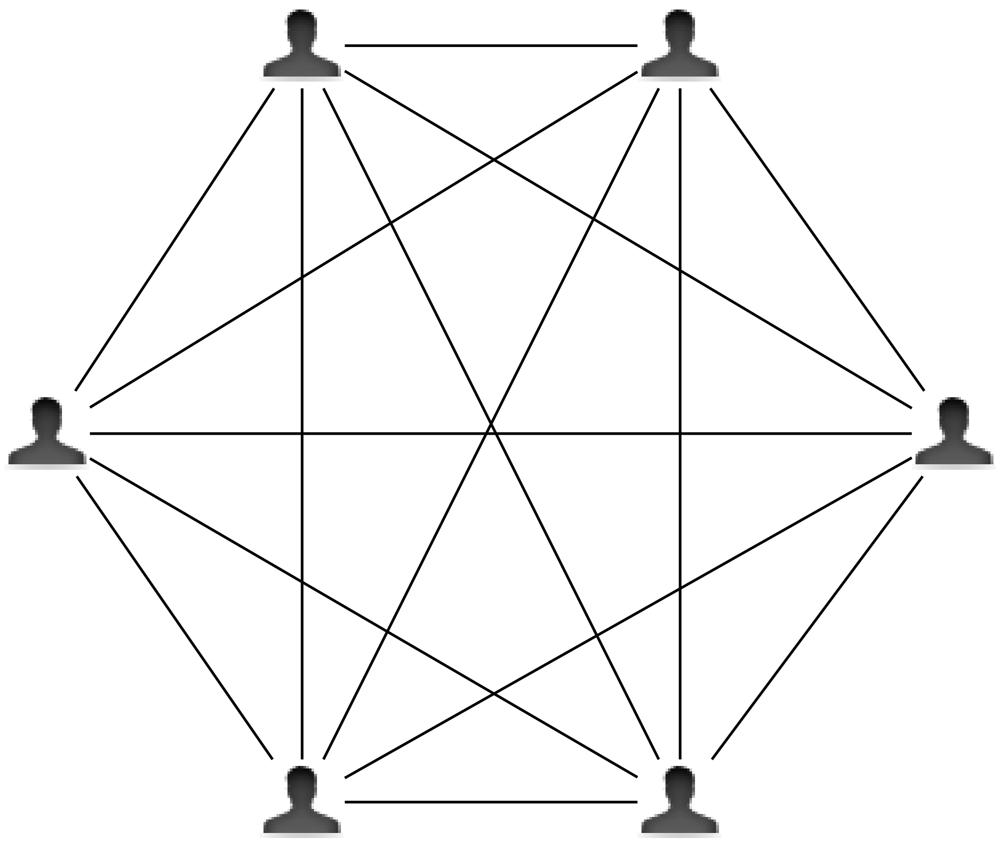
2. Argue. If your partner says “this means *X*”, and you either don't understand why or you have another opinion, demand an explanation.

3. Sometimes, when dealing with a chunk of text, it's easier to figure out the middle *after* you understand what's on both ends. Therefore, if a fragment of text has you stumped, try skipping over it and seeing if you can come back to it later. (But you still have to come back to it eventually.)

4. Read the text both “inside” and “outside”. An inside reading translates the text into English (or whatever your native language is) phrase-by-phrase; an outside reading translates a larger chunk into an idiomatic paragraph. If you only read inside, you can miss the forest for the trees; if you only read outside, you can fool yourself by making broad guesses and not verifying them with details.

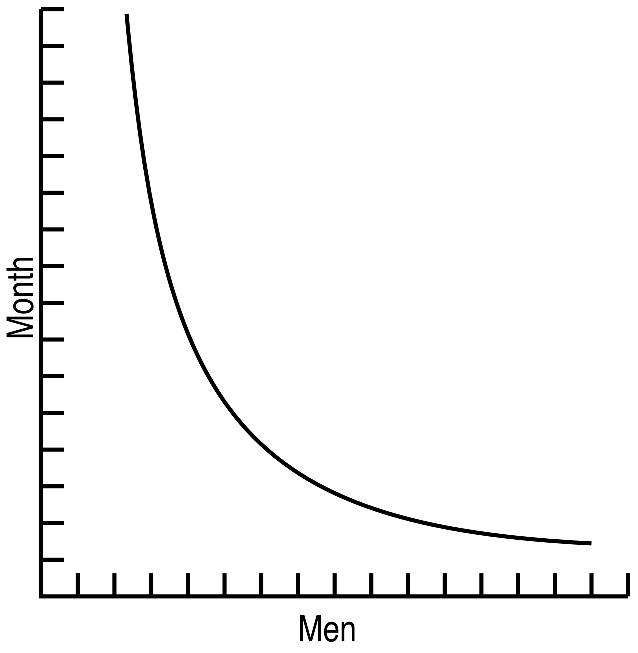
Human Factors in Large-Scale Programming

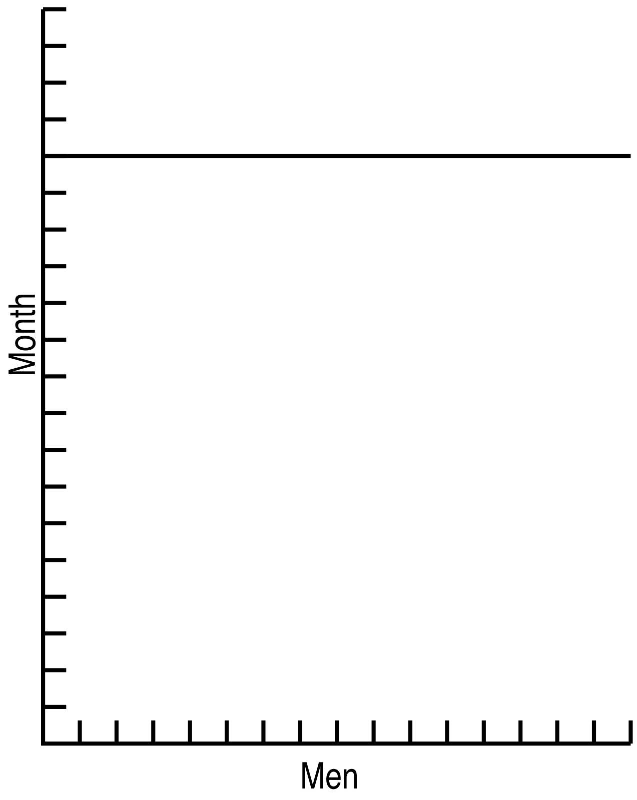
Brooks' Law and More

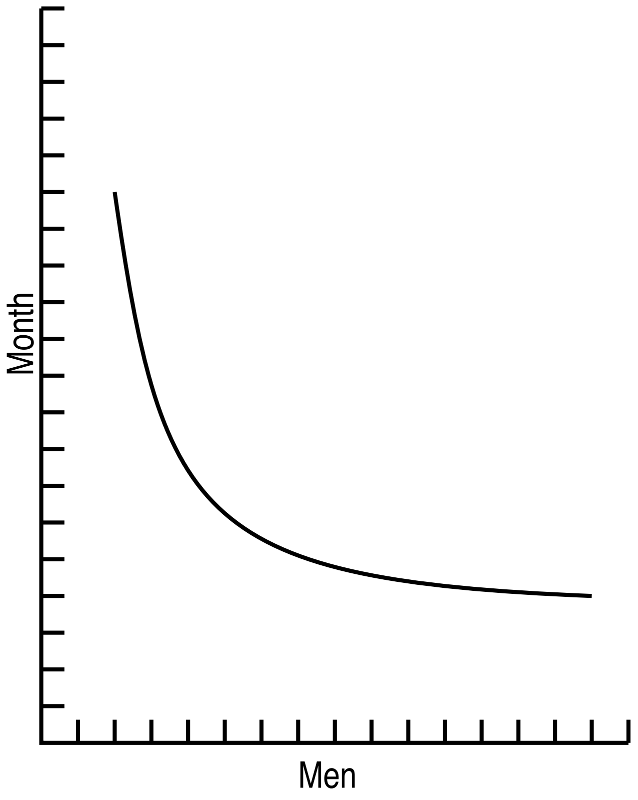


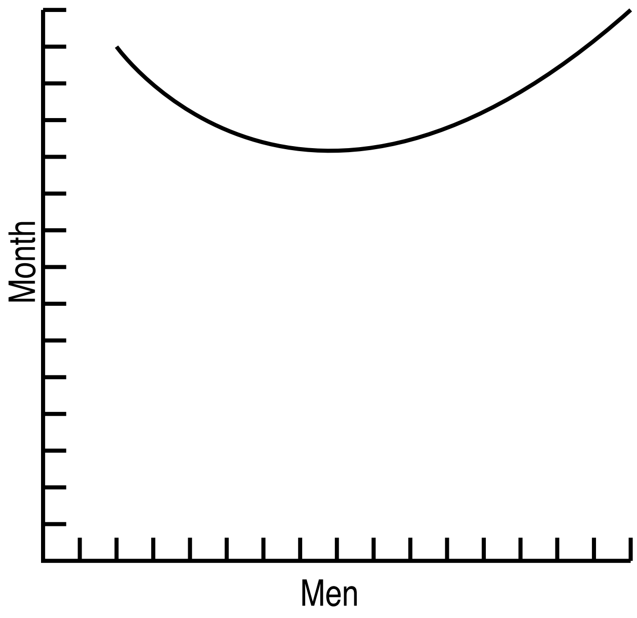
- This team cannot possibly do in 1 month what 1 person can do in 6 months
- Minimal stretch in calendar time
- People don't account for group forming

Mythical Man-Month

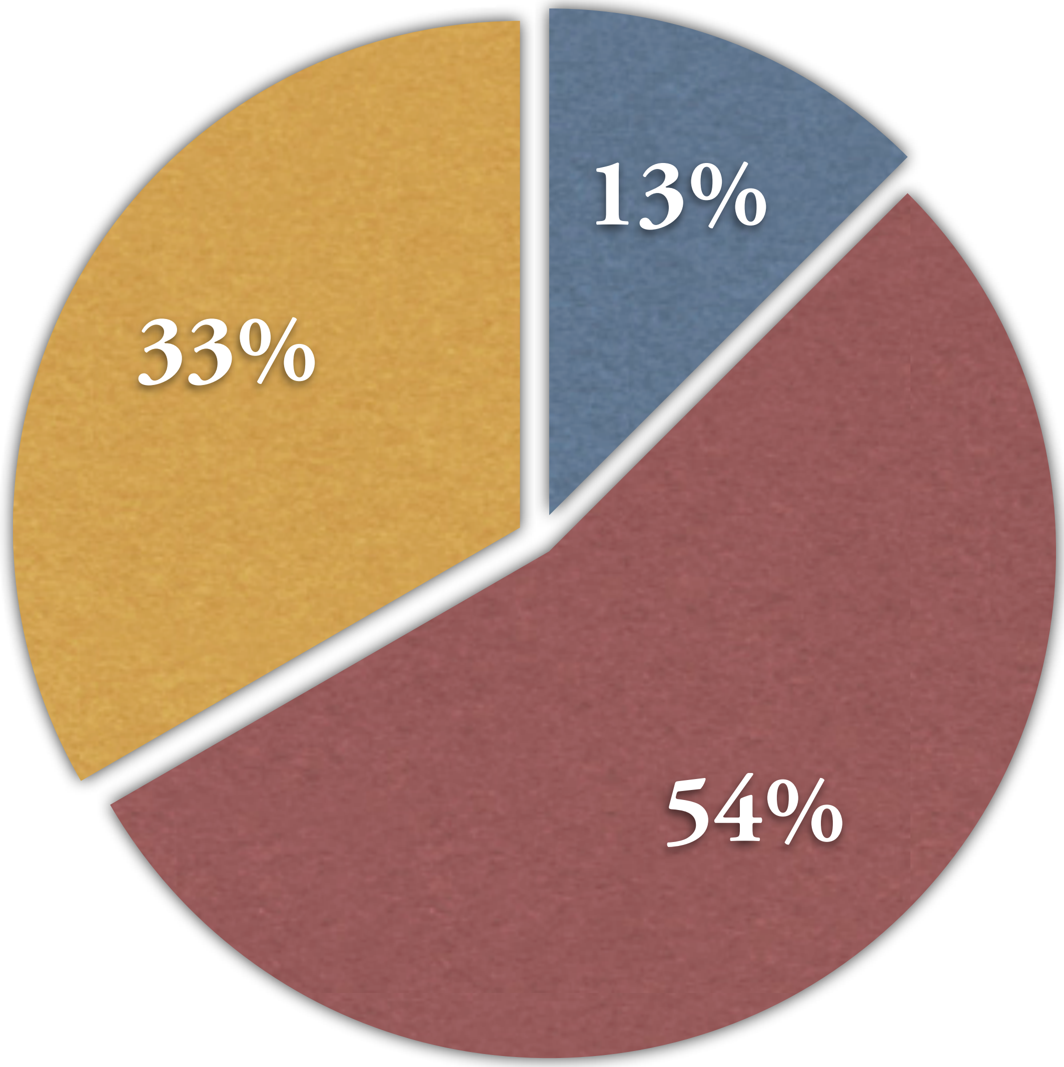








What Causes Most Problems in Projects?



● Technology
● Management
● Human Factors

Survey about Human Factors in Projects

Survey about Human Factors in Projects

- People do not assume responsibility

Survey about Human Factors in Projects

- People do not assume responsibility
- People are not following/do not understand design rules/processes

Survey about Human Factors in Projects

- People do not assume responsibility
- People are not following/do not understand design rules/processes
- People are not assigned so that their competence is utilised in a good/the best way

Survey about Human Factors in Projects

- People do not assume responsibility
- People are not following/do not understand design rules/processes
- People are not assigned so that their competence is utilised in a good/the best way
- Lack of overview of the process

Survey about Human Factors in Projects

- People do not assume responsibility
- People are not following/do not understand design rules/processes
- People are not assigned so that their competence is utilised in a good/the best way
- Lack of overview of the process
- Formal/Informal communication does not work well, especially not between teams

Survey about Human Factors in Projects

- People do not assume responsibility
- People are not following/do not understand design rules/processes
- People are not assigned so that their competence is utilised in a good/the best way
- Lack of overview of the process
- Formal/Informal communication does not work well, especially not between teams
- People do not have the required competence

Survey about Human Factors in Projects

- People do not assume responsibility
- People are not following/do not understand design rules/processes
- People are not assigned so that their competence is utilised in a good/the best way
- Lack of overview of the process
- Formal/Informal communication does not work well, especially not between teams
- People do not have the required competence
- People do not know how to prioritise

Survey about Human Factors in Projects

- People do not assume responsibility
- People are not following/do not understand design rules/processes
- People are not assigned so that their competence is utilised in a good/the best way
- Lack of overview of the process
- Formal/Informal communication does not work well, especially not between teams
- People do not have the required competence
- People do not know how to prioritise
- People make mistakes trying to understand requirements

Motivating People

Project Management

- *"The people working in a software organisation are its greatest assets"* [7]
- The responsibility of a project manager is to get the best possible return on the investment in people

Poor management of people is one of the most significant contributions to project failure [7,11,12]

Four Critical Factors of Project Management [7]

- ❖ Consistency
  - ~ People should be treated in a comparable way
  - ~ No-one should feel that their contribution is undervalued
- ❖ Respect
  - ~ Respect that people have different skills
  - ~ All members of a team should have the opportunity to contribute

Four Critical Factors of Project Management [7]

- ❖ Inclusion
  - ~ People contribute best when they feel others listen to them and take account of their proposals
  - ~ Even the views of junior staff should be considered in a good working environment
- ❖ Honesty
  - ~ Be honest about what goes bad and well
  - ~ Be honest about your own level of technical knowledge
  - ~ Being “found out” is too costly to risk playing this game

Keeping Programmers Happy [3]

- The four major factors involved in satisfying programmers
  - Material reward and opportunities
  - Challenge and interest in the work itself
  - Employee benefits, working conditions and status of the organisation in its context
  - Competence of supervisors and leaders
- Variation in the programming task can be a bigger issue than salary
- Not uncommon for a programmer to quit right after a raise

Appraising Programmers

- After assessment, a team member with the lowest productivity had to leave the team
  - However, this person was the one with the building and maintenance rules in the group
  - Shortly after the team member left the team, the team fell apart
- Productivity is not the only factor when assessing a team member
  - Need to look at both task-specific and social-emotional aspects

Think about Status!

- Different programming tasks have different status
  - ~ Maintenance programming
  - ~ Testers
  - ~ Programming web applications
- Persisting reputation or co-operating?

- *Does status play a role?*

Exercise

❖ *Identify **three** incidents or times when you felt particularly pleased or **happy** about something related to your studies. Identify **three** occasions when you were particularly **dissatisfied** with your work. Compare your findings with the person(s) sitting next to you, and try to identify any patterns.*

Motivating People [7]

- Social needs
  - Time to meet co-workers
- Esteem needs
  - Be valued by the organisation
  - Monetary rewards
- Self-realisation
  - Responsibility for their work
  - Demanding but not impossible tasks
  - Provide training to develop skills

How Motivate?

- Set specific goals—*demanding but acceptable, preferably in conjunction with you*
- Provide feedback—*give regular feedback on your progress vs. your goals*
- Consider job design [8]
  - Increase *variety of activities* (increases job satisfaction [6])
  - Increase *responsibility*

Unmotivated People are Unproductive

- Cannot realise full potential or develop herself
  - ~ *"If I keep working with this outdated technology, I will never be able to get out of it!"*
  - Damages communication
  - Produces crap or doesn't care to report bugs because she wants to move on
  - Will work on the *wrong thing* because it is more interesting, etc.
  - Wont produce anything, which is demotivating in itself

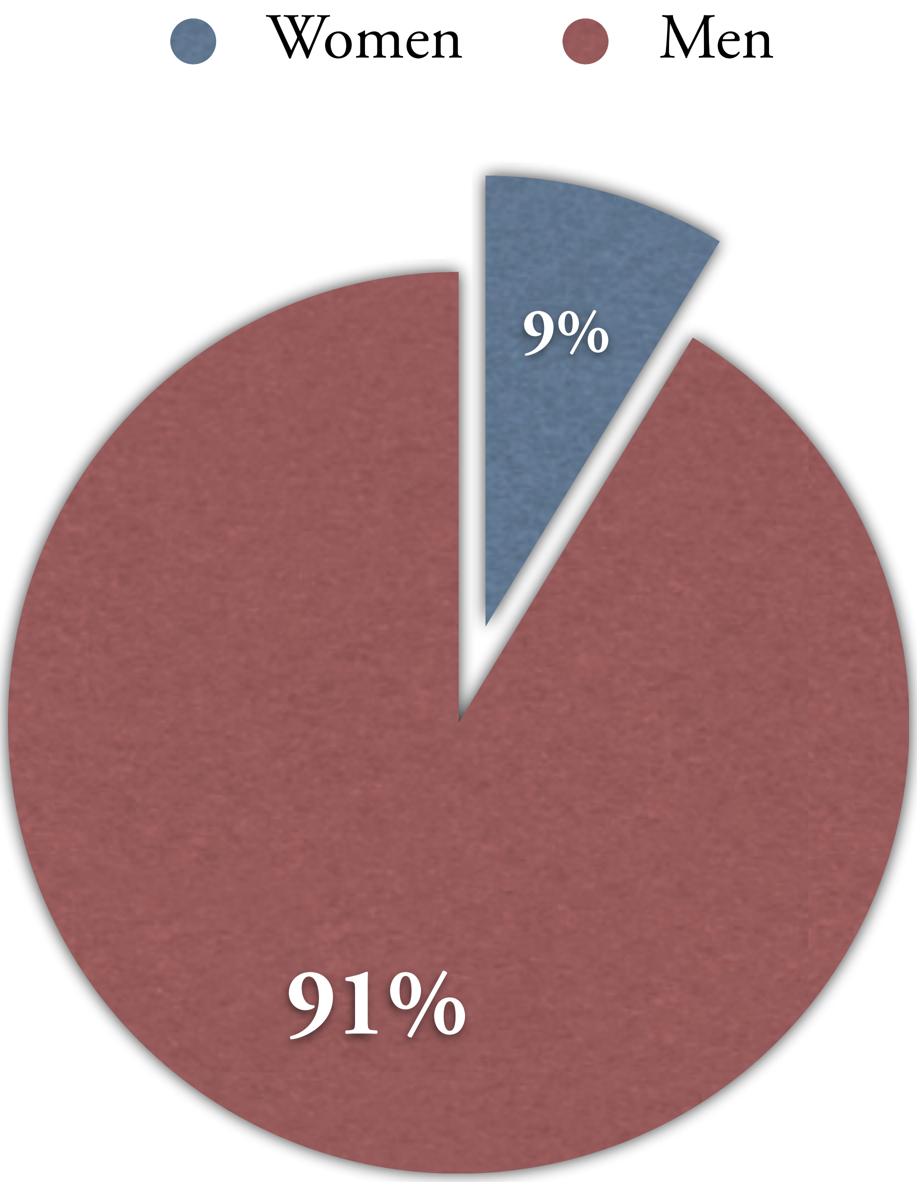
*Note — **de**-motivation is contagious*

Working with Teams

Factors that Influence Group Working [11]

- Group composition
  - Right balance of skills, experience and personalities
  - Gender
- Group cohesiveness
  - Team or group of individuals working together
- Group communication
- Group organisation
  - Respect leadership
  - Shared values

The Importance of Gender



Personality Traits [3,9]

- Compliant (*interaction-oriented*)
  - Wants to work with others and be helpful
- Aggressive (*self-oriented*)
  - Wants to earn money and prestige
  - It is sometimes claimed that aggressive people have trouble understanding that not all people are aggressive
- Detached (*task-oriented*)
  - Wants to be left alone to be creative

- Italicised terms from [9]

Balance of Personalities [9]

- Personalities bring different things into the mix
  - Task-oriented persons are generally more technically competent
  - Self-oriented persons are good at pushing projects to finish
  - Interaction-oriented persons are good at detecting tension and disagreement early, and facilitate communication in a group
- **No group should be without interaction-oriented people**
- Heterogeneity is *more important in large teams* than in small teams [10]
  - However, heterogeneity between team leader and team members is always significant

A Good Balance of Personalities [12]

- The chair
  - *Good at running meetings, being calm, strong and tolerant (not necessarily smart)*
- The plant
  - *Good at generating ideas and solve problems*
- The monitor-evaluator
  - *Good at evaluating ideas and solutions and select the best ones*
- The shaper
  - *Worries a lot, helps the team to direct attention to the important issues*

A Good Balance of Personalities [12]

- The team worker
  - ~ *Good at creating a good working climate and keeping co-workers happy*
- The resource investigator
  - ~ *Good at finding physical resources and information*
- The completer–finisher
  - ~ *Oriented at finishing tasks*
- The company worker
  - ~ *Team player who is willing to "take one for the team", e.g., do less attractive assignments*

Gerald Weinberg's Rules for Forming a Team

- Give the best possible programmers you can find sufficient time so you need the smallest number of them
  - Maximal competence
  - Minimal co-ordination and co-operation overhead
  - The smallest number is never one
  - Sufficient time must include time to form a team (good to choose a pre-existing team)
- Getting a team started before it is given a critical task is a good idea

Team Forming Phases [13]

- Forming
  - *The members of the group get to know each other and establish ground rules for behaviour*
- Storming
  - *Conflicts arise as various members of the group try to exert leadership*
  - *The methods of operation are established*
- Norming
  - *Conflicts are settled and the feeling of group identity emerges*
- Performing
  - *The group tackles the task at hand*
- Adjourning
  - *The group disbands*

Team Forming Phases [13]

- Forming
  - *The members of the group get to know each other and establish ground rules for behaviour*
- Storming
  - *Conflicts arise as various members of the group try to exert leadership*
  - *The methods of operation are established*
- Norming
  - *Conflicts are settled and the feeling of group identity emerges*
- Performing
  - *The group tackles the task at hand*
- Adjourning
  - *The group disbands*

Ego-less Programming [3]

- ~ *“The problem of the ego must be overcome by a restructuring of the social environment”*

❖ Advantages of ego-less programming

- ~ Finds bugs early—saves money on debugging
- ~ More people know the code, less dependency on original implementer
- ~ Writing a program to be read generally produces code that is better structured and have fewer errors from the start
- ~ Harder to keep people in the dark about progress—which is good for everyone

Commandments of Ego-less Programming

1. Understand and accept that you will make mistakes
2. You are not your code
3. No matter how much “karate” you know, someone else will always know more
4. Don't rewrite code without consultation
5. Treat people who know less than you with respect, deference, and patience
6. The only constant in the world is change
7. The only true authority stems from knowledge, not from position
8. Fight for what you believe, but gracefully accept defeat
9. Don’t be “the guy in the room”
10. Critique code instead of people—be kind to the coder, not to the code

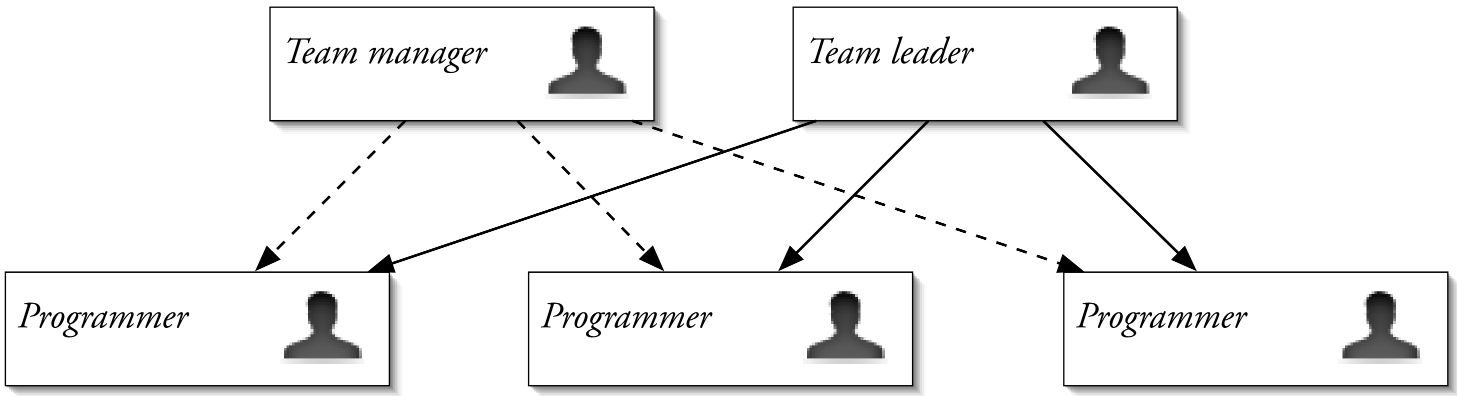
Ego-less Teams

- Leadership is formal
- Informal leadership varies during a project
- Encourage team members to find faults in code
- Making faults is expected and accepted
- Encourage a group identity
- Encourage *group ownership* of produced code
- Strengths
  - Very productive
  - Work well (or best) with difficult problems
- Downsides
  - Difficult to introduce into a non-democratic environment
  - Hard to create

Democratic Teams

- A team of peers
  - Counterintuitive for large-scale projects
  - Student's and beginners like them (power!)
- Virtual hierarchies, where seniority counts are generally put into place
- Software engineers can reduce any team methodology to democratic teams
- Downside
  - "Everyone is in charge and no one is responsible"
- Upside
  - Anyone is replaceable (competence is sum of parts)

“Modern Team Structure”



- Upholds the fundamental managerial principle—every employee reports to *only one* manager
- Team leader is present in reviews, but does not appraise
- Team manager cannot make product promises
- This approach can be easily up-scaled to several teams
- Good practice is to decentralise decision making where appropriate

Synchronise-and-Stabilise Teams

- ❖ Windows 2000 facts
  - ~ 30 million lines of code
  - ~ 3000 programmers and testers
  - ~ Extensive code reuse from NT 4.0
- ❖ Small parallel teams
  - ~ 3–8 developers
  - ~ 3–8 testers (work 1–1 with developers)
  - ~ Team is given overall task specification
  - ~ Team may design the task as they wish
- ❖ Few but strict rules
  - ~ Must check-in at the specified time
  - ~ Code that breaks the build must be fixed immediately

Synchronise-and-Stabilise Teams

- Two important factors
  - Daily synchronisation step
  - Individual components always work together

Extreme Programming Teams

- Programming pairs are formed from the larger team
  - Pairs are rotated frequently
  - Pairs distributes knowledge of the software
  - Pairs enable experienced developers to pass on knowledge to less experienced ones
  - One draws up test-cases, the other implements according to test specifications
- All teams work in the same area, which is said to promote *group ownership of code*

Choosing A Good Team Organisation

- Ego-less teams
- Democratic teams
- Modern hierarchical programming teams
- Synchronise-and-stabilise teams
- Extreme programming teams

Creating & Maintaining A Healthy Environment

- This is an open problem
- We behave the way we see people behaving around us
- Shaping the new programmers
  - Is encouraged to *e.g.*, seek aid
  - Is ridiculed for her stupidity
  - Does someone ever ask her for help?
- Cash award anecdote
  - A group protecting itself against a common enemy
  - Group philosophy might not be consistent with management philosophy
  - Working in a good team is a reward in itself

Creating & Maintaining A Healthy Environment

- “What can these trainees possibly teach me?”
  - Trying to hide your program from the less experienced
  - Being counterproductive
  - Ridiculing people for “having to work together to be productive”
- Many software managers are judged on deliverables, not on their ability to create good teams (that can deliver)
  - Short-term vs. long-term
  - Aggressive personalities
- *See e.g., [14]*

The Importance of Informal Channels

- A couple of age-old, but interesting examples
  - The secretary who could see the job listing
  - Moving the vending machines
  - Replacing the old elevators
  - Knowledge exchange in the waiting line to the computer room

- *How does this relate to outsourcing?*

Working Environments

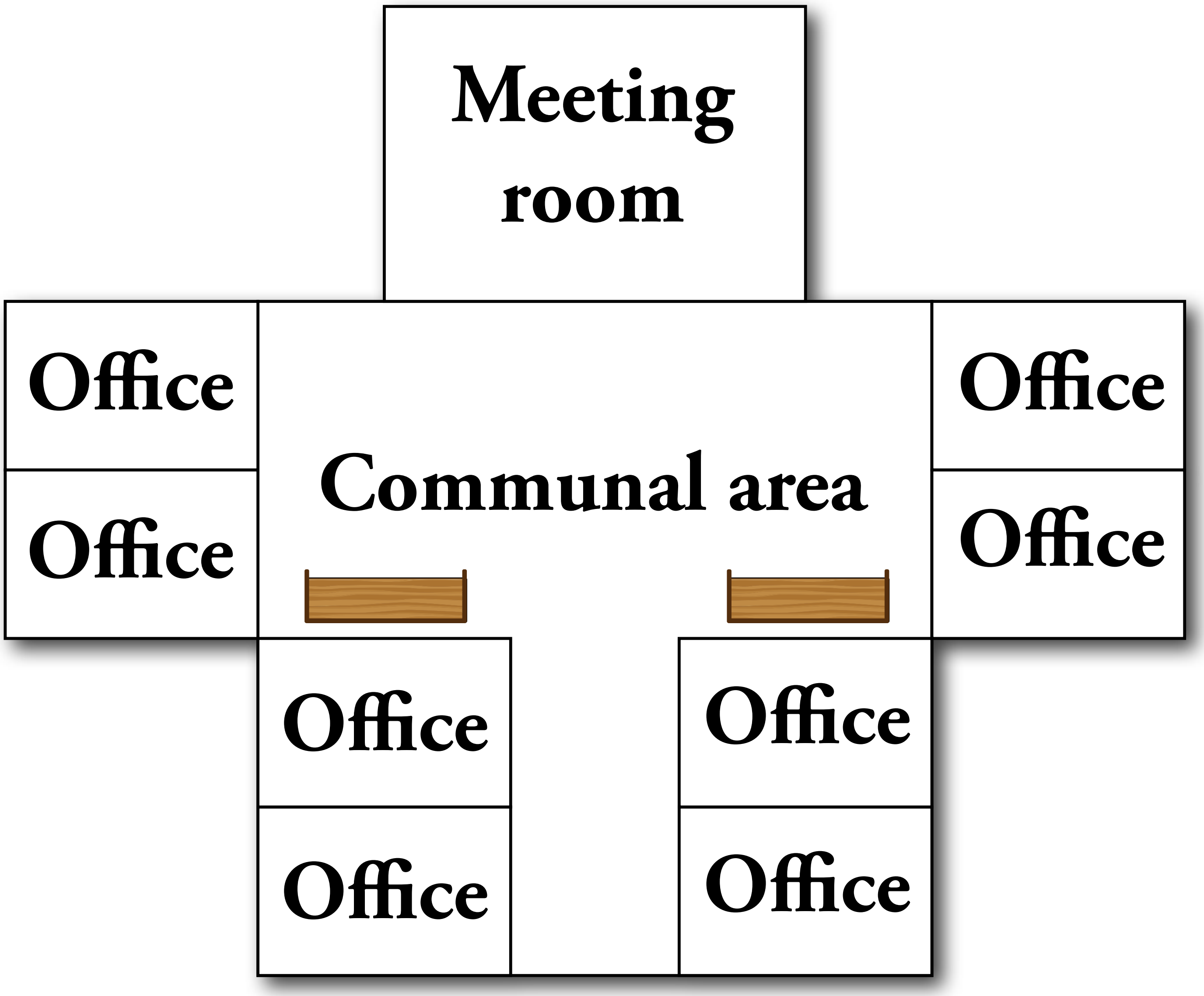
Working Environments

- Have great effects on people's performance and job satisfaction
  - ~ Group *behaviour* is affected by architectural organisation and communication facilities
  - ~ Group *communication* is affected by building architecture and workspace structure
  - ~ Bad working conditions increases staff turnover

McCue 1978

- Study of development staffs working in large open-plan office areas (sometimes cubicles)
  - Result showed such working environments are neither popular nor productive
- Most important environmental factors
  - Privacy
  - Outside awareness
  - Personalisation
- *People like individual offices that they can organise to their taste and needs*
- Development teams need spaces where all members of the team can meet together—both formally and informally
  - Must accommodate privacy
  - Group individual offices round larger meeting rooms

McCue 1978


Meeting room
Office
Office
Communal area
Office
Office
Office
Office
Office
Office

DeMarco & Lister 1985

- Comparing programmer productivity
  - Software engineers *with* individual offices
  - Software engineers *without*
- Result: private workspace with ability to cut off interruptions yields *twice the productivity*

The Code War Games
Case for *Peace & Quiet*

Environment	*Top 25%*	*Bottom 25%*
Dedic. work space	78 ft^2	46 ft^2
Acceptably quiet	57% yes	29% yes
Acceptably private	62% yes	19% yes
Can silence phone	52% yes	10% yes
Can divert calls	76% yes	19% yes
Often interrupted	38% yes	76% yes

Working Conditions

- Do programmers have quiet working conditions?
  - From “The Joel Test: 12 Steps to Better Code” [17]
  - Programming requires focus—you want to be “in the zone”
- With cubicles
  - Mutt ask Jeff a simple lookup-question to save 15 seconds over a 30 second googling
  - Jeff loses 15 minutes of productivity (or more)
- With proper offices
  - Asking Jeff takes 45 seconds—so googling is 15 seconds faster
  - Mutt loses 15 seconds, Jeff saves 15 *minutes*

Beck 2000

- Working environments in e*Xtreme Programming*
- By default, coding etc. goes on in the *communal area*
  - This is claimed to promote group ownership of code, etc.—group thinking and identity
- Rooms or cubicles for teams wishing to work alone are provided close by
- While taking an opposite angle, the ideas are the same as in McCue—provide both *individual space* and *group space*

Cultural Factors

Culture and Agile Methods

- Is agile methodology a “western thing?”
- High power distance cultures might not benefit from agile methodology
  - Cannot contradict the boss
  - Cannot speak freely in front of the manager
  - Losing face and customer-on-site practices may conflict
  - Out-in-the-open risk assessment will become harder

Cultural Patterns and Anti-Patterns [18]

- Pattern
  - Proxy
- Anti-patterns
  - Yes (but no)
  - We will take you literally
  - We are one single team
  - The customer is king

End of Line

References

[1] Robert L. Glass, *Facts and Fallacies of Software Engineering,* Addison-Wesley, 2003

[2] Joel Spolsky, interview by ITConversations 2004-09-20 (http://www.itconversations.com//clip.php?showid=207)

[3] Gerald Weinberg, *The Psychology of Computer Programming, Silver Anniversary Addition*, Dorset House, 1998.

[4] Joel Spolsky, *Reading Code is Like Reading the Talmud*, 2000 (http://www.joelonsoftware.com/articles/fog0000000053.html)

[5] Fred Brooks, *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition. Reading, MA: Addison-Wesley, 1995

[6] J. D. Couger and R. A. Zawacki, *What motivates DP professionals?*, Datamation, 24, 1947

[7] Ian Sommerville, *Software Engineering*, 8th edition, Addison-Wesley, 2006

[8] F. Herzberg, B. Mausner and B. Snyderman, *The Motivation to Work*, N.Y., John Wiley & Sons, 1959

[9] B. M. Bass and G. Dunteman, *Behaviour in groups as a function of self, interaction and task-orientation*, Journal of Abnormal Social Psychology, 66(4), 1963

Continues on the next page!

References

[10] Narasimhaiah Gorla and Yan Wah Lam, *Who Should Work with Whom?—Building Effective Software Project Teams*, Communications of the ACM, June 2004, 47(6)

[11] Bob Hughes and Mike Cotterell, *Software Project Management*, 3rd ed. McGraw-Hill, 2002

[12] R. Meredith Belbin, *Management Teams: Why They Succeed or Fail*, Heinemann, 1981

[13] B. W. Tuckman and M. A. Jensen, *Stages of small-group development revisited*, Group & Organization Studies, 2 (4), 1977

[14] Tom DeMarco and Timothy Lister, *Peopleware: Productive Projects and Teams*, (1987), Dorset House Publishing Company, Incorporated; 2nd edition (February 1, 1999)

[15] G. M. McCue, *IBS's Santa Teresa laboratory: architectural design for program development*, IBM Systems Journal, 17(1), 1978

[16] Kent Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley Professional, 2000

[17] Joel Spolsky, *The Joel Test: 12 Steps to Better Code*, 2000, (http://www.joelonsoftware.com/articles/fog0000000043.html)

[18] Eve MacGregor, Yvonne Hsieh and Philippe Kruchten, *Cultural Patterns in Software Process Mishaps: Incidents in Global Projects*, in Proceedings of The Workshop on Human and Social Factors of Software Engineering 2005 (HSSE 2005), May 16, St. Louis, Missouri, USA

Continues on the next page!

References

[19] B. Curtis, W. E. Hefley et al., *The People Capability Maturity Model: Guidelines for Improving the Workforce*, Boston, Addison-Wesley, 2001[20] M. E. Fagan, Advances in Software Inspections, IEEE Transactions of Software Engineering, SE-12 (7), 1986

[21] R. W. Selby, V. R. Basili, et al. *Cleanroom software development: an empirical evaluation*, IEEE Transactions of Software Engineering, SE-13(9), 1987

[22] T. Gilb and D. Graham, *Software Inspection*, Wokingham, Addison-Wesley, 1993

[23] J. Barnard and A. Price, *Managing code inspection information*, IEEE Software 11(2), 1994

[24] A. P. Ershov, *Aesthetics and the Human Factor in Programming, Communications of the ACM* 15(7), 1972