Software Engineering CS3003

Lecture 10 - Software Engineering Techniques in action

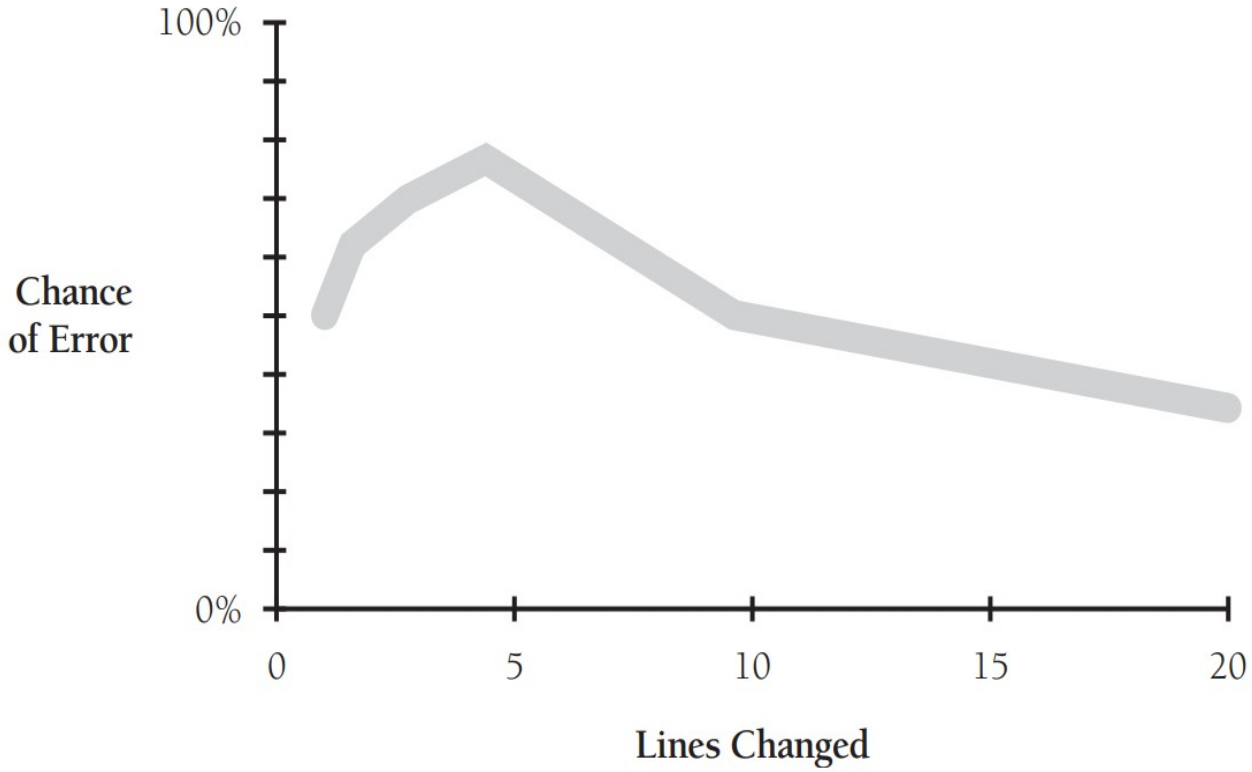


Figure 24-1 Small changes tend to be more error-prone than larger changes (Weinberg 1983).

Structure of this lecture

- Some topics in Software Engineering that we need to be aware of:
  - Software cost estimation
  - 80/20 rule
  - Code re-engineering
  - Developer motivation

Lecture schedule

Week	Lecture Topic	Lecturer	Week Commencing
1	Introducing the module and Software Engineering	Steve Counsell	20th Sept.
2	Software maintenance and Evolution	Steve Counsell	27th Sept.
3	Software metrics	Steve Counsell	4th Oct.
4	Software structure, refactoring and code smells	Steve Counsell	11th Oct.
5	Test-driven development	Giuseppe Destefanis	18th Oct.
6	Software complexity **Coursework released Tues 26th Oct.**	Steve Counsell	25th Oct.
7	**ASK week**	**N/A**	**1st Nov**
8	Software fault-proneness	Steve Counsell	8th Nov.
9	Clean code	Steve Counsell	15th Nov.
10	Human factors in software engineering	Giuseppe Destefanis	22th Nov.
11	SE techniques applied in action	Steve Counsell	29th Nov.
12	Guest Lecture (tba) **Coursework hand-in 6th December**	Guest Lecture	6th Dec.

Software Cost Estimation

Software estimation techniques

- Algorithmic cost modelling
- Expert judgement
- Estimation by analogy
- Parkinson's Law
- Top down, bottom up
- 3-point estimation

Algorithmic cost modelling

- A formulaic approach based on historical cost information and which is generally based on the size of the software

Algorithmic cost modelling

- Cost is estimated as a mathematical function of product, project and process attributes
- Values estimated by project managers (PM)
  - For example: Effort = A $\times$ B $\times$ M
    - A is anticipated system complexity decided by the PM, B the number of hours available and M a value reflecting the number of product, process and people attributes
- Most models are basically similar but can have wildly different values for A, B and M

Expert judgement

- One or more experts in both software development and the application domain
  - Use their experience to predict software costs
  - Process iterates until consensus is reached
- Advantages: Cheap estimation method
  - Can be accurate if experts have direct experience of similar systems
- Disadvantages: Very inaccurate if there are no experts

Estimation by analogy

- Cost of a project computed by comparing the project to a similar project in the same application domain
- Advantage:
  - Accurate if project data available
- Disadvantages:
  - Impossible if no comparable project has been tackled
  - Needs systematically maintained cost data
- NASA use this a lot
  - Because many of their systems are similar
    - So they use past systems as a good guide to future ones

NASA Software Cost Estimation Model:
An Analogy Based Estimation Method

Jairus Hihn
Leora Juster
Jet Propulsion Laboratory/California Institute of Technology

Tim Menzies
George Mathew
North Carolina State University

James Johnson
National Aeronautics and Space Administration

Abstract

The cost estimation of software development activities is increasingly critical for large scale integrated projects such as those at DOD and NASA especially as the software systems become larger and more complex. As an example MSL (Mars Scientific Laboratory) developed at the Jet Propulsion Laboratory launched with over 2 million lines of code making it the largest robotic

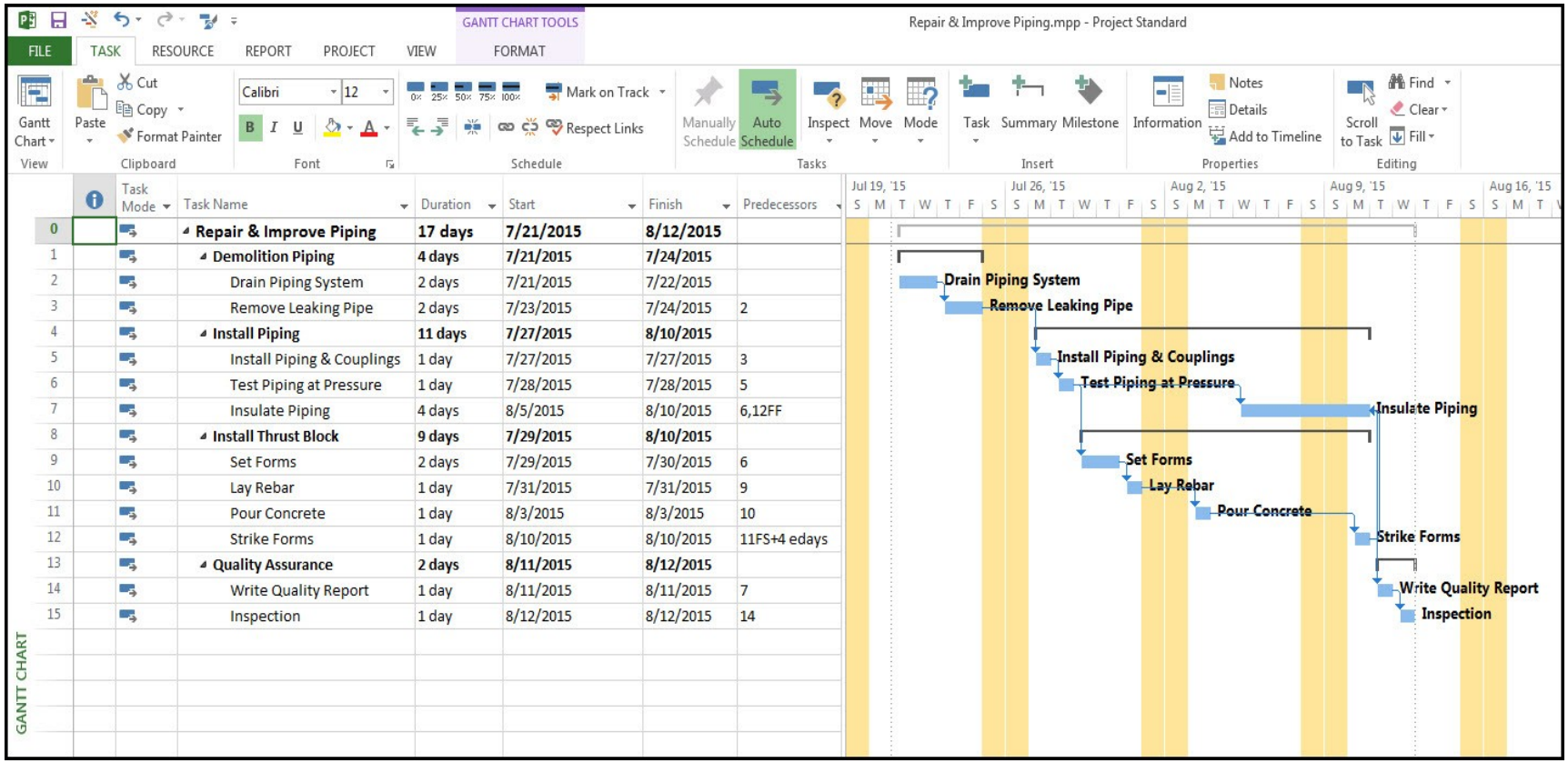
Parkinson's Law

- Parkinson’s Law states that work expands to fill the time available
  - So whatever costs you set, you’ll always use them up
- To combat this, the approach states that:
  - The project will cost whatever resources are available and that’s all it’s getting
- Advantages: No overspend
- Disadvantages: System is usually unfinished
  - The cost is determined by available resources rather than by objective statement

Top-down and bottom-up estimation

- Top-down estimation
  - Start at the top level and assess the overall system functionality and effort required for only the high-level components and add these up
- Bottom-up
  - Start at the component level and estimate the effort required for each component
  - Add these efforts to reach a final estimate

Top down vs bottom up estimation



Source: tensix.com

3-point estimation

- To calculate a probable project cost with the 3-point estimating technique, you need to create three types of estimates:
  - Point 1: An optimistic estimate – the amount of money your team hopes to spend on project activities
  - Point 2: A pessimistic estimate – the project's cost in the worst-case scenario when many things go not as expected
  - Point 3: The best-guess estimate – a realistic figure, the amount of money the team will most likely spend on project activities.
- The final project cost value is then calculated by using the following equation, also known as the PERT formula:
  - [Optimistic Estimate + Pessimistic Estimate + (4 × Best-guess Estimate)] / 6

Factors affecting developer productivity

- Application domain experience
  - Knowledge of the application domain is essential for effective software development.
  - Engineers who already understand a domain are likely to be the most productive
- Process quality
  - The development process used can have a significant effect on productivity.
- Project size
  - The larger a project, the more time required for team communications.
  - Less time is available for development so individual productivity is reduced
- Technology support
  - Good support technology such as tools, configuration management systems, etc. can improve productivity
- Working environment
  - A quiet working environment with private work areas contributes to improved productivity

A study of programmer productivity

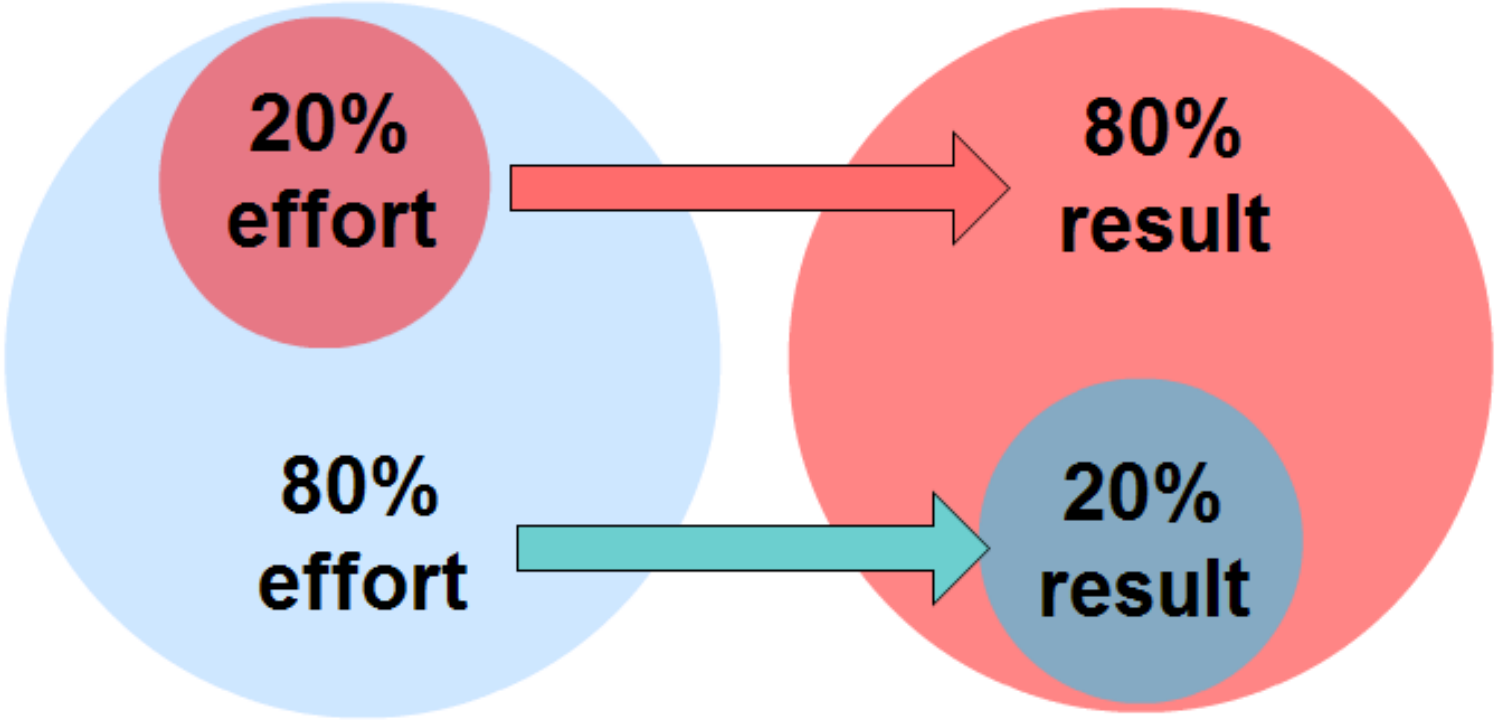
- Original study conducted in the late 1960s by Sackman, Erikson, and Grant
- They studied professional programmers with an average of seven years' experience and found that:
  - The ratio of initial coding time between the best and worst programmers was about 20 to 1
  - The ratio of debugging times over 25 to 1
  - The ratio of program execution speed about 10 to 1
  - They found no relationship between a programmer's amount of experience and code quality or productivity

80/20 Rule also known as Pareto Analysis

History of Pareto Analysis

- The Pareto effect named after Vilfredo Pareto,
  - Economist
  - Sociologist
  - Lived from 1848 to 1923
  - Established that 20% of the population in Italy owned 80% of the land

80/20 illustrated



Control-alt-success.com

Examples in day-to-day life

- 80% of customer complaints arise from 20% of products and services
- 20% of products and services account for 80% of your profit
- 20% of sales force produces 80% of your company revenues

IBM study

- In 1963, a study by IBM discovered that roughly 80% of a computer's time was spent executing only 20% of the instructions
- A counterintuitive breakthrough at the time
- IBM rewrote the operating system to make the 20% faster and easier to use
  - Thereby improving computer performance

Do 80% of faults exist in 20% of classes?

- If 20% of classes are responsible for 80% of bugs this has potentially significant implications
- If it is possible to identify the 20% of specific classes responsible for 80% of faults in a system:
  - Our fault-detection efforts can be focused on rigorous testing, doing code reviews, planning refactoring activity

Re-engineering code

Software re-engineering

- Two complementary viewpoints:
  - Reorganising and modifying existing software systems to make them more maintainable
  - The examination of a subject system to reconstitute it in a new form and the subsequent implementation of the new form

Why is maintenance so hard to do?

- Maintenance staff are rarely the development staff
- People move jobs:
  - If not from the company perhaps to new responsibilities
- Our code memory is short lived.
  - Code you wrote one month ago is barely recognizable by even you
- Coders hate doing documentation
  - Often leave poorly documented code or documentation

Code Restructuring

- Source code is analyzed using a restructuring tool or visually
  - Poorly designed code segments are redesigned
- Violations of proper programming constructs noted and
  - Code is then restructured
    - This can often be done automatically
- The resulting restructured code is reviewed and tested
  - To ensure that no anomalies have been introduced
- Internal code documentation is updated
  - This last part is often the one that is dangerously ignored

“Green” code

“It may seem strange to think about the potential of “green code” – what’s environmentally unsustainable about a bunch of letters and numbers? But when we consider the strain that cluttered code puts on our collective computer systems and the energy expenditures slowdowns can cause, the problem becomes clear: Green code has a lot of potential, and businesses are taking notice.”

Anna Wilson
(triplepundit.com/story/2017/potential-green-code-business/20256)

Developer Motivation

What studies report…

McConnell (1998) points out,

> *"Motivation is a soft factor: It is difficult to quantify, and it often takes a back seat to other factors that might be less important but are easier to measure. Every organisation knows that motivation is important, but only a few organizations do anything about it. Many common management practices are pennywise and pound-foolish, trading huge losses in motivation and morale for minor methodology improvements or dubious budget savings."*

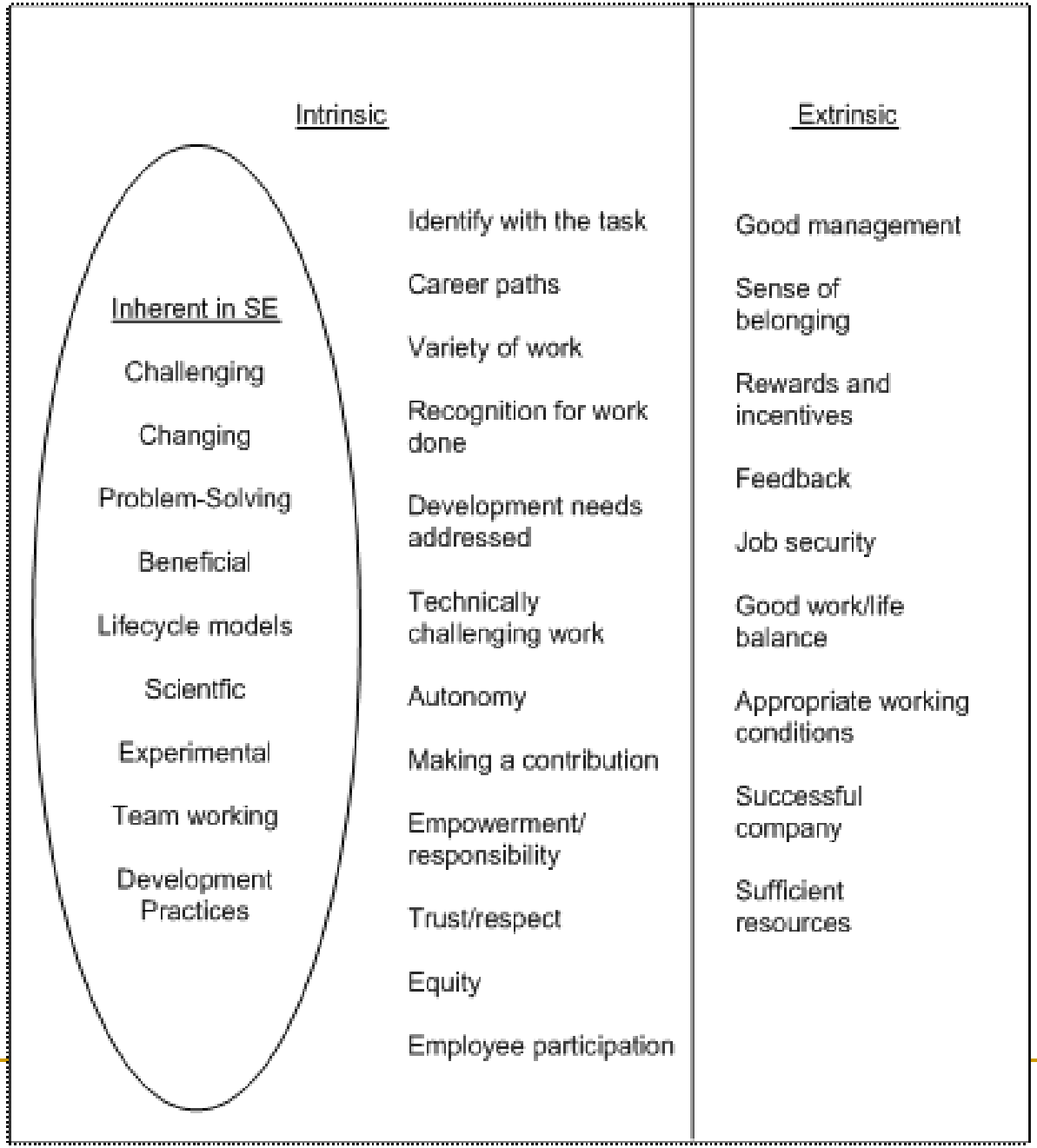
Defining motivation

- Difficult to define as fairly intangible.
- …to give reason, incentive, enthusiasm or interest that causes an action or behavior
- Two types of motivation:
  - Intrinsic – internal desire
  - Extrinsic – external compulsion

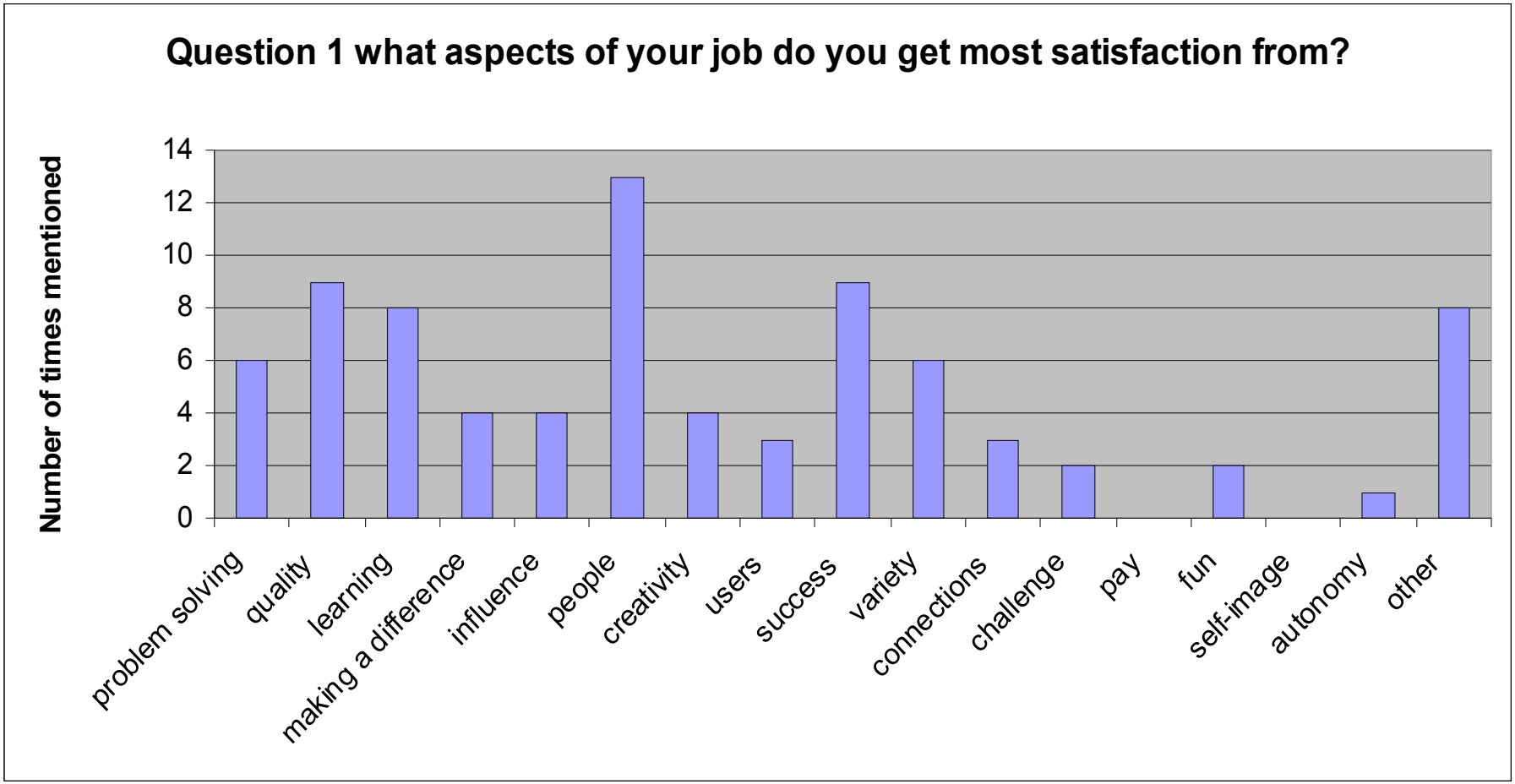
Motivation-Hygiene Theory (Herzberg et al. 1959)

Extrinsic factors	Intrinsic factors
Pay	Achievement
Interpersonal relations, subordinate	Recognition
Status	The work itself
Interpersonal relations with superior	Responsibility
Interpersonal relation with peers	Possibility of growth
Technical supervision	Advancement
Company policy and administration	
Work conditions	
Personal life	
Job security	

Motivators


Intrinsic
Extrinsic
Inherent in SE
Challenging
Changing
Problem-Solving
Beneficial
Lifecycle models
Scientfic
Experimental
Team working
Development Practices
Identify with the task
Career paths
Variety of work
Recognition for work done
Development needs addressed
Technically challenging work
Autonomy
Making a contribution
Empowerment/ responsibility
Trust/respect
Equity
Employee participation
Good management
Sense of belonging
Rewards and incentives
Feedback
Job security
Good work/life balance
Appropriate working conditions
Successful company
Sufficient resources

Question: what aspects of your job do you get most satisfaction from?



May’s Exam things

Two points to start with

- There are no such thing as:
  - Topics that you should leave out of your revision or
  - Topics that you should focus on more than any other
- You should revise everything

Exam paper format

- Five questions in total on the paper
  - You must answer any **4** of that 5 of your choice
- Three hours
- Time you should spend on each question?
  - ~35-40 minutes if it was me
- Every question has sub-parts
  - Worth varying amounts of marks
- Clarity is important

Exam paper format

- Are you allowed to answer all 5 questions?
  - Answer: Yes, but you will get credit ONLY for your best FOUR answers
- But a warning about people who have done this

Exam will cover three aspects

- The lecture material
- The labs and your coursework
- Your own experience and thoughts
  - Maybe down to the background reading you've done
    - Some background reading was given at the end of the slides for each lecture
- The guest industry lecture by Dr Shippey is NOT examinable
  - The guest lecture was for your own understanding of topics only

How it all fits together…

Lecture schedule – re-visited

Week	Lecture Topic	Lecturer	Week Commencing
1	Introducing the module and Software Engineering	Steve Counsell	20th Sept.
2	Software maintenance and Evolution	Steve Counsell	27th Sept.
3	Software metrics	Steve Counsell	4th Oct.
4	Software structure, refactoring and code smells	Steve Counsell	11th Oct.
5	Test-driven development	Giuseppe Destefanis	18th Oct.
6	Software complexity **Coursework released Tues 26th Oct.**	Steve Counsell	25th Oct.
7	**ASK week**	**N/A**	**1st Nov**
8	Software fault-proneness	Steve Counsell	8th Nov.
9	Clean code	Steve Counsell	15th Nov.
10	Human factors in software engineering	Giuseppe Destefanis	22th Nov.
11	SE techniques applied in action	Steve Counsell	29th Nov.
12	Guest Lecture (tba) **Coursework hand-in 6th December**	Guest Lecture	6th Dec.

Reading for the week

- Somerville, Chapter 23 on software cost estimation
- Hall T, Sharp H, Beecham S, Baddoo N., Robinson H (2008) ‘What do we know about software developer motivation?, IEEE Software, July/Aug 2008
- Bob Glass: Facts and Fallacies of Software Engineering
- This is the last lecture of material
  - The invited guest lecture by Dr Thomas Shippey of Dexda is next Week 12
    - The guest lecture is not examinable material

Finally……..