Lec 2 Evidence-based Software Engineering (EBSE)

Objectives

- [ ] Introduction to Evidence-Based Software Engineering
- [ ] Give some collected evidence about Software Engineering
- [ ] Have you think about the knowledge you have, and the evidence that supports it

---

Evidence can be produced in:

Quantitative (numeric and objective)

- Whether a **cause-effect relationship** exists
    - E.g using randomised experiments to check if the number of LOC can be used as a software quality metric
- Whether there are **associations between factors**
    - E.g using correlation analysis of observational data to investigate what type of development process is used per application domain

Qualitative (concerned with subjective phenomena)

- The causes and effects of the behaviour of software developers and managers
    - E.g is using ‘agile’ a source of satisfaction for developers?

Observational

- Measurements or surveys of a sample without trying to affect them

Experimental

- Sub-groups with one baseline and a ‘treatment’ for other groups

Systematic literature reviews

- Thorough analysis of the literature

Systematic mapping study

- Lightweight analysis of the literature

The **allegory of observational science** is that what we perceive may not be the full reality.

- Although 14 rocks are visible from any position around the garden, there are actually 15 rocks.
- “When we do observational science, we may see 14 stones but we don’t know if there are 14,15, of 256”

Limitations/TOV of empirical methods:

- **Internal**: factors that may affect the validity of the causal relationship between treatment and outcomes
- **External**: are the results generalisable to the intended population of interest
- **Construct**: how well the outcomes of the study are linked to the concepts or theory behind the study

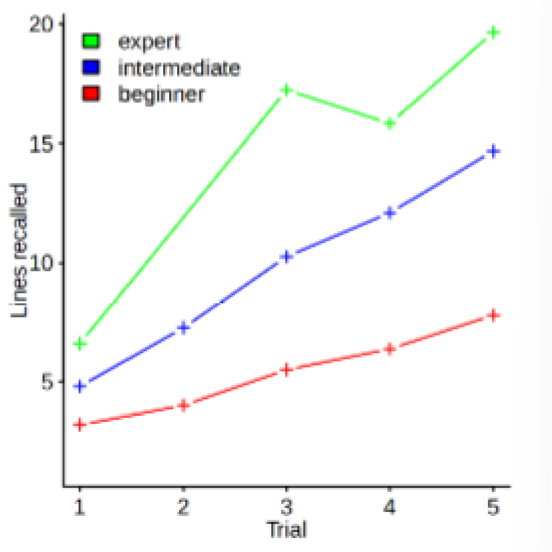
---

EBSE Studies

Software Engineers Expertise

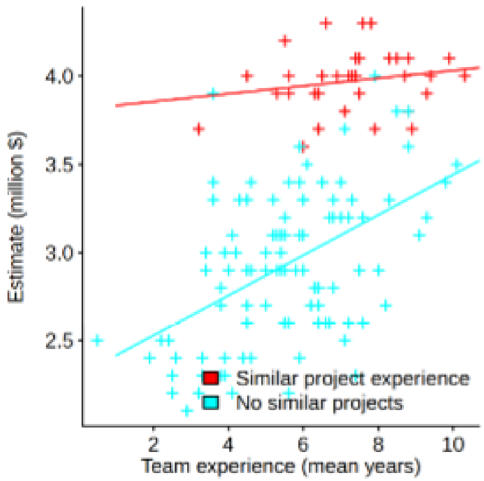
What is an expert?

- McKeithen et al. categorised programmer skill levels based on their deliberate practice time.
- **Experts** → at least 2000 hours of general programming experience.
- **Intermediates** → just completed a programming course
- **Beginners** → about to start a programming course
- In the experiment, groups were tasked with recalling the sequence of code in a programming project (Method).
- The results revealed that **recall performance can differentiate between different skill** levels, as experts continued to recognise familiar segments of code after the first trial, compared to lesser skilled subjects.



Expertise impact on estimation - How does experience impact effort estimation?

- Cost estimates depend on two types of team experience:
    - Average experience of team members.
    - Whether any team members have similar project experience.
- If no team members have similar project experience, cost estimates correlate with average team experience, with teams having greater experience producing higher cost estimates.
- If at least one team member has similar project experience, the relationship between average team experience and cost is weaker.
- Teams with similar project experience produce cost estimates closer to those of teams with the greatest average team experience.
- Less experienced teams tend to produce lower cost estimates due to:
    - Failure to include certain tasks included by more experienced teams.
    - Shorter estimated task durations compared to more experienced teams.



Software Quality Metrics

Ctrl -c/ ctrl -v → Poor Quality Software?

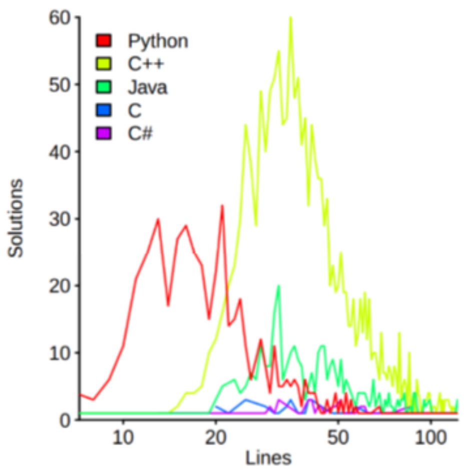
subsystem	arch	fs	kernel	mm	net	sound	drivers	crypto	others	LOC
arch	*25.1*	1.4	0.5	0.3	1.1	1.3	3.2	0.1	0.8	724,858
fs	1.4	*16.5*	0.6	0.5	1.7	1.2	2.2	0.0	0.7	475,946
kernel	3.0	1.8	*7.9*	0.6	2.3	1.6	2.8	0.1	0.8	30,629
mm	2.6	2.2	0.8	*6.2*	1.7	1.1	2.0	0.0	0.7	23,490
net	1.8	2.5	1.1	0.7	*20.7*	2.1	3.7	0.1	1.0	334,325
sound	2.3	2.0	1.0	0.6	2.2	*27.4*	4.6	0.2	1.1	373,109
drivers	2.3	1.7	0.6	0.4	1.8	2.0	*21.4*	0.1	0.6	2,344,594
crypto	2.3	2.2	0.3	0.1	1.1	1.5	2.5	*26.1*	2.2	9,157
others	3.8	1.9	0.8	0.4	1.7	1.5	2.6	0.3	*15.2*	49,016

Percentage of a subsystem's source code cloned within and across subsystems of Linux 2.6.6.

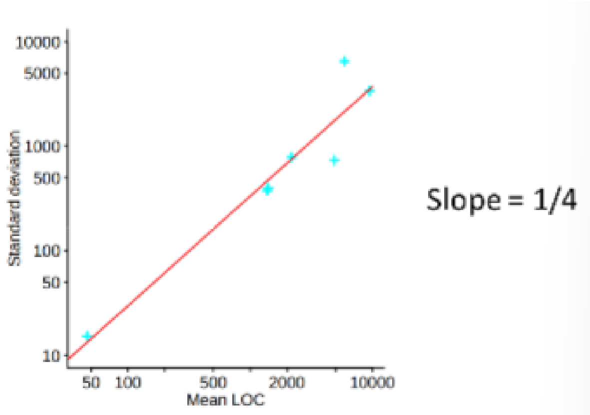
- Aimed at identifying copy-pasted code in large software suites and detecting copy-paste bugs.
- They believe that maintaining copy-pasted code would be very useful for programmers because it is commonly used in large-scale software such as operating system code and it can easily introduce hard-to-detect bugs.
- Their research focused on "forget-to-change" bugs caused by copy-paste.
  However, copy-paste can introduce many other types of bugs:
    - After the copy-paste operation, the programmer forgets to add some statements that are specific to the new copy-pasted segment.

#SLOC → Is the number of lines of code a good quality metric?

- Yes → as it tells you how big the software is but hard to say when it is too big.
- No → **significant variability** and little information → Bugs, Maintainability



- Graph shows the number of solutions to one problem in a Google Code jam competition, containing a given number of lines, grouped by programming language.



- Mean LOC against standard deviation of LOC, for multiple implementations of 7 distinct problems; line is a fitted regression model of the form_deviation for SLOC

Software Projects

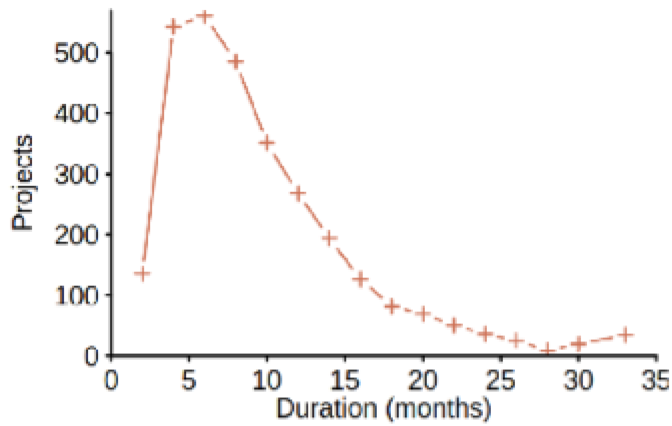


Figure 5.1 Number of projects having a given duration (2,992 projects).

- Most projects last less than ~10/12 months

Software projects size (SLOC)

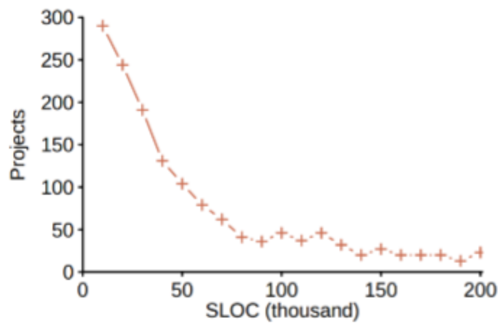


Figure 5.1 Number of projects delivered contain a given number of SLOC (1,859 projects).

- Most projects have less than ~50 KSLOC

Software projects size (SLOC)

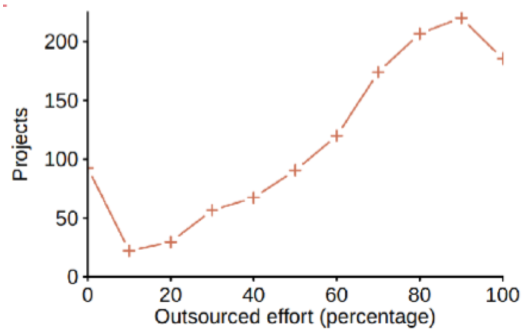
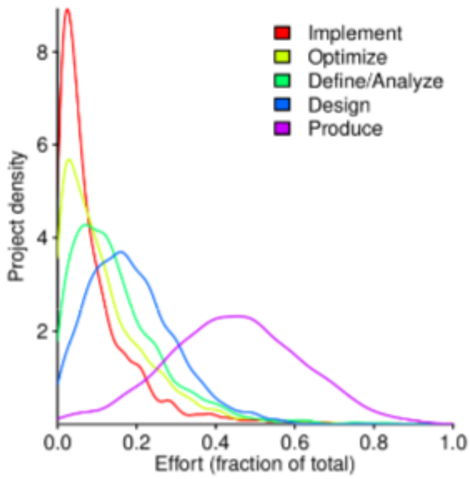


Figure 5.1 Number of projects using a given percent
of out-sourced effort (1,267 projects).

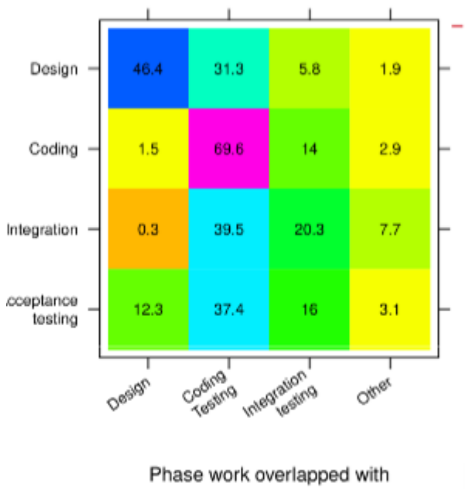
- 75% of projects outsource more than 75% of their effort

Software development process
Effort dedicated to each phase



- Implement = Deployment
- Optimise = Test
- Produce = Implement

Is the waterfall process sequential?

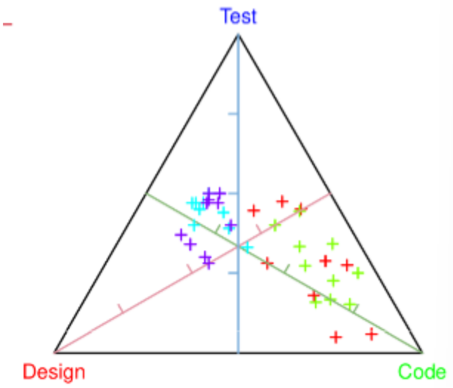


Waterfall: Design → coding + testing → integration testing → acceptance testing

- 31.3% of the time spent on design occurred during the coding and testing phase

- The waterfall model has never been sequential in practice

Impact of the application domain



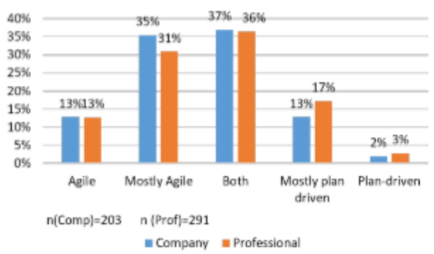
The graph shows the percentage distribution of effort across design/coding/testing for 10 computer manufacturer projects (red), 11 telecom projects (green), 11 space projects (blue) and defence projects (purple).

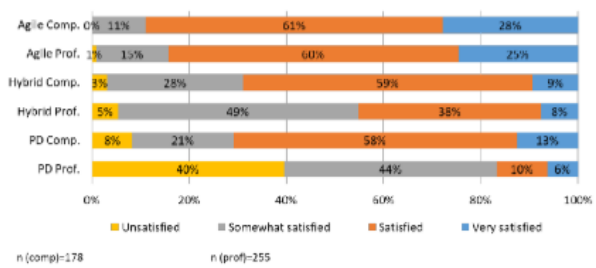
Two groups of domains:

- Space/defense
- Computer/telecom

There is less scope to recover from the failures of software systems operating in some space/defense environments; this operational reality makes it worthwhile investing more in design and testing.

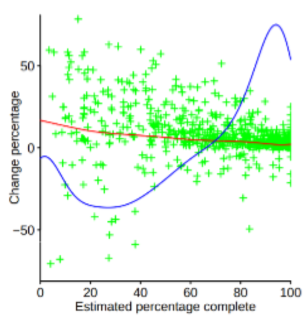
EBSE and agile methods





- The choice of project implementation strategy is strongly influenced by the risk profile of changes to requirements and company cash flow.

Planning Robustness



- Optimistic delivery date (on average)
- Pessimistic estimates are corrected early
- Optimistic estimation are corrected late: most requests to change the delivery data come within the last 30% of the initial estimation time (25% in the last 6.4% of remaining time)

- Larger changes at the beginning, smaller changes towards the ends

---

R&D→ little incentive to provide more evidence
EBSE is difficult → software is a human creation with a huge diversity of (application domains, organisation types, cultures & legacy)
Software industry → young, immature, led by trends/market/applications not regulations/standards/EB research