The impact of experience on software developer performance

Derek Jones
Knowledge Software Ltd
derek@knosof.co.uk
My Background

What developers actually wrote

Compiler writer – front ends, back ends, language translators

Static analysis – finding faults

What developers meant to write

Book- The New C Standard: An Economic and Cultural Commentary
www.knosof.co.uk/cbook/cbook.html
Introduction

Cognitive psychology

Predicting developer performance

A hypothesis

Source code measurements
Experiment
Results
Human Mental Characteristics

Orders-of-magnitude

$10^{-4}$ - $10^{-2}$  Biological band
$10^{-1}$ - $10^{+1}$  Cognitive band
$10^{+2}$ - $10^{+5}$  Social band

Abilities

Built-in – autonomic nervous system
Learned
Some Performance Factors

Performance improves with Practice
   Response time, error rate

   \[ E = c \, P^{-m} \]

Power law of forgetting
   Retention rate decreases with time
   \[ R = k \, T^{-n} \]
Developer Performance

What improves?
How much; how to measure; cost of measurement
Formula to calculate...

Source code
My interest; lots available; can be measured

Developers spend time interacting with code
Lots and lots of time
Doing things not generally done elsewhere
Binary Operator Precedence

Lots of rules

13'ish rules (shared by C, C++, Java, Perl, Python, C#)

\[ x + y \mid z \]

Amenably to measurement

Source code
Developer performance
Hypothesis

Every source occurrence provides practice
  Relative percentage a measure of relative practice

More practice aids learning/retention
  Practice only occurs when a decision has to be made
  Occurrences rare enough that performance not 'saturated'
  \[ P = x + y; \]
  \[ Q = a + b | c; \]

Prediction
  More source code occurrences \( \rightarrow \) better developer performance
Source Measurements

What measured

Large C programs
Visible source
Binary operators common to C/C++/Java/Perl/etc.
Operator pairs in expressions

\[
\begin{align*}
x &= y + z; \\
a &= b + c * d ;
\end{align*}
\]

Ignored (not considered to be operators)

\[
= \quad . \quad \rightarrow \quad [ ] \quad ( )
\]
The Experiment

The ACCU

C and C++ user group: now includes Java, C#, Perl + others
Annual conference: 250+ professional developers
Willing to make lunchtime slot available

Practical constraints

Time: 40 minutes
Venue: Room at a conference
Subjects: Volunteers willing to give time during lunch
What Subjects asked to do

Three stage problem, repeated

Remember information

  zip = 4;
  zap = 8;
  bat = 6;

Time filler task

  x + y * z
  p || q >> r

Recall information
Results '06/'07 Overview

Numbers
Subjects (years experience): 17 (14.6) /6 (14.5)
Answers: 123.5/116.2  sd 35.0
Percent correct: 66.7/63.3  sd 8.7
Random answers, binomial distribution: 0.1% prob > 60% correct

Bradley-Terry Statistics

Highest
Lowest
Performance/Source Correlation

Pearson’s $r = 0.64$

Pearson’s $r = 0.27$
33% incorrect!?!?

Implication for faults in real code
  2% of expressions contain two or more binary operators
  Implies almost 1% of expressions 'wrong'

'Naked' expressions rare in code
  Expressions generally exist within a context
  Expressions often contain context information
Context Information

\[ x + y \mid z \]

arith + context_clue \mid bit
Source Measurements

Names of operand identifiers

Arithmetic names: size, len, count

Bitwise names: flags, status, mask

Boolean names: finished, done, started

Anonymous names: val, temp, field
Experimental Manipulation

arith + arith_bit_anon | bit
Result '07 Naming

arith + arith_bit_bool_anon | bit

Same context 76.3 (96,56,58)
Match higher/Not match lower 72.5
Match higher/Match lower 61.5
Not match higher/Not match lower 64.4
Not match higher/Match lower 43.4
Conclusion

Occurrence/performance correlation

Exists for experienced developers
Unexperienced developers?

Use of non-precedence information

Developers associate some words with some operators
Operator/operand spacing?

TODO

Measurements of other language source