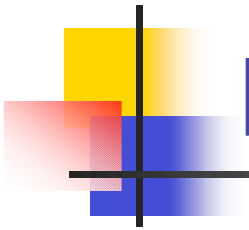
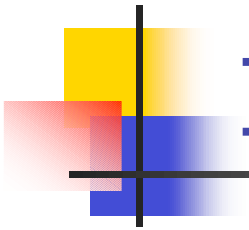
Measuring LOC and other basic measurement

Emanuel Weiss



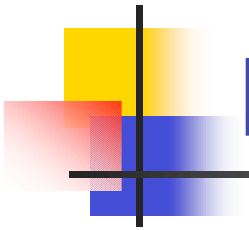
Bibliography

- Conte, S.D.; Dunsmore, H.E.; Shen V.Y.: Software Engineering Metrics and Models, Menlo Park, Ca.: Benjamin/Cummings, 1986
- Jones, T.C.: Estimating Software Costs, New York: McGraw-Hill, 1998
- Stevenson, C.: Software Engineering Productivity, London, etc.: Chapman&Hall, 1995



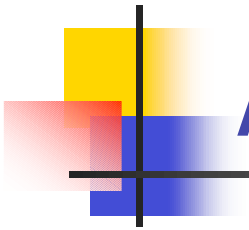
Introduction

- The Goal: Measuring effort
- How do we do this in daily life?
- The Method: Measuring size
- Advantages:
  - Easy to do
  - Size as basic measurement for a number of estimation methods
  - We generally measure productivity by size



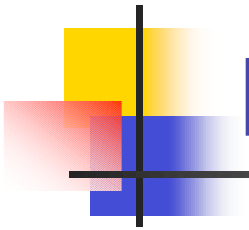
Lines of Code (LOC)

- Line of Code = Effort
- Not every LOC needs the same effort to write
- Notations:
  - Ss: 100‘000 LOC
  - S: 100 KLOC
- Is this sufficient as a definition?



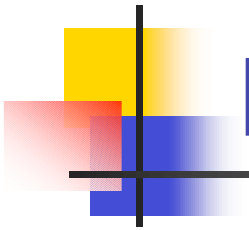
Aspects of a definition

- What do we count as LOC?
  - Documentation
  - Non-executable code
  - Non-delivered code
  - Number of statements per LOC
- Influence of coding style or program language on the number of LOC's



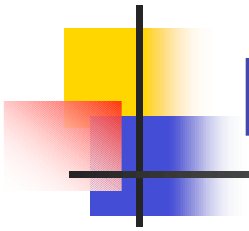
Definition of LOC

- A line of code is any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line. This specifically includes all lines containing program headers, declarations, and executable and non-executable statements.



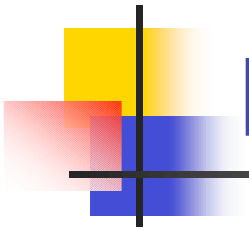
Disadvantages / Problems (1)

- Productivity and ...
  - Documentation
  - Experience
  - Efficient code
  - Languages
- ...whereas productivity is defined as:
  - Output quantity / period of time



Productivity and Language

	Macro assembly	Ada 83	C++
Source code required, LOC	10‘000	3‘500	2‘500
Total project, months	18	10	7.5
Total project, LOC/staff months	555	350	333
Coding, % total project	28%	15%	13%
Total project, LOC/staff months of coding	2000	2300	2500

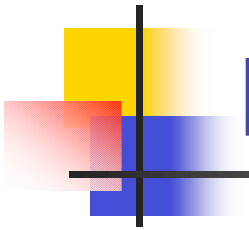


Disadvantages / Problems (2)

- Modularisation
  - By functions: + 25% LOC
  - By data type: + 53% LOC
  - Super-Modularisation: + 73% LOC
- Strongly typed languages
- Reused Code

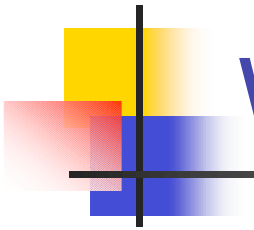
Reused code

- $S_e = f (S_n , S_u)$
- Two models, how f could be defined:
  - $S_e = S_n + a/100\ S_u$
    with: a = 0.4(DM) + 0.3(CM) + 0.3(IM)
    - DM = Design Modulation
    - CM = Code Modulation
    - IM = Integration Modulation
  - $S_e = S_n + S_u^{\,k}$, $k<1$, $k \approx 6/7$



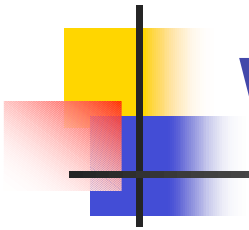
Disadvantages / Problems (3)

- Complexity, reliability and quality
- Technological aids
- Non-programming activities
- Different measuring methods
- No prediction possible



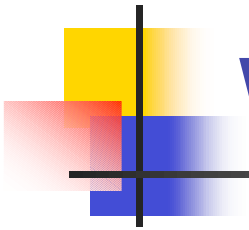
We use LOC because ... (1)

- ... it is one of the most widely used technique in cost estimation.
- ... it is the basic metric underlying to several cost estimation models by Boehm.
- ... due to the widespread use, it allows a simple comparison to data from many other projects, the historical ones included.
- ... the alternative methods to the counting of LOC are also fighting with problems and weaknesses.
- ... it is an easy method to measure effort.



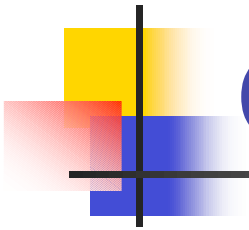
We use LOC because ... (2)

- ... in spite of its unreliability for individual programs, it gives reliable average results, which is crucial especially for huge projects.
- ... it is also reliable in small projects when we quantify the method.
- ... the discrepancies caused by including or excluding JCL, administration or clerical work are usually small.
- ... it considers in fact the higher productivity of high-level languages.



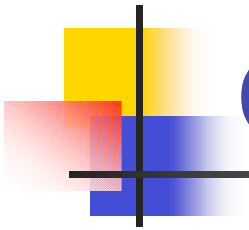
We use LOC because ... (3)

- ... we can avoid lengthy code by organizing reviews, increase the pressure of work or introduce adjustment factors for especially verbose programmers.
- ... cost estimation models like COCOMO, which are based on lines of code, show a close agreement between predicted and actual effort.
- ... there is a strong correlation between lines of code and effort.
- ... there is also a strong correlation between lines of code and alternative metrics.



Qualifying LOC

- "LOC is not a good productivity measure because it penalizes high-level languages: Assembler programmers produce five statements to a COBOL programmer's one. But you should not compare COBOL to Assembler: they are as different as night and day. If you compare COBOL programs only to other COBOL programs, and PL/1 to PL/1, then LOC provides a stable comparison tool." (Arthur)



Counting tokens (1)

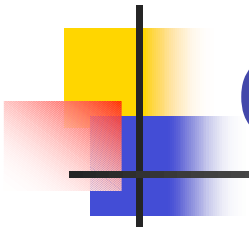
- From Halstead's Software Science

- N_1 = Number of operators
- N_2 = Number of operands
- $N = N_1 + N_2$
  = Number of tokens

Counting tokens (2)

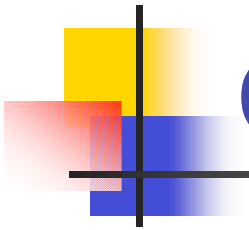
- η_1 = Number of different operators
- η_2 = Number of different operands
- η = $\eta_1 + \eta_2$
  = Vocabulary of a Program

- Volume V = N x log2 η
- S = N/c (c = language-dependent constant)



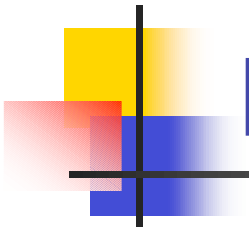
Counting modules

- Modularisation is used in every larger program for reduction of complexity
- Average size is about 100 LOC
- But it varies from 10 to 10‘000 LOC
- Is there a similar measurement but with smaller size variation of the single units?



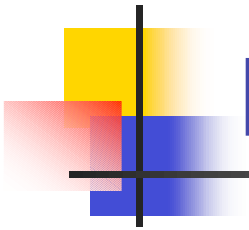
Counting functions

- Same principles as for modules
- An interesting study shows
  - A large number of students had the task to write a program with the same defined functionality.
  - The resulting programs were similar in their number of functions, but not at all similar in their number of modules.
- Independent from LOC
- Near to the Function Point Method



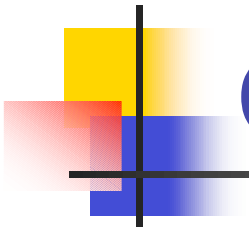
Metrics by Boehm

- a combination of parameters such as the number of routines, operators, operands, files or master files and inputs/outputs.
- the number of variables of the program
- the amount of documentation
- the number of paragraphs in the requirements specifications
- the number of structure-chart in the software design specifications
- the of lines of documentation written in a program design language
- a subjective estimate of difficulty
- the number of machine instructions



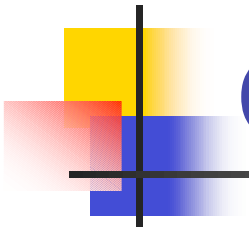
Metrics by Jones

- Based on non-coding activities:
  - the number of pages for the description of software design
  - the number of tests
  - the number of repairs
- Jones himself states:
  - "The same practical counting difficulties which hamper use of line of code also hamper taking measures of operators and operands."



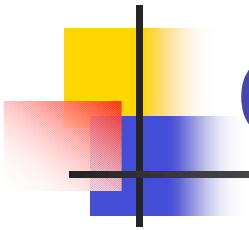
Cost and Effort

- We need figures about cost and effort
  - for valuating estimators
  - for managing projects
- It is important to know how data will be used in the future before collecting them
- Are they coming from small or large projects?
- What are the difficulties in collecting data?



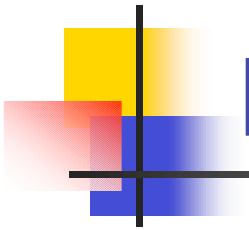
Collecting data in small projects

- Counting hours or days
- Precision is important:
  - Interruption in work
  - Intentional bias
  - Non-programming activities
  - Working time vs. education time
- For testing single factors of a project



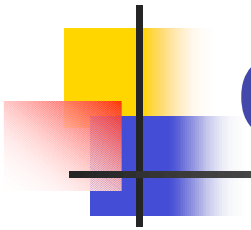
Collecting data in large projects

- Counting person-months or person-years
- Things to consider:
  - Four or five hours work in an eight-hour day
  - Overestimation / Underestimation
  - Use tools to collect data, not paper forms
  - Do not change the measuring process
  - Do use data for both, project management and performance evaluation



Productivity rates

- Cost and effort measures have always to be relevant for management goals
- Advantages of productivity rates:
  - It provides information, which tools and techniques are the best, why they are and under what circumstances.
  - It improves the precision of deadline prediction.
  - It improves the productivity of programmers.



Conclusion

- Before using LOC we need to make precise definitions.
- Considering a lot of alternatives, LOC is still the widest used metric.
- Measuring effort and productivity has a lot of advantages, but has to be done carefully.