

SOFTWARE QUALITY

CPTS 583

Software Product Quality Metrics and Measurement (I)

-- *Size metrics*

Outline

- Product quality: scope
- Size metrics
 - LOC (lines of code)
 - FP (function point)
- LOC
 - Problems
- FP
 - Pros / Cons
 - Basic computation
 - Extensions

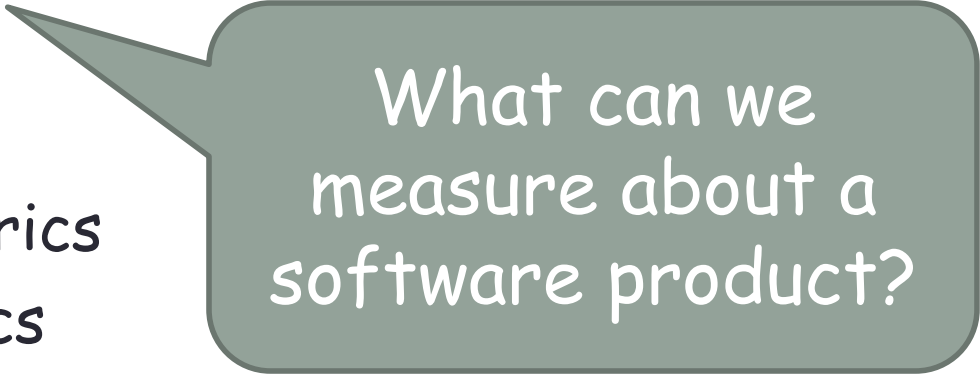
Scope of product quality

- Software Product: deliverables
 - Requirements/design specifications
 - Work products of analysis and design
 - Programs and tests
 - Work product of software construction
 - Documents
 - Work product of various phases in software process
- Product quality
 - **Quality of Programs**
 - Quality of other work products



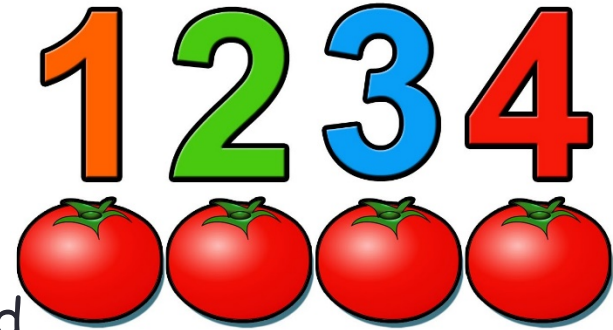
Scope of product quality metrics

- Size metrics
- Customer metrics
- Cost/Time metrics
- Complexity metrics
- Quality attribute metrics
- Design/analysis metrics



What can we
measure about a
software product?

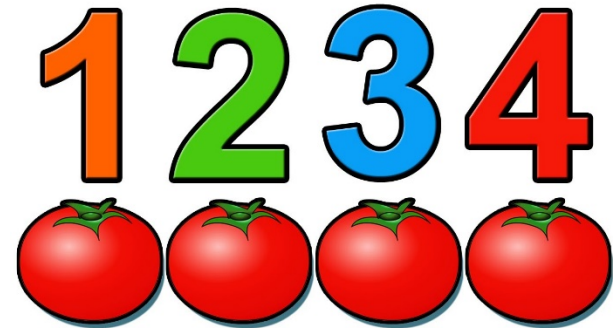
Size/count-based metrics



- Number and type of defects
 - found during requirements, design, code, and test inspections
- Total number of bugs found as a result of unit testing
- Total number of bugs found as a result of integration testing
- Total number of bugs found as a result of validation testing

Size/count-based metrics

- Number of pages of documentation delivered
- Number of new source lines of code created
- Number of source lines of code delivered
- Average size of modules
- Total number of modules
- Productivity, as measured by lines of code (**LOC**) per person-hour



Software Size/Volume

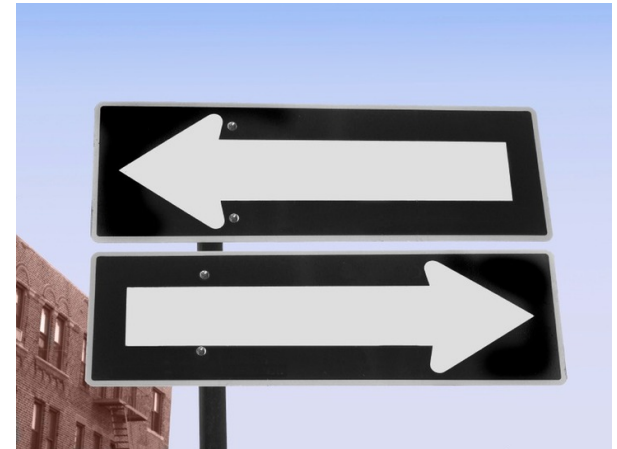
- **KLOC** — classic metric that measures the size of software by thousands of code lines.
 - Software measured in lines of code (**LOC**)
 - As systems grew larger **KLOC** (thousands of lines of code) was also used
- **Number of function points (NFP)** — a measure of the development resources (human resources) required to develop a program, based on the functionality specified for the software system.
- Knowing the size of a system was important for **comparing** different systems together

Lines of Code (LOC)

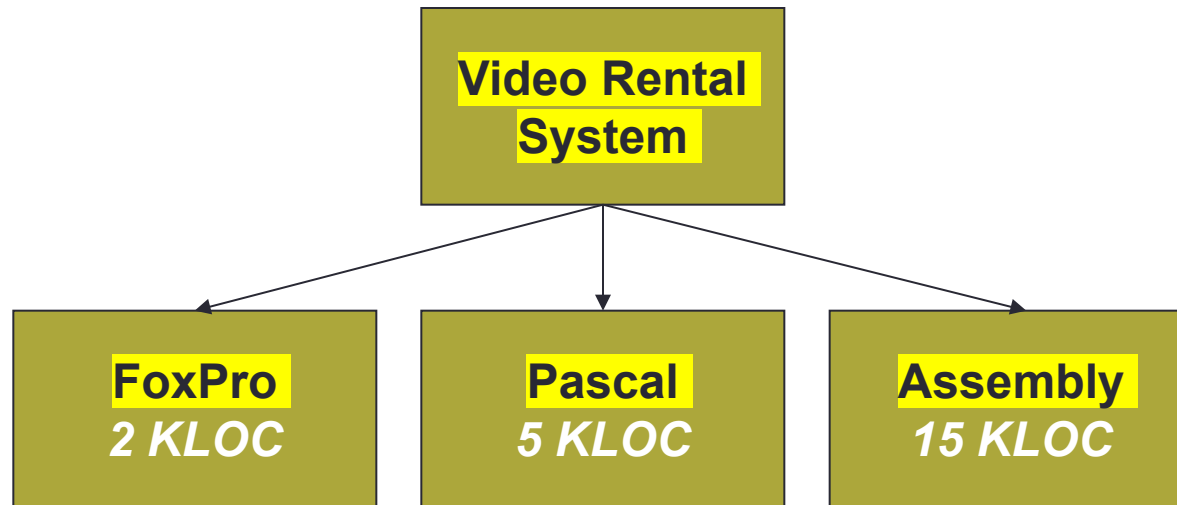
Lines of Code as a measure of system size?

- Easy to measure; but *not well-defined* for modern languages
 - *What's a line of code?*
 - *How do you count them?*
 - *Ignore blanks?*
 - *Count unique lines only?*
 - *Skip headers?*
 - *Multiple statements on a single line?*
 - *.....*

Ambiguity

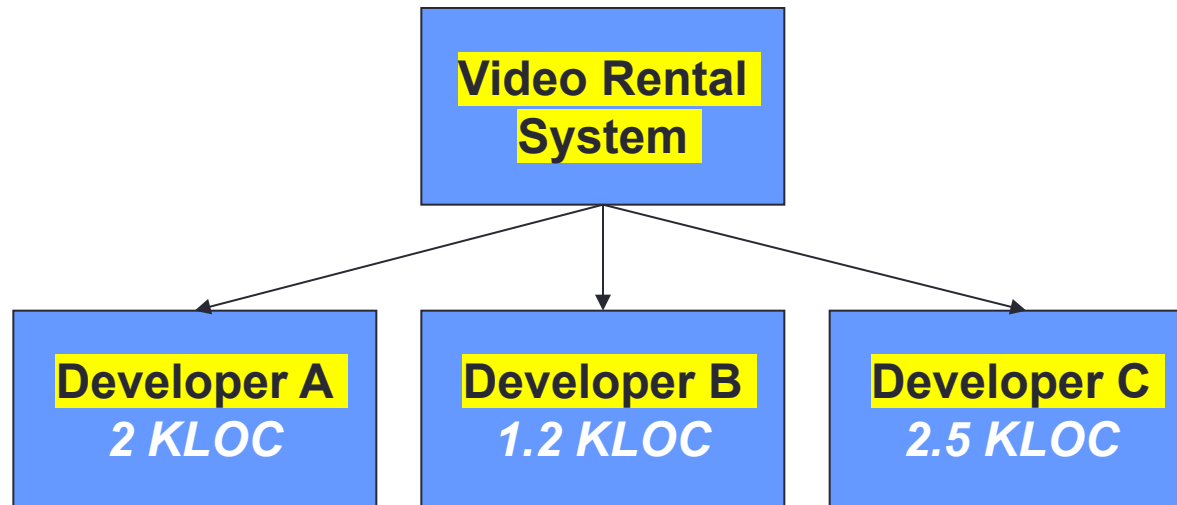


Problems with LOC



- Same system developed with different programming languages will give different LOC readings

Problems with LOC



- Same system developed by different developers using the same language will give different LOC readings

Lines of Code

Lines of Code as a measure of productivity?

- A *poor indicator of productivity*
 - Ignores software reuse, code duplication, benefits of redesign
 - The lower level the language, the more productive the programmer!
 - The more verbose the programmer, the higher the productivity!



Problems with LOC

- To calculate LOC you have to wait until the system is implemented
- This is not adequate when management requires prediction of cost and effort
- A different approach is sometimes necessary...

Function Points

Function Points (Albrecht, 1979)

- Measuring 'how many functionalities'
- Addressing limitations of LOC

EI	External input
EO	External output
LIF	Logical internal file
EIF	External interface file
EIN	External inquiries

- Instead of measuring size, function points measure the **functionality** offered by a system.
- Still use today:
<http://www.ifpug.org>

$$\#EI_S * W1 + \#EO_S * W2 + \#LIF_S * W3 + \#EIF_S * W4 + \#EIN_S * W5 = \#FP_S \text{ (NFP)}$$

Function Points

- **Function** - a collection of executable statements



- Can be calculated before a system is developed (after design)
- Language and developer independent
- Can be used to estimate LOC



- May vary wildly in relation to LOC
- Very subjective
- Cannot be counted automatically

Why Opt for FP Measures?

- Taking into account the “information domain” of the problem
- Not “penalizing” inventive implementations of fewer LOC
- Accommodating reuse and object-oriented approaches
- Good for typical Information Systems applications (interaction complexity)
- Accommodating real-time and scientific software (algorithm and state transition complexity)

Function Points

- The *simplest* way to calculate a function point count is calculated as follows:

$$\begin{aligned} &(\text{No. of external inputs} \times 4) + \\ &(\text{No. of external outputs} \times 5) + \\ &(\text{No. of logical internal files} \times 10) + \\ &(\text{No. of external interface files} \times 7) + \\ &(\text{No. of external enquiries} \times 4) = \text{NFP} \end{aligned}$$

Computing NFP: Examples

Consider the following system specs:

Develop a system which allows customers to report bugs in a product. These reports will be stored in a file and developers will receive a daily report with new bugs which they need to solve. Customers will also receive a daily status report for bugs which they submitted. Management can query the system for a summary info of particular months.

1

External Inputs

1

Logical Internal Files

2

External Outputs

1

External Inquiries

Computing NFP: simple example

External Inputs: 1

External Outputs: 2

Logical Internal Files: 1

External Interface Files: 0

External Enquiries: 1

Total Functionality is $(1 \times 4) + (2 \times 5) + (1 \times 10) + (0 \times 7)$
 $+ (1 \times 4) = 28$

Computing NFP: general steps

Analyze information domain of the application and develop counts

Establish count for input domain and system interfaces

Weight each count by assessing complexity

Assign level of complexity (simple, average, complex) or weight to each count

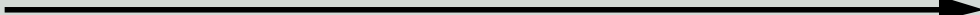

Assess the influence of global factors that affect the application

Grade significance of external factors, F_i , such as reuse, concurrency, OS, ...

Compute function points

$FP = SUM(count \times weight) \times C$ where
complexity multiplier $C = (0.65 + 0.01 \times N)$
degree of influence $N = SUM(F_i)$

Analyzing the Information Domain

<u>measurement parameter</u>	<u>count</u>		<u>weighting factor</u>				
			<u>simple</u>	<u>avg.</u>	<u>complex</u>		
number of user inputs	<input type="text"/>	X	3	4	6	=	<input type="text"/>
number of user outputs	<input type="text"/>	X	4	5	7	=	<input type="text"/>
number of user inquiries	<input type="text"/>	X	3	4	6	=	<input type="text"/>
number of files	<input type="text"/>	X	7	10	15	=	<input type="text"/>
number of ext.interfaces	<input type="text"/>	X	5	7	10	=	<input type="text"/>
count-total							<input type="text"/>
complexity multiplier							<input type="text"/>
function points							<input type="text"/>

Taking Complexity into Account

- Complexity Adjustment Values (F_i) are rated on a scale of 0 (not important) to 5 (very important):
 1. Does the system require reliable backup and recovery?
 2. Are data communications required?
 3. Are there distributed processing functions?
 4. Is performance critical?
 5. System to be run in an existing, heavily utilized environment?
 6. Does the system require on-line data entry?
 7. On-line entry requires input over multiple screens or operations?

Taking Complexity into Account

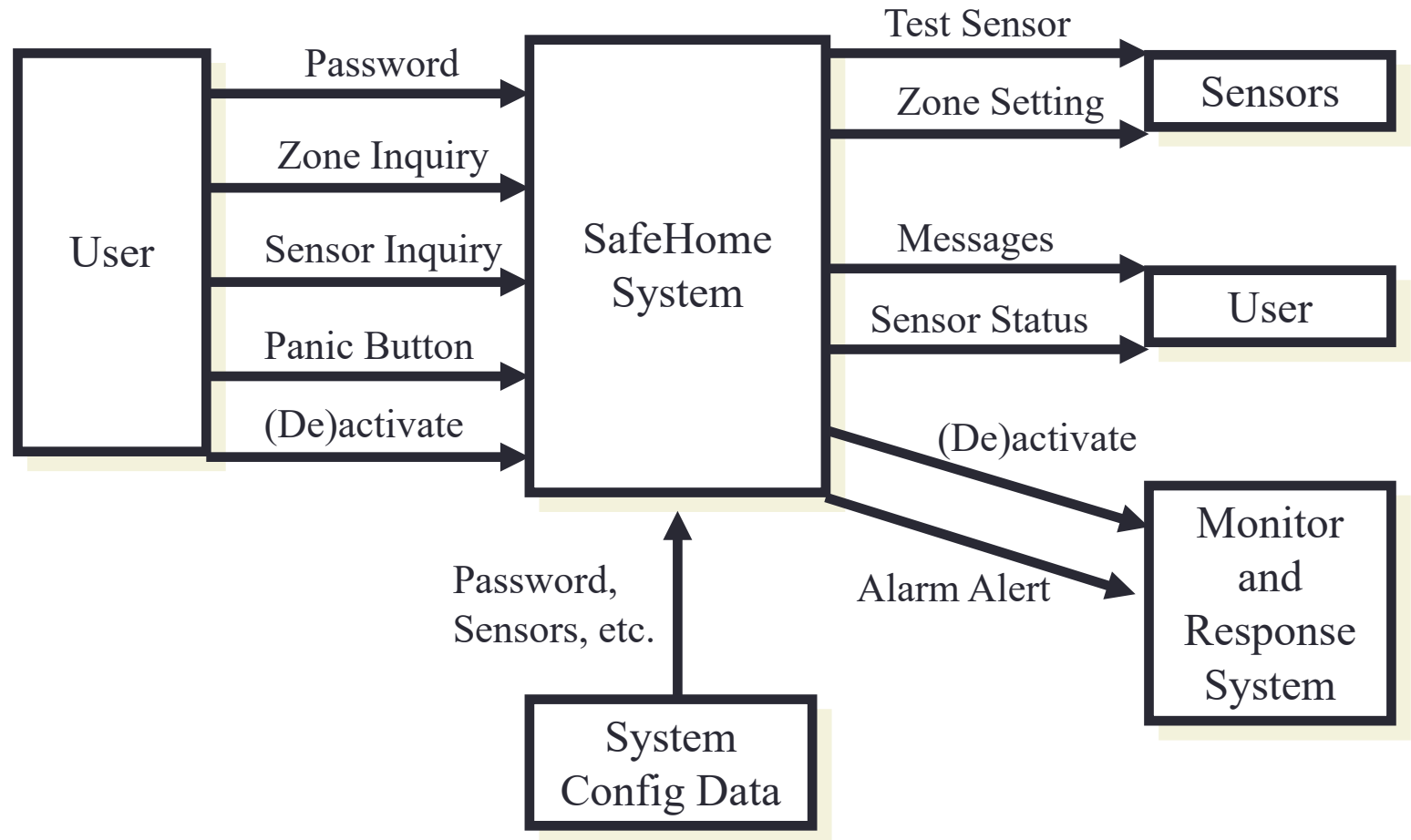
- Complexity Adjustment Values (F_i) are rated on a scale of 0 (not important) to 5 (very important):
 8. Are the master files updated on-line?
 9. Are the inputs, outputs, files, or inquiries complex?
 10. Is the internal processing complex?
 11. Is the code designed to be reusable?
 12. Are conversion and instillation included in the design?
 13. Multiple installations in different organizations?
 14. Is the application designed to facilitate change and ease-of-use?

Exercise: Function Points

- Compute the function point value for a project with the following information domain characteristics:
 - Number of user inputs: 32
 - Number of user outputs: 60
 - Number of user enquiries: 24
 - Number of files: 8
 - Number of external interfaces: 2
 - Assume that weights are average and external complexity adjustment values are not important.
- Answer:

$$(32 \times 4 + 60 \times 5 + 24 \times 4 + 8 \times 10 + 2 \times 7) \times 0.65 = 401.7$$

Example: SafeHome Functionality



Example: SafeHome FP Calc

<u>measurement parameter</u>	<u>count</u>	<u>weighting factor</u>				
			<u>simple</u>	<u>avg.</u>	<u>complex</u>	
number of user inputs	3	X	3	4	6	= 9
number of user outputs	2	X	4	5	7	= 8
number of user inquiries	2	X	3	4	6	= 6
number of files	1	X	7	10	15	= 7
number of ext.interfaces	4	X	5	7	10	= 20
count-total	—————→					50
complexity multiplier	$[0.65 + 0.01 \times \sum F_i] = [0.65 + 0.46]$					1.11
function points	—————→					55.5

Typical Normalized Metrics

Project	LOC	FP	Effort (P/M)	R(000)	Pp. doc	Errors	Defects	People
alpha	12100	189	24	168	365	134	29	3
beta	27200	388	62	440	1224	321	86	5
gamma	20200	631	43	314	1050	256	64	6

- **Size-Oriented:**
 - errors per KLOC, defects per KLOC, R per LOC, page of documentation per KLOC, errors / person-month, LOC per person-month, R / page of documentation
- **Function-Oriented:**
 - errors per FP, defects per FP, R per FP, pages of documentation per FP, FP per person-month

Function Point Extensions

- The original function points were sufficient but various people extended them to make them **more expressive for particular domains**.
- Examples
 - General System Characteristics (**GSC**) Extension
 - 3D Function Points for real time systems
 - Object Points
 - Feature Points

The GSC Function Points Extension

- **Reasoning:** Original Function Points do not address **certain functionality** which systems can offer
 - E.g. Distributed functionality, performance optimisation, etc
- The GSC extension involves **answering 14 questions** about the system and modifying the original function point count accordingly

The GSC Function Points Extension

1. Data communications
2. Distributed Functions
3. Performance
4. Heavily used configuration
5. Transaction rate
6. Online Data Entry
7. End-user Efficiency
8. On-line update
9. Complex Processing
10. Reusability
11. Installation ease
12. Operational Ease
13. Multiple sites
14. Facilitation of Change

The GSC Function Points Extension

- The analyst/software engineer assigns a value between 0 and 5 to each question
- *0 = not applicable and 5 = essential*
- The Value-Adjustment Factor (VAF) is then calculated as:

$$VAF = 0.65 + 0.01 \sum_{i=1}^{14} Ci$$

You then adjust the original function point count as follows:

$$GSC\ FP = FP * VAF$$

The GSC Function Points Extension

Consider the bug-reporting system for which we already looked at and suppose the analyst involved answers the GSC questions as follows...

- | | | | |
|-------------------------------|---|----------------------------|---|
| 1. Data communications | 5 | 8. On-line update | 3 |
| 2. Distributed Functions | 0 | 9. Complex Processing | 1 |
| 3. Performance | 1 | 10. Reusability | 0 |
| 4. Heavily used configuration | 0 | 11. Installation ease | 2 |
| 5. Transaction rate | 1 | 12. Operational Ease | 3 |
| 6. Online Data Entry | 5 | 13. Multiple sites | 4 |
| 7. End-user Efficiency | 0 | 14. Facilitation of Change | 0 |

Total GSC Score = 25

GSC FP = 28 * (0.65+0.01*25) = 25.2

Summary

- The scope software product quality: size, cost, defects, time, quality attributes
- Size metrics: LOC and FP
- LOC: ambiguous definition of a line, poor indicator of productivity
- FP: advantages over LOC, computation steps
 - Analyze information domain
 - External inputs/outputs, internal files, external interface, inquiries
 - Determine the weights
 - Compute the complexity multiplier
 - Obtain the NFP