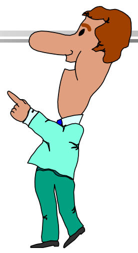
Guidelines



- ❐ Sources
- ❐ Implemented applications
- ❐ Planned applications
- ❐ Hints

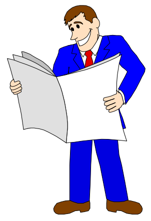
Counting the Function Points of existing systems is not always as simple as it may appear on first glance. Certainly all details of the application are available for inspection. In practice, however, the flood of implementation details may obscure the generic functions of a system, as required by the user. There is the temptation to count the implementation idiosyncrasies, rather than the underlying functions requested by the user. On the other side the system may be poorly documented or, even worse, the documentation may be misleading or incomplete.

The most serious error in counting Function Points of existing systems, is to overlook functions which are not obvious, or which are not fully documented. Detection of any omissions will require practical experience in the operation of the business area. Typical candidates of such omissions are functions which are used infrequently or performed by only a small and restricted group of the user community. Sometimes Function Points can be missed because the application boundary is unclear or misunderstood. Omission of functionality is potentially even more likely at the early stages of development when functions are only partly or vaguely defined.

Another serious error is the count of implementation functionality not requested by the user but included in the system for technical reasons.

In this session we will look at both the various forms of documentation and development stages, provide guidelines and offer hints to minimize the counting errors.

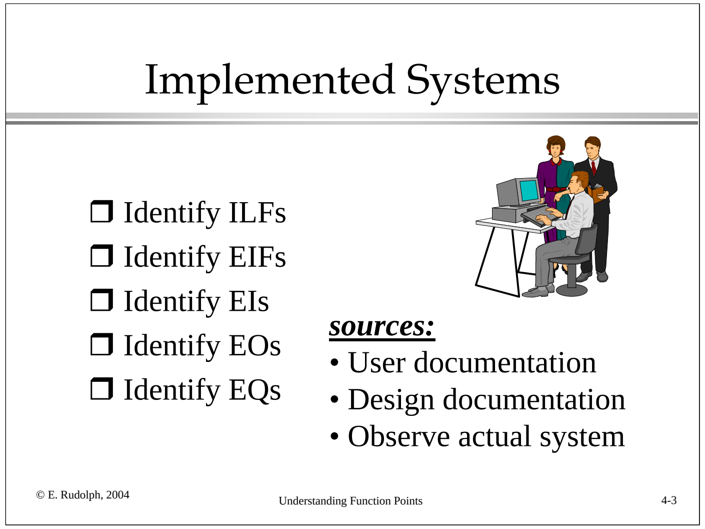
Sources



- ❐ User documentation
  - ➢ User Manual
  - ➢ Requirements
- ❐ Design documentation
  - ➢ Data Model
  - ➢ Context Diagram
- ❐ Observation of application
- ❐ ***Definitively not source code***

The Function Point count of implemented systems should be straight forward and it can be carried out in a short period of time. A team of two should be able to count up to 3000 Function Points during an eight hours working day. This, however, would imply that the documentation is readily available. Otherwise the counting speed can be reduced significantly. Counting speed is also substantially reduced when exactly all DETs and referenced files have to be recorded for each function. Correct classification of functional complexity, however, does not require such a level of detail. An EI with more than 3 FTRs and more than 5 DETs will be high in complexity. The fact that it may have 10 FTRs and 15 DETs is irrelevant for that functions point value. It may be argued, that the details are needed should that function be enhanced at a later stage. A better policy would be to only count as much detail as required and therefore count the details when in deed an enhancement is done.

The documentation relevant for the Function Point count will include the requirement specification, the design documentation and most importantly the user documentation. These documents provide the high level of documentation which will identify user functions clearly. Basic design documentation (particularly the data model) is primarily needed to identify the internal file types. Any detail documentation, particularly program listings, can be hard to use. For one they will be voluminous, forcing the Function Point counter to sift through much unnecessary detail. At this level of documentation the focus is on technical implementation and therefore the line between user function and implementation technique can be obscure. - Some check should be made to ensure that all the documented functionality has indeed been implemented in the version of the system to be counted and hasn't been deferred to future releases. - The implemented system may contain functions not requested in the requirements, or complexities which cannot be supported by the documented user requests. In such instances the system implementer may claim that the additional functionality was necessary, and wanted by the user (or to be more precise, the user did not oppose the additions). The issue can often be settled by checking if the user will pay for the increased application size. When the documentation is suspicious (or lacking) function point counts could also be made by inspecting all system functions together with experienced users of that system.



Implemented Systems

- ❒ Identify ILFs
- ❒ Identify EIFs
- ❒ Identify EIs
- ❒ Identify EOs
- ❒ Identify EQs

sources:

- User documentation
- Design documentation
- Observe actual system

© E. Rudolph, 2004 Understanding Function Points 4-3

For existing systems it will be effective and practical to start the Function Point count with the file types and derive the remaining function types thereafter. The following steps are recommended:

• Identify and classify all internal logical files.

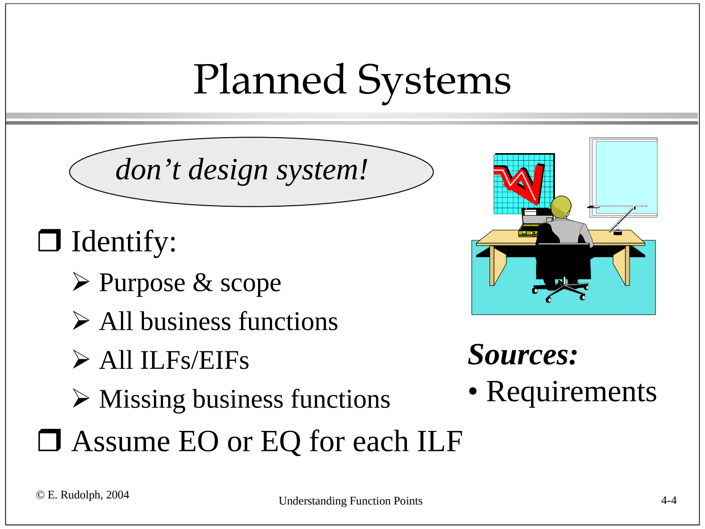
• Identify and classify all external interface files.

• Expect for each internal logical file at least two (add, delete) or three (add, change, delete) external input types. When the input types for a counted file are dubious, then assume at least two external input types.

• Identify and classify all external output types. Assume for each internal logical file at least one external output (listing) when not even a part of the file is shown in any other external output.

• Identify and classify all external inquiry types. Do not assume an external inquiry type for each internal file unless the inquiry function can be found in the systems requirements.

• Distinguish between internal input, output, and inquiry (which all are not counted), and external input, output, and inquiry which are included in the count. Use Overview Figure when in doubt if a function is internal or external.



Planned Systems

don't design system!

- ❒ Identify:
  - ➢ Purpose & scope
  - ➢ All business functions
  - ➢ All ILFs/EIFs
  - ➢ Missing business functions
- ❒ Assume EO or EQ for each ILF

Sources:

- Requirements

© E. Rudolph, 2004 Understanding Function Points 4-4

At the early stage, when most functions are not fully identified, there is a considerable risk of error (mostly in underestimating the size). It is the strength of the Function Point technique that it is based on overall requirements and not on implementation details. The important point is that a function has to be identified in principle, but not in all detail. IT staff often focus too early on detail and overlook the significance of requirements (which are regarded as cloudy). In fact sound requirements should identify all functionality or as one FP counter put it:

If you cannot count Function Points from a requirement definition then you do not have a requirement definition. At the requirement stage the data analysis is not yet completed, probably it has not even formally started. Most entities or interfaces are not established and can only be assumed. Yet typically the overall flow of data and information is visualized and often described in a context diagram.

There is no hard rule on how to best proceed at the early project stage with an Function point count. **A common mistake, however, is to attempt to completely design the information system during the measurement process**. Time constraints will prevent this. The Function Point measurement will have to be completed in hours or days, a formal design would require weeks or months.

Requirement specifications typically focus on the outputs of the information system. Not the output layouts, but the type of information required by the users. It then is the task of the designer to devise an information system which provides these information functions and identify necessary prerequisites or data resources.

The Orr/Warnier technique is based on such an approach and could serve as guidance. It identifies the required outputs, establishes the associated logical files to provide such outputs, and the defines the inputs, necessary to supply the data for those logical files not yet in existence:

1. Identify all outputs and inquiries required to be supplied by the planned information system. Together with a user, identify for a few examples the detail levels of key information required. This will identify EOs, EQs, and some EIs.

2. Sketch the files / interfaces necessary to provide the output, or to store the inputs, identified in the first step. Most organizations already have a data model which will give a good indication of the size of data elements to consider.

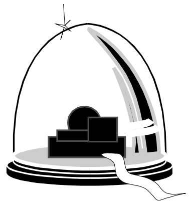
3. Add all inputs not yet identified in step 1, but necessary to maintain the files identified in the second step. Expect on average 2 - 3 input types per logical file. This identifies most EIs.

4. Ensure that there exists at least one output or inquiry to view the content of a logical file identified during step 2.

When the individual business transactions cannot be assessed expect as a very general rule of thumb 30FP per business entity.

Automated Count

- ❒ Case Tools
  - IEF
  - IEW
- ❒ Advantages
  - Consistent
  - Fast
- ❒ Disadvantages
  - No personal involvement
  - User view and purpose dubious
  - Only counts CASE implementation



Some tools (often in connection with CASE tools such as IEW or IEF) will automate the function point count. Such automation can reduce the cost and increase the consistency of a Function Point count. There are, however, some aspects which may favour a manual count.

The main disadvantage stems from the fact that the user view may be lost. Typically the tools will be based on IT representations of the system (program design or program code). Particularly when using the program source code as input such tools tend to count the technical implementation. Under such circumstances counting tools will find it difficult to distinguish between program parts based on functional requirements or technical implementation.

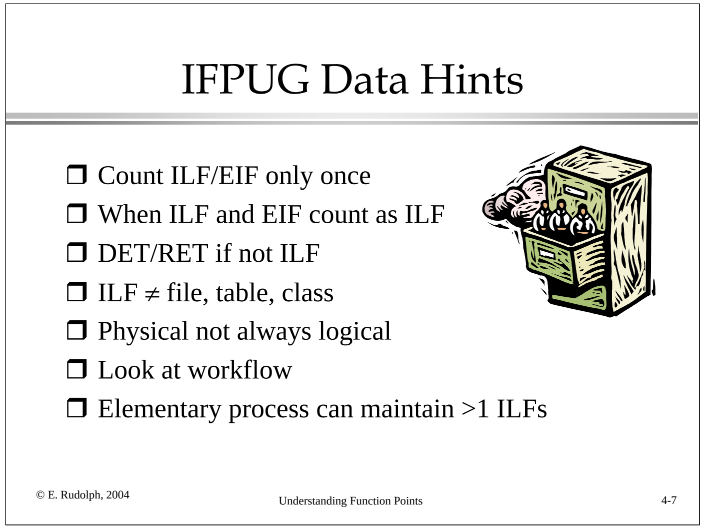
The risk of counting the “programmers system” (rather than the users system) will be reduced when the tools use high level design documentation or business requirements as input. Upper CASE or workflow environments will have such information in computer accessible form. Still, such high level tools are restricted to applications using the specific notation of the CASE or workflow environment.

Although automation will be reduce the counting process from a few hours to a few minutes or even seconds there are other reasons to have the development team count manually the application, preferably during the early stage of the project. Being involved in the count the team will gain a better understanding of the application and will also be committed to the count.

Hints

- ❒ Insist on user view
- ❒ Concentrate on ***what***, not ***how***
- ❒ Two counters (user & IT) are best
- ❒ More are unproductive
- ❒ Focus on function type identification
- ❒ Default complexity: average
- ❒ Expect ~3 EIs and one EO per ILF

1. Whenever possible use two persons to measure a system, one of which should be familiar with the business area of the information system.
2. It is more important to identify a function than to correctly classify it. When uncertain, use an average classification level.
3. When the individual Degrees of Influence cannot be established, assume an adjustment factor of 1.0 for an on-line data base application and 0.8 for a batch application
4. Expect three external input types, one external output type, and one inquiry type for each internal logical file. However, positively identify these function types, before counting them.
5. Oversight of functions is a major cause of error. Documentation is not necessarily complete. Expect and probe for overlooked functions, but only accept those which were requested by the user.
6. When in doubt if a function has been requested by the user, find out if the user is willing to pay for it.
7. When a vital function (such as backup and recovery of a volatile internal logical file) was not specified in the requirement document, bring the matter to the attention of the user. Such a vital function should be included in the count (and the resulting system) unless the user specifically requests its omission.



IFPUG Data Hints

- ❒ Count ILF/EIF only once
- ❒ When ILF and EIF count as ILF
- ❒ DET/RET if not ILF
- ❒ ILF ≠ file, table, class
- ❒ Physical not always logical
- ❒ Look at workflow
- ❒ Elementary process can maintain >1 ILFs

© E. Rudolph, 2004 Understanding Function Points 4-7

Note that the IFPUG hints are not rules and should not be taken as IFPUG rules.

– An application can use an ILF/EIF in multiple processes, but the ILF or EIF is counted only once.

– A logical file cannot be counted as both an ILF and EIF. If it satisfies both rules, count as an ILF.

– If a group of data was not counted as an ILF/EIF itself, count its data elements as DETs for the ILF or EIF, which includes that group of data.

– Do not assume that one physical file, table or object class equals one logical file when viewing data logically from the user perspective.

– Although some storage technologies such as tables in a relational DBMS or sequential flat file or object classes relate closely to ILFs or EIFs, do not assume that there is always a one-to-one physical-logical relationship.

– Do not assume all physical files must be counted or included as part of an ILF or EIF.

– Look at the workflow.

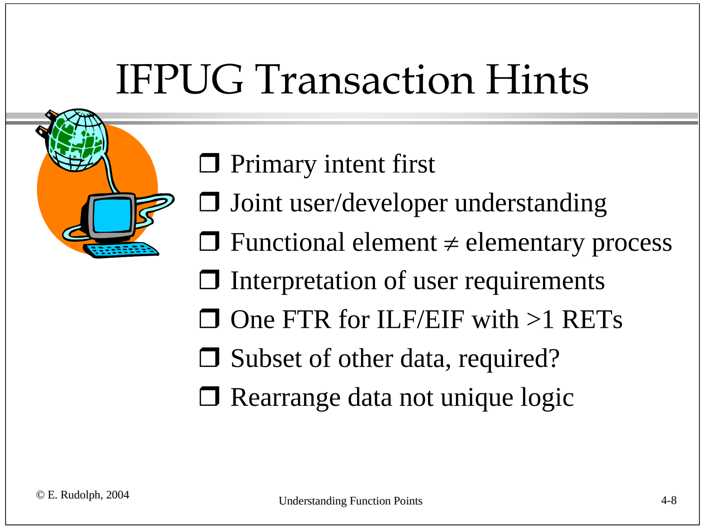
– In the process functional decomposition, identify where interfaces occur with the user and other applications.

– Work through the process diagram to get hints.

– Credit ILFs maintained by more than one application to each application at the time the application is counted. Only the DETs being used by each application being counted should be used to size the ILF/EIF.

– An elementary process can maintain more than one ILF.

– Work through the process diagram to get hints.



IFPUG Transaction Hints

- ❒ Primary intent first
- ❒ Joint user/developer understanding
- ❒ Functional element ≠ elementary process
- ❒ Interpretation of user requirements
- ❒ One FTR for ILF/EIF with >1 RETs
- ❒ Subset of other data, required?
- ❒ Rearrange data not unique logic

© E. Rudolph, 2004 Understanding Function Points 4-8

• Identify the primary intent of the elementary process before classifying it as an EI, EO, or EQ.

• Identification of the elementary process(es) is based on a joint understanding or interpretation of the requirements between the user and the developers.

• Each element in a functional decomposition may not map to a unique elementary process.

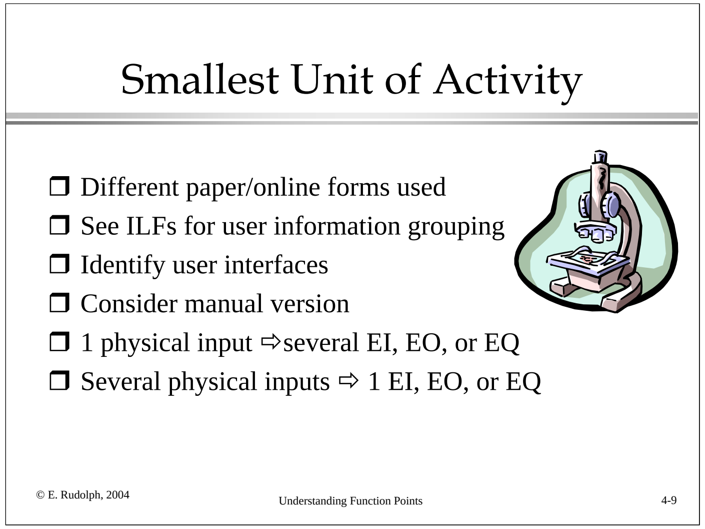
• The identification of the elementary processes requires interpretation of the user requirements.

• Count only one FTR for each ILF/EIF referenced even if the ILF/EIF has multiple RETs.

• If the data elements appear to be a subset of the data elements of another EI, EO, or EQ, be sure two elementary processes are required by the user—one for the main data elements and one for the subsets.

• Remember that sorting or rearranging a set of data does not make processing logic unique.

Note that the IFPUG hints are not rules but just guidance. When there appears to be a conflict with IFPUG rules then IFPUG rules should be applied.



Smallest Unit of Activity

- ❒ Different paper/online forms used
- ❒ See ILFs for user information grouping
- ❒ Identify user interfaces
- ❒ Consider manual version
- ❒ 1 physical input ⇨several EI, EO, or EQ
- ❒ Several physical inputs ⇨ 1 EI, EO, or EQ

© E. Rudolph, 2004 Understanding Function Points 4-9

The perceived granularity of the function can lead to difference in counting Function Points. The notion of elementary process or smallest unit of activity can help to avoid to count a sub-activity as elementary function (or to count several elementary functions as a single function. In order to establish if a function is the smallest unit of user activity:

– Look at the different paper or on-line forms used.

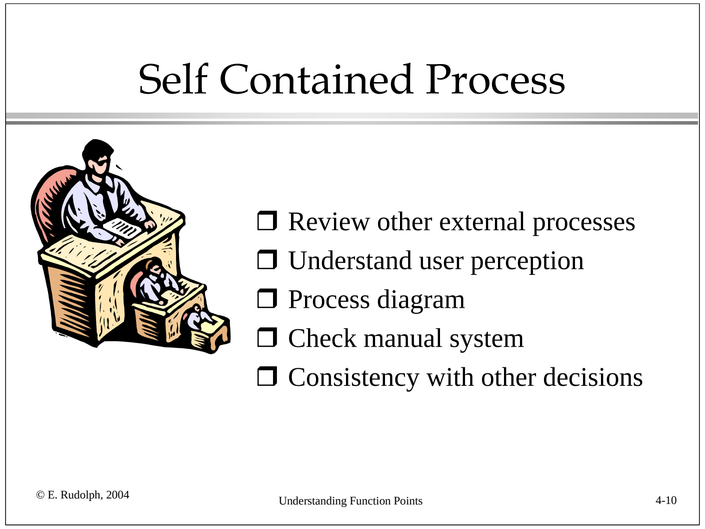
– Review the ILFs to identify how the user groups the information.

– Identify where the user and other application interfaces occur in the process functional decomposition.

– Look at what happened in the manual system.

– Note that one physical input or transaction file or screen can, when viewed logically, correspond to a number of EIs, EOs or EQs.

– Note that two or more physical input or transaction files or screens can correspond to one EI, EO or EQ if the processing logic is identical.



Self Contained Process

- ❒ Review other external processes
- ❒ Understand user perception
- ❒ Process diagram
- ❒ Check manual system
- ❒ Consistency with other decisions

© E. Rudolph, 2004 Understanding Function Points 4-10

– Review other external inputs, external outputs and external inquiries to understand how the user works with the information.

– Work through the process diagram to get hints.

– Look at what happened in the manual system.

– Check for consistency with other decisions.