2. What Information Should Be Provided in Computer System Documentation?

Text for Chapter 2

[PaMa95] Parnas, D. L. and Madey, J. *Functional documents for computer systems*. Sci. Comput. Programming **25**(1), 41–61 (Oct. 1995).

Structure of the requirements documentation and software documentation.

Additional Background for Chapter 2

[PaCl86] Parnas, D. L. and Clements, P. C. *A rational design process: how and why to fake it*. IEEE Trans. Softw. Eng. **12**(2), 251–257 (Feb. 1986).

Structure of the documentation vs. structure of the development process.

Overview of Documents

- system requirements document
- system design document
- software requirements document
- software behaviour specification
- software module guide
- module interface specification
- uses-relation document
- module internal design document

- communication: service specification document
- communication: protocol design document

Specification Form vs. Specification Content

- this overview: concerned with content only
- formalism must be adapted to situation
- choice of some formalism alone does not guarantee completeness of content!
  - “formal” vs. “rigorous”

The System Requirements Document

description of:

- environmental quantities of concern
- association of env. quantities to math. variables
- relationships between values of these due to environmental constraints (NAT)
- relationships between values of these due to new system (REQ)

- descriptions are black-box
- details: see Chapter 1.1

Structure of the System Requirements Document

required sections:

- environmental quantities
- environmental constraints
- system behaviour
- dictionary
  - definitions of:
    - ▷ math. functions and relations
    - ▷ words that are not common natural language
    - ▷ words that have special meaning in application domain

optional sections:

- system overview
  - informal
  - possibly including non-behavioural requirements
- notational conventions
  - if non-standard notation used
  - variable naming
  - special variable mark-up
  - . . .
- anticipated changes
  - important to reduce effort for later changes
  - see also Chapter 5

The System Design Document

- introduces input and output variables

description of:

- relationships between monitored and input variables (IN)
- relationships between output and controlled vars. (OUT)
- relationships between input and output variables (SOF) (software requirements)
  - in separate document, see below
- details: see Chapter 1.2

The Software Requirements Document

- software requirements (SOFREQ) implicitly determined by
  - system requirements document
  - system design document

  } = software requirements doc.

  (NAT, REQ, IN, OUT)
- usually design step:
  explicit, more deterministic
  software behaviour specification (SOF)
- details: see Chapter 1.2

The Software Behaviour Specification

- SOF
- details: see Chapter 1.2
- particularly important for
  multi-processor / multi-computer / network systems
  - allocation of tasks to individual computers
  - hierarchy of software behaviour specifications

Software Modules

Definition 12 (Module)
A module is a programming work assignment.

- (see other definitions of "module" later in lecture)
- assume information hiding principle was used (see below)
- black-box description of module's behaviour

The Software Module Guide

- division of of software into modules
- states responsibilities of each module
- informal "guide"
  - rigorous module interface specification necessary to start implementation
- details: see Chapter 3.2 later in lecture

The Module Interface Specification

- each module implements one or more finite state machines (FSMs)
  - FSMs also called *objects* or *variables*
- description of module interface is black-box description of these objects
  - every "*program*" (= method/function/. . .) belongs to exactly one module
  - programs use objects created by other modules as components of their data structure

Writing Module Interface Specifications

- similar to documenting software requirements
- simplifications possible
  - many software modules are entirely internal
    - ▷ no environmental quantities
    - ▷ all communication through external invocation of the module's programs
  - state set finite
  - state transitions can be treated as discrete events
  - often: real-time can be neglected, only the sequence of events matters
    - ▷ replace time-functions by traces

- details: see Chapter 4 later in lecture

Formalisms for Module Interface Specifications

- "Trace Assertion Method" proposed by Parnas *et.al.* was never used much
- many other formalisms known and in use:
  - CSP } see lecture Safety-Critical Systems 3
  - Z (/ Object-Z) } see lecture Safety-Critical Systems 3
  - SDL
  - StateCharts
  - ...

  advantages/disadvantages depend on application domain

The Uses-Relation Document

- range and domain of "uses" relation: subsets of set of access-programs of the modules
  - (P, Q) in relation if program P uses program Q
- document often is a binary matrix
- constrains work of programmers
- determines viable subsets of the software
- for details, see Chapter 3.4 later in lecture

The Module Internal Design Document

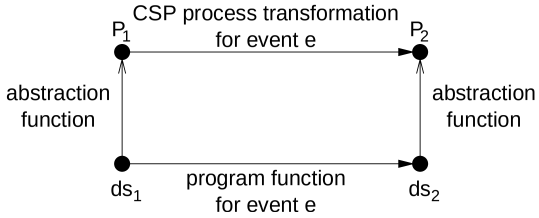
- for each module
- describe module's data structure
- state intended interpretation of data structure (in terms of external interface)
- specify effect of each access-program on data structure
- "clear-box description"
- sufficiently precise to verify the workability of the design (together with module interface specification)

Information in the Module Internal Design Document

1. complete description of data structure
   (may include objects implemented by other modules)
2. abstraction function
   from values of objects
   to descriptions in terms of external program calls
3. program function:
   an LD relation specifying each program as a
   mapping from states before to states after execution

Abstraction Function

for deterministic programs; using CSP:



- if design correct, then diagram commutes for all events
- if program non-deterministic, program funct. is LD relation

Programs

Definition 13 (Program)
A program is a text describing a set of state sequences in a digital (finite state) machine.

- Each state sequence is called an execution of the program.

Documenting the Effect of Individual Programs

- execution
  - starting state
  - final state (if finite)
  - or infinite sequence
- intermediate states often not interesting, only:
  - termination possible? } LD relation
  - termination guaranteed? } LD relation
  - if termination possible, then in which final states? } LD relation
- if with parameters, then functions from parameters to programs

LD Relation

→ blackboard. . .

Documenting by LD Relations

- for specification of program
- for actual behaviour of program
- notations:
  many, depending on application area
  - "displays" proposed by Parnas *et.al.*
    were never used much

Communication: The Service Specification Document

- communication system often implemented as a hierarchy of services
- each level can be viewed as a module
- black-box behaviour of a module = service specification

Communication: The Protocol Design Document

- implementation = protocol design
  - using lower-level services
  - using local data structures
- is a kind of internal module design document