Producing Precise Complete Software Reference Documentaion: An Introduction

Summary

The problem of producing software reference documentation, similar in purpose to the documents produced by civil engineers has not been taken seriously by Computer Scientists.

This lecture introduces the way one can use traditional mathematics to define the required content of a document and how a design can be checked on the basis of such a document.

The Role of Documents in Engineering

Engineers Design Through Documentation

Documents record key design decisions

- to guide the builders
- to assist in inspection
- to assist in maintenance
- to enable review

Documents are binding on everyone

Documents are carefully controlled

Documents are precise documents that use mathematics w

Documents are not introductions or tutorials

Documents are not extracted comments (javadoc)

The 40 Year "Software Crisis"

The three primary causes

- Lack of disciplined design, careful choice of interfaces and careful structural decisions
- Lack of thorough, review, inspection and testing
- Lack of accurate, complete, precise documentation

Other problems (bugs, security flaws, cost of change, poor team cooperation, • • •) are all secondary.

We must attack the cause, not just the symptoms.

Documentation is the key to all three.

Why Documentation is the Key

- One must document decisions precisely and systematically so that they can be reviewed for accuracy and completeness.
- Good Documentation takes the guesswork out of programming
- Hierarchical Documentation is the key to inspection.
- Precise documentation can be used to generate test cases and evaluate results.

What Do We Mean by "Documentation"

Practical tool, not a theoretical achievement.

- **Authoritative repository of information**
- **Usable as a quick and reliable reference by developers, reviewers, maintainers, users.**
- **Easier to use for information retrieval than the code**
- **Quicker and more authoritative than trial executions**

Computer Science and Software Documentation

Engineers bemoaned inability to specify software.

Programmer and Actuary, Tony Hoare, responded by writing "A specification of an Operating System" containing bits of program - no statement of requirements for that program.

Computer Scientists have regarded specification as meaning programming in a useless language since then.

They regard documentation as an essay or commentary.

If you take the word "Engineer" in Software Engineer" seriously, you must ignore such people.

The Importance of Documentation in Europe Today

Microsoft is being fined Millions per day because they have not documented the interfaces to their servers.

The EC wants a document sufficient to allow a competitor to build a compatible competitive server.

Microsoft does not seem to be able to do this.

Desperate, they claimed that source code is the most precise and authoritative documentation possible

(see http://www.microsoft.com/presspass/press/2006/jan06/01-25EUSourceCodePR.mspx)

At best code tells you what it does, not what it should do.

At worst, you can't read it accurately.

Requirements for Software Documents

- Accuracy
- Precision
- Consistency
- Completeness
- Ease of reference.

The first two can be achieved by using mathematics.

The last three require improved notation and ease of reference.

All require content definitions.

Basic Documentation Guidelines

Never mix reference documents with introductions.

Do not rely on words; they will never be precise enough.

Mathematics is the only way to be precise, but

- **expressions must be simple and easily parsed**
- **Interpretation should be direct (closed form).**

Only relevant information should be included and this should be stated directly, not implied.

Each item of information should be in only one place and everyone should know where it will be put/found .

These are all easier said than done.

The Roles of Documents

- A *description* states properties of a product; it may include a mixture of incidental and required properties.
- A *specification* is a description that states only required properties.
- A *full specification* is a specification that states all required properties.

The same notation may be used for all 3.

These classifications are a matter of intent not notation.

There is no such thing as a “specification language”.

Is Mathematical Documentation just "Formal Methods"?

Our goal is to organize information for easy retrieval.

Engineering style of mathematics (closed form solutions)

Not axiomatic in style. Users calculate, not derive.

No new mathematics needed - classical concepts suffice.

Not intended for automatic programming (but possible).

Proof is not the main goal (but possible).

Documents are neither models nor programs.

Content Definitions: Why and How

Organizations often specify the format of a document.

They rarely specify the content.

We need a content definition to check for completeness.

Each document is a description of some mathematical relations.

Define the contents by defining the relations.

There are many ways to describe a relation.

If you have specified a relation, the documentation must specify whether any pair is contained or not.

Each document has a different range and domain

The Main Documents

(1) “System Requirements Document”: Treats computer system as “black-box”.

(2) “System Design Document”: Describes computers and communication.

(3) “Software Requirements Document” [(1) + (2) or (1) - (2)].

(4) “Software Function Specification”: Describes actual software behavior.

(5) “Software Module Guide”: How to find your module. (informal document).

(6) “Module Interface Specifications”: Treat each module as black-box.

(7) “Uses Relation Document”: Range and domain are sets of programs.

(8) “Module Internal Design Documents”: data structure, abstraction relation, and program functions.

Example of a Functional Definition of Document Content

Deterministic Program effect described by a function from data-states to data-states.

(x,y) in that function if a program started in state x will terminate in state. Function may be partial.

- **Function can be described by precondition/postcondition pair.**
- **Function can be described by Dijkstra's "wp" (weakest precondition).**
- **Function can be described by predicates (Hehner).**
- **Function can be described by concurrent-assignment statement (Mills).**

Real need is to deal with non-deterministic programs.

Notational Conventions:

If R is a relation,

(1) R refers to the set of ordered pairs that constitutes the relation,

(2) R (x,y) is a predicate, the characteristic predicate of the set R. R(x,y) is *true* if and only if (x,y) is in R.

(3) If R is a function, R(x) refers to y such that R(x,y)

How can we document system requirements? (I)

Identify monitored variables ($m_1, m_2, \cdots, m_N$).

Identify controlled variables ($c_1, c_2, \cdots, c_P$).

For each scalar variable, x, denote the time-function describing its value by "x^t".

The value of x at time t is denoted "$x^t(t)$".

The vector of time-functions ($v^t_1, v^t_2, \ldots, v^t_n$) will be denoted by "$\underline{v}^t$".

How Can We Document System Requirements (II)?

Describe the following relations:

Relation NAT

- **domain contains values of m^t,**
- **range contains values of c^t,**
- **(m^t, c^t) is in NAT if and only if nature permits that behaviour.**

Relation REQ

- **domain contains values of m^t,**
- **range contains values of c^t,**
- **(m^t, c^t) is in REQ if and only if system should permit that behaviour.**

How can we document system requirements?

It must be true that,

- **(1) domain(REQ) ⊇ domain(NAT).**
- **The relation REQ can be considered *feasible with respect to NAT* if (1) holds and,**
- **(2) domain (REQ∩NAT) = (domain(REQ) ∩ domain(NAT)).**

How can we document system design?

i^t denotes the vector valued time function $(i^t_1, i^t_2, \bullet\bullet\bullet, i^t_r)$
one element for each of the input registers
o^t denotes the vector valued time function $(o^t_1, o^t_2, \bullet\bullet\bullet, o^t_q)$
one element for each of the output registers

Document the following relations

Relation IN

domain contains values of $\underline{m}^t$
range contains values of $\underline{i}^t$
$(\underline{m}^t, \underline{i}^t)$ is in IN if and only if input device permits that behaviour

It must be the case that

(1) domain(IN) $\supseteq$ domain(NAT)

Relation OUT

domain contains the possible values of $\underline{o}^t$
range contains the possible values of $\underline{c}^t$
$(\underline{o}^t, \underline{c}^t)$ is in OUT if and only if output device permits that behaviour

How can we document software requirements?

Software requirements = system design + system requirements

(1) $REQ(\underline{m}^t, \underline{c}^t)$

(2) $IN(\underline{m}^t, \underline{i}^t)$

(3) $OUT(\underline{o}^t, \underline{c}^t)$ and

(4) $NAT(\underline{m}^t, \underline{c}^t)$

The actual software can be described by

(5) $SOF(\underline{i}^t, \underline{o}^t)$

For the software to be acceptable, SOF must satisfy:

(6) $\forall \underline{m}^t \forall \underline{i}^t \forall \underline{o}^t \forall \underline{c}^t [IN(\underline{m}^t, \underline{i}^t) \wedge SOF(\underline{i}^t, \underline{o}^t) \wedge OUT(\underline{o}^t, \underline{c}^t) \wedge NAT(\underline{m}^t, \underline{c}^t) \rightarrow REQ(\underline{m}^t, \underline{c}^t)]$

Using functional notation:

(6a) $\forall \underline{m}^t [\underline{m}^t \in domain(NAT) \rightarrow (REQ(\underline{m}^t) = OUT(SOF(IN(\underline{m}^t))))]$

How can we document black-box module interfaces?

Module: group of programs given as a work assignment.

Essential that the interface be specified precisely.

Information Hiding Modules have private data structure, no shared data.

Time-functions can be replaced by discrete sequences (traces).

Output variables contain information for outside world.

THIS IS DISCUSSED IN A LATER LECTURE.

How can we document the effect of individual programs?

Definitions:

Program denotes a text that may be executed

Program denotes a text describing a set of state sequences (executions)

- sequence may be finite (termination)
- sequence may be infinite (nontermination)

Program function (relation)

Domain: set of starting states (first states of finite sequences)

Range: set of final states (last states of finite sequences)

(s,t) in function if there exists a finite execution <s,...,t>

Competence set (for LD-relation)

s is in the competence set if all executions starting with s are finite

If the competence set is exactly the domain of the relation, it may be omitted

How can we document internal module design?

Module = private data structure + set of access programs

Design, usually in designer's head, must be written down

Three essential elements:

(1) description of the data structure

(2) abstraction relation

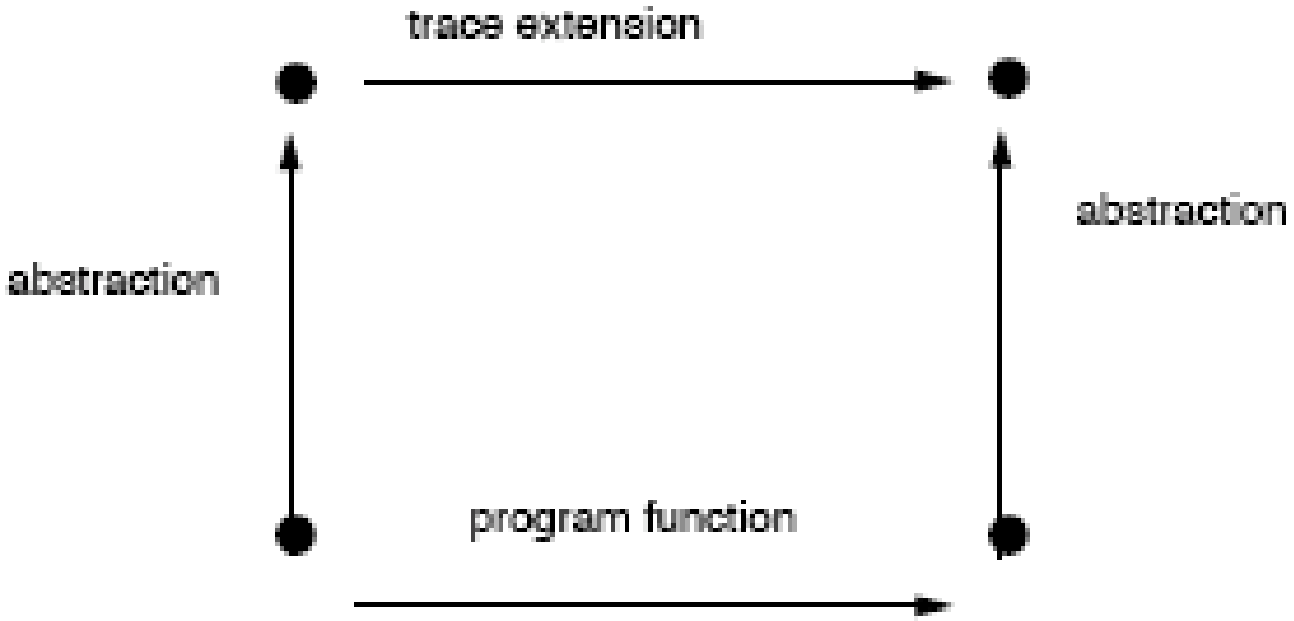
- **Domain: pairs (object name, data state)**
- **Range: canonical traces**
- **((d, n), tr) is in the relation if tr, for the object named n, could produce data state, d**

(3) program function (LD-relation)

- **one for each possible invocation of each access program**

How can the workability of a design be verified?

For all possible invocations, the following must hold



Summary

In Software Engineering we can and should take documentation as seriously as they do in other Engineering Disciplines.

We can define the contents of documents

We can check the feasibility of a design before implementing.

We use traditional mathematics

No new “specification language” is needed, but we can improve the notation used for mathematical expressions.

Future Lectures will look at examples and notation.