Hints for Computer System Design[1]

Butler W. Lampson

Computer Science Laboratory
Xerox Palo Alto Research Center
Palo Alto, CA 94304

Abstract

Studying the design and implementation of a number of computer has led to some general hints for system design. They are described here and illustrated by many examples, ranging from hardware such as the Alto and the Dorado to application programs such as Bravo and Star.

1. Introduction

Designing a computer system is very different from designing an algorithm:

> The external interface (that is, the requirement) is less precisely defined, more complex, and more subject to change.
>
> The system has much more internal structure, and hence many internal interfaces.
>
> The measure of success is much less clear.

The designer usually finds himself floundering in a sea of possibilities, unclear about how one choice will limit his freedom to make other choices, or affect the size and performance of the entire system. There probably isn't a 'best' way to build the system, or even any major part of it; much more important is to avoid choosing a terrible way, and to have clear division of responsibilities among the parts.

I have designed and built a number of computer systems, some that worked and some that didn't. I have also used and studied many other systems, both successful and unsuccessful. From this experience come some general hints for designing successful systems. I claim no originality for them; most are part of the folk wisdom of experienced designers. Nonetheless, even the expert often forgets, and after the second system [6] comes the fourth one.

Disclaimer: These are not
novel (with a few exceptions),
foolproof recipes,
laws of system design or operation,
precisely formulated,
consistent,
always appropriate,
approved by all the leading experts, or
guaranteed to work.

---

[1] This paper was originally presented at the. 9th ACM Symposium on Operating Systems Principles and appeared in *Operating Systems Review* **15**, 5, Oct. 1983, p 33-48. The present version is slightly revised.

They are just hints. Some are quite general and vague; others are specific techniques which are more widely applicable than many people know. Both the hints and the illustrative examples are necessarily oversimplified. Many are controversial.

I have tried to avoid exhortations to modularity, methodologies for top-down, bottom-up, or iterative design, techniques for data abstraction, and other schemes that have already been widely disseminated. Sometimes I have pointed out pitfalls in the reckless application of popular methods for system design.

The hints are illustrated by a number of examples, mostly drawn from systems I have worked on. They range from hardware such as the Ethernet local area network and the Alto and Dorado personal computers, through operating systems such as the SDS 940 and the Alto operating system and programming systems such as Lisp and Mesa, to application programs such as the Bravo editor and the Star office system and network servers such as the Dover printer and the Grapevine mail system. I have tried to avoid the most obvious examples in favor of others which show unexpected uses for some well-known methods. There are references for nearly all the specific examples but for only a few of the ideas; many of these are part of the folklore, and it would take a lot of work to track down their multiple sources.

> *And these few precepts in thy memory*
> *Look thou character.*

It seemed appropriate to decorate a guide to the doubtful process of system design with quotations from *Hamlet*. Unless otherwise indicated, they are taken from Polonius' advice to Laertes (I iii 58-82). Some quotations are from other sources, as noted. Each one is intended to apply to the text which follows it.

Each hint is summarized by a slogan that when properly interpreted reveals the essence of the hint. Figure 1 organizes the slogans along two axes:

Why it helps in making a good system: with functionality (does it work?), speed (is it fast enough?), or fault-tolerance (does it keep working?).

Where in the system design it helps: in ensuring completeness, in choosing interfaces, or in devising implementations.

Fat lines connect repetitions of the same slogan, and thin lines connect related slogans.

The body of the paper is in three sections, according to the *why* headings: functionality (section 2), speed (section 3), and fault-tolerance (section 4).

2. Functionality

The most important hints, and the vaguest, have to do with obtaining the right functionality from a system, that is, with getting it to do the things you want it to do. Most of these hints depend on the notion of an *interface* that separates an *implementation* of some abstraction from the *clients* who use the abstraction. The interface between two programs consists of the set of assumptions that each programmer needs to make about the other program in order to demonstrate the correctness of his program (paraphrased from [5]). Defining interfaces is the most important part of system design. Usually it is also the most difficult, since the interface design must satisfy three conflicting requirements: an interface should be simple, it should be complete, and it should

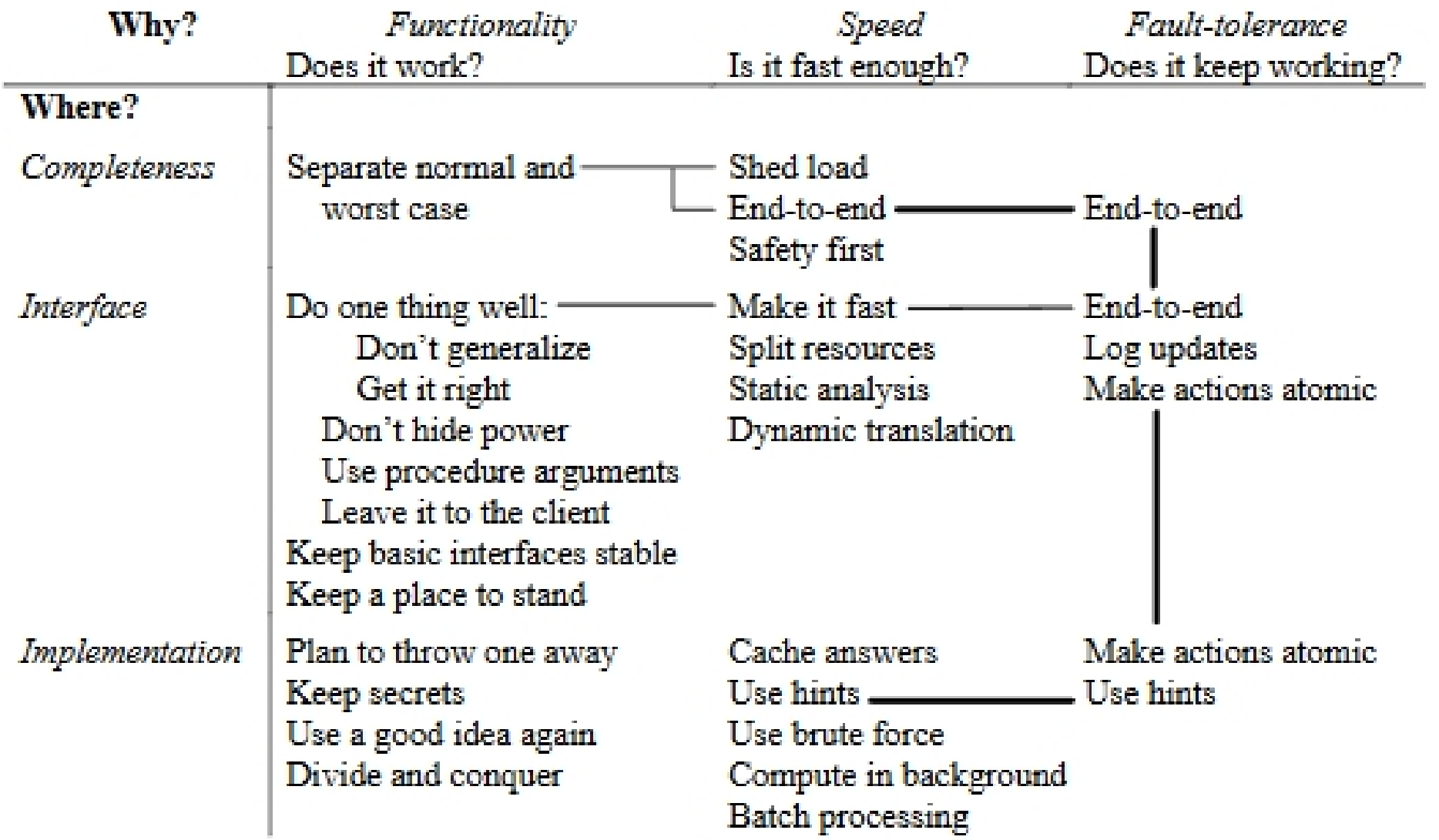

Figure 1: Summary of the slogans

admit a sufficiently small and fast implementation. Alas, all too often the assumptions embodied in an interface turn out to be misconceptions instead. Parnas' classic paper [38] and a more recent one on device interfaces [5] offer excellent practical advice on this subject.

The main reason interfaces are difficult to design is that each interface is a small programming language: it defines a set of objects and the operations that can be used to manipulate the objects. Concrete syntax is not an issue, but every other aspect of programming language design is present. Hoare's hints on language design [19] can thus be read as a supplement to this paper.

2.1 Keep it simple

> *Perfection is reached not when there is no longer anything to add,*
> *but when there is no longer anything to take away.* (A. Saint-Exupery)

> *Those friends thou hast, and their adoption tried,*
> *Grapple them unto thy soul with hoops of steel;*
> *But do not dull thy palm with entertainment*
> *Of each new-hatch'd unfledg'd comrade.*

• *Do one thing at a time, and do it well.* An interface should capture the *minimum* essentials of an abstraction. *Don't generalize*; generalizations are generally wrong.