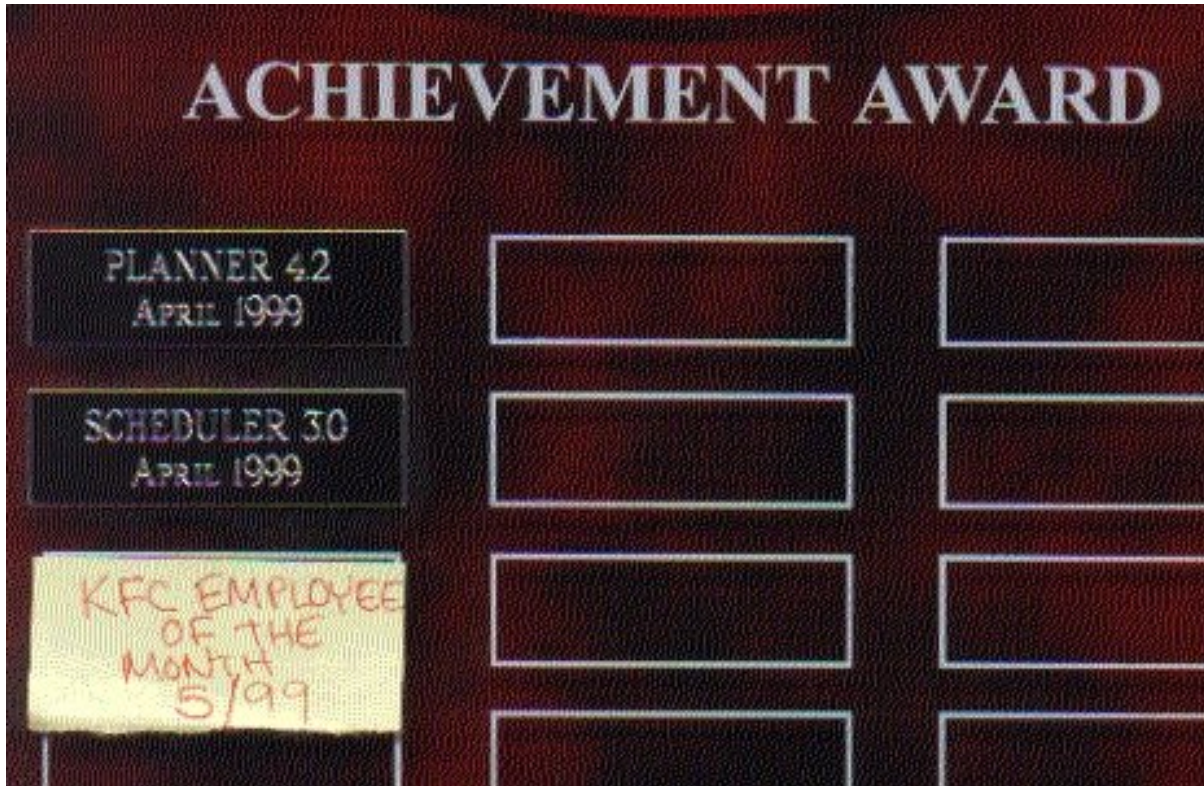


Adventures In Software Development

Table of contents

1 Introduction.....	2
2 Recommended Reading.....	3

1. Introduction



[KFC](#)

After writing a postmortem article for a video game project, I considered documenting lessons learned from various diverse projects, ranging from flashy video games, shrink-wrapped consumer software, boring but useful factory automation systems, and wireless web portals that are supposed to run 24/7.

But after more than a dozen software releases, it's like watching a syndicated TV show over and over - I know how each episode is going to end. So rather than repetitiously list the same lessons, I've partitioned my thoughts according to whatever themes grab me. (It's really a form of therapy)

Alas, the number of articles has grown unfinished like an out-of-control software project, as I "refactor" larger articles into smaller ones. So this isn't a blog - rather, like software, it's a continual work-in-progress. I hope you can find useful nuggets of reason in them.

Thanks to everyone who's taken to the time to read these essays. The numerous emails I've

received with feedback, suggestions and corrections are welcome and immensely helpful. Code safely!

2. Recommended Reading

2.1. Engineering



[Ken, Jr.](#)

I built the *Kenputer* as part of [6.004](#), one of many [MIT](#) courses with material now available on the [Open Courseware Project](#). The text for [6.001](#), [Structure and Interpretation of Computer Programs](#), demonstrates programming can be an esthetic and intellectual endeavor. [Abstraction and Specification in Program Development](#) is no longer used for [6.170](#) (and [CLU](#) was never that useful), but the material was my first introduction to useful data abstraction and documentation of function and module interfaces.

[Henry Petroski](#) writes on engineering design and engineering failures. *The Evolution of Everyday Things* explains how engineering design is more of an evolutionary process than a one-shot design. *To Engineer is Human* examines high-profile engineering failures and their causes.

[Bob Colwell](#) writes on various engineering issues, including amusing anecdotes from his

days as a processor architect at Intel.

For a trip down memory lane, check out [Apple folklore](#), [Mac system notes](#), the [Symbolics Museum](#), and the [Old Computers Museum](#).

The movie [Office Space](#) is required viewing. And of course, there's [Dilbert](#), which is funny until you realize it's all true.

2.2. Software Development



[Hardware Bug](#)

If you're going to read one book on software development, read [Steve McConnell's](#) *Rapid Development*, a compendium of modern software development practices. [Gordon Bell's](#) *High-Tech Ventures*, although dated, is my favorite how-to guide for startups. Fred Brook's [The Mythical Man-Month](#) is another classic (the title refers to the point that adding unnecessary people on a project is not only useless, but counterproductive). Everyone should expend some thought on the intellectual property issues of software and the commons that benefit us all - read the analyses and explanations by [Lawrence Lessig](#).

See [game development resources](#) for recommended books and web sites on game development.

I was largely inspired to put my own thoughts in writing by the essays on hacker culture by [Eric Raymond](#), thoughts on programmers and program languages by [Paul Graham](#), the personal anecdotes and insights into software design of [Richard Gabriel](#), and the intelligent rants of [Joel on Software](#). [Eric Sink](#) offers tips on one-person software developer outfits ("Micro-ISV"s).

2.3. Best Practices



[Street Racing](#)

The term "best practices" is about as accurate and realistic as "zero-defect software", but there are some useful ideas embedded in all those fads and buzzwords.

[Refactoring techniques](#) are listed at [Refactoring.com](#).

The [Agile Manifesto](#) lists the principles of [agile software development](#) methodologies, such as [extreme programming](#) and [scrum development](#).

[Design patterns](#) are listed at the [Portland Pattern Repository](#). Lest one think that design patterns originated entirely from within the programming community, read architect [Christopher Alexander's](#) pattern language trilogy.

The [Java Coding Style Guidelines](#) are fairly reasonable, yet many feel compelled to mangle it with [Hungarian Notation](#), explained in [Charles Simonyi's original paper](#).

2.4. User Interfaces



[Rat and Mouse](#)

Anyone designing user interfaces should be familiar with graphic design, especially the work of [Paul Rand](#). [Edward Tufte](#) demonstrates how to convey information efficiently and accurately via graphics. [Donald Norman](#) argues against unnecessary complexity in everyday life. [Ask Tog](#) about GUI design. [Jakob Nielsen](#) writes on web usability.

the [Java Look and Feel Design Guidelines](#) are pretty good and Apple literally wrote the book on GUI's with the [Apple Human Interface Guidelines](#).

If you're willing to look beyond standard GUI design, read about interactive storytelling from

[Chris Crawford](#) and graphical narrative from [Scott McCloud](#).