Pragmatic Programmers Summary

Chapter 1-2-3-4

Learning is a continuous and ongoing process.

They think beyond the immediate problem, always trying to place it in its larger context, always trying to be aware of the bigger picture.

The greatest of all weaknesses is the fear of appearing weak.

One of the cornerstones of the pragmatic philosophy is the idea of taking responsibility for yourself and your actions in terms of your career advancement, your project, and your day-to-day work.

We can be proud of our abilities, but we must be honest about our shortcomings—our ignorance as well as our mistakes.

Remember the Big Picture.

Don’t put all your technical eggs in one basket.

Don’t forget the human side of the equation. (Read book invest yourself)

If you can’t find the answer yourself, find out who can. Don’t let it rest.

Communicate is important.

Be a Listener.

Choose a style.

DRY (Don't repeat yourself)

Make It Easy to Reuse

If it isn’t easy, people won’t do it. And if you fail to reuse, you risk duplicating knowledge.

An orthogonal approach reduces the risks inherent in any development.

Avoid global data.

The Singleton pattern in Design Patterns is a way of ensuring that there is only one instance of an object of a particular class.

Be careful with singletons—they can also lead to unnecessary linkage.

Duplicate code is a symptom of structural problems. Have a look at the Strategy pattern in Design Patterns for a better implementation.

An orthogonally designed and implemented system is easier to test. There is always more than one way to implement something.

There Are No Final Decisions Code That Glows in the Dark.

We build software prototypes in the same fashion, and for the same

reasons—to analyze and expose risk, and to offer chances for correction

at a greatly reduced cost.

Architecture

New functionality in an existing system.

Structure or contents of external data.

Third-party tools or components.

Performance issues.

User interface design.

Estimate to Avoid Surprises.

All estimates are based on models of the problem.

The normal rules of estimating can break down in the face of the complexities and vagaries of a sizable application development. We find that often the only way to determine the timetable for a project is by gaining experience on that same project. This needn't be a paradox if you practice incremental development, repeating the following steps.

-Check requirements-

-Analyze risk-

-Design, implement, integrate-

-Validate with the users-

Basic Tools: Shell or GUI, Editor, Source Control, Debugging, Code Generators

Choose One Editor : know it thoroughly, and use it for all editing tasks. If you use a single editor (or set of keybindings) across all text editing activities, you don't have to stop and think to accomplish text manipulation: the necessary keystrokes will be a reflex.

Source Control System: But a source code control system does far more than undo mistakes.

A good SCCS will let you track changes, answering questions such as: Who made changes in this line of code? What's the difference between the current version and last week's? How many lines of code did we change in this release? Which files get changed most often? This kind of information is invaluable for bug-tracking, audit, performance, and quality purposes

Always Use Source Code Control

Fix the problem, Not blame.

You Can't Write Perfect Software. Because perfect software doesn't exist. No one in the brief history of computing has ever written a piece of perfect software. Don't waste your time following the impossible dream.

Inheritance and polymorphism are the cornerstones of object-oriented

languages and an area where contracts can really shine.

One of the benefits of detecting problems as soon as you can is that you can crash earlier. And many times, crashing your program is the best thing you can do.

"There is a luxury in self-reproach. When we blame ourselves we feel no one else has a right to blame us" -Oscar Wilde

Ask yourself, "Will this code still run if I remove all the exception handlers?" If the answer is "no," then maybe exceptions are being used in nonexceptional circumstances.

An error handler is a routine that is called when an error is detected.

Finish What You Start