**The Pragmatic Programmer**

1. A Pragmatic Philosophy
   1. It’s Your Life

What is a better context to implement pragmatic programmer other than “It’s your life”?

It’s your life, your career, what you do in life. You own it. You run it. You create it.

* 1. The Cat Ate My Source Code

Everything is in your control.

Be someone can be relied to, only then you can also rely to someone.

Take responsibility, trust teammates, provide options, don’t make lame excuses.

* 1. Software Entropy

Entropy refers to the amount of disorder in a system. In software engineering, commonly called technical debt. We think we will repair later, but usually we won’t.

Analogy with a rotten building, started with a single broken window, then people psychologically feel that the whole building is not maintained, no one cares, and they begin to break other parts of the building.

* 1. Stone Soup and Boiled Frogs

Stone soup 🡪 the story of soldiers boiling stones first, then villagers become curious and adding ingredients one by one. Make a big change in your team by start making running little thing gradually, then show people so they start to join out of progress curiosity.

Boiled frogs 🡪 story about frog that in a gradually harming environment without noticing. Don’t be this frog. Keep an eye on the big picture, constantly review what is happening.

* 1. Good-Enough Software

Discipline to write software that is good enough for users, for future maintainers, for your own peace of mind.

Good enough 🡪 meet user’s requirement, basic performance, privacy, security standard.

involve your users in the trade off, know when to stop.

* 1. Your Knowledge Portfolio

Like financial portfolio, build your knowledge portfolio as well.

Invest regularly, diversify, buy low-sell high, review and rebalance.

Learn one new language every year, read technical boo each month, participate in community.

Critical thinking, think critically about what you read and hear (5W1H).

* 1. Communicate!

Know your audience, know what you want to say, choose your moment, choose a style, make it look good, involve your audience, be a listener, get back to people.

Make documentation.

1. A Pragmatic Approach
   1. The Essence of Good Design

ETC (Easier To Change). Good design is ETC than bad design. Decoupling by isolation is good because it is ETC. Single Responsibility is good because it is ETC. Naming principle is good to make code easier to read so it is ETC.

* 1. DRY – The Evils of Duplication

DRY (Don’t Repeat Yourself), not only not copying code with same syntax, but the main idea is don’t do duplication of knowledge/intent 🡪 can be in many places: in field data, in obvious function documentation, in a same local code base hold by every developer in a team.

Simple test 🡪 when a part of code has to change, do you have to make that change in multiple places, change code and documentation, change database schema, etc? The it is not DRY.

* 1. Orthogonality

Orthogonal 🡪 every single entity is fully independent/decoupled. Change in one part wont influence other entity 🡪 The main objective of OOP.

Non orthogonal 🡪 Like in helicopter main control, each input of the four buttons influences the helicopter’s stable point.

* 1. Reversibility

Reversibility 🡪 our work is not fixed by an irreversible decision (stick to only solution).

There are no final decisions, use flexible architecture, always ready for changes of stacks.

* 1. Tracer Bullets

Tracer bullets 🡪 Like a laser pointing at the gun’s target. If the tracers are hitting the target, so are the regular bullets. Use the tracer to refine the aim. It’s pragmatic, real-time feedback under actual conditions.

In project 🡪 Build a skeleton of the project first with simple feature for each component: Database, data model, business logic, authorization, and User Interface. Start light while gradually enrich and pivot your project with feedback from user.

* 1. Prototype and Post-it Notes

Prototype 🡪 built for testing some idea. Can be in whiteboard, post-it notes, prototyping tool.

The difference with tracer bullet is prototype is not the real product, ready to dump, while tracer bullet is the skeleton/structure of the real product.

* 1. Domain Languages

Internal vs External languages

* 1. Estimating

1. The Basic Tools
   1. The Power of Plain Text
   2. Shell Games
   3. Power Editing
   4. Version Control
   5. Debugging
   6. Text Manipulation
   7. Engineering Daybooks
2. Pragmatic Paranoia
   1. Design by Contract
   2. Dead Programs Tell No Lies
   3. Assertive Programming
   4. How to Balance Resources
   5. Don’t Outrun Your Headlights
3. Bend, or Break
   1. Decoupling
   2. Juggling the Real World
   3. Transforming Programming
   4. Inheritance Tax
   5. Configuration
4. Concurrency
   1. Breaking Temporal Coupling
   2. Shared State is Incorrect State
   3. Actors and Processes
   4. Blackboards
5. While You Are Coding
   1. Listen to Your Lizard Brain
   2. Programming by Coincidence
   3. Algorithm Speed
   4. Refactoring
   5. Test to Code
   6. Property-Based Testing
   7. Stay Safe Out There
   8. Naming Things
6. Before the Project
   1. The Requirement Pit
   2. Solving Impossible Puzzles
   3. Working Together
   4. The Essence of Agility
7. Pragmatic Project
   1. Pragmatic Teams
   2. Coconuts Don’t Cut It
   3. Pragmatic Starter Kit
   4. Delight Your Users
   5. Pride and Prejudice
8. a