Penguin Programming Patterns

A PRAGMATIC GUIDE TO SOFTWARE DEVELOPMENT

DEJONGH STIJN
software development

Author(s):	Stijn Dejongh
Date:	2021-02-14
Document version:	1.0.0-DRAFT
Customer:	public domain knowledge

Contents

1 Document Overview

1.1 Purpose of the document

1.2 Intended Audience

1.3 Document Structure

2 Productivity Patterns

2.1 Overview

This section is a collection of Productivity related patterns. The patterns are meant as a guideline for your day-to-day development activities and will hopefully offer you a mental framework to reason about the tasks you are asked to perform.

For consistency, the patterns follow a similar structure. As we all know, **context matters**. This is why each pattern is prefaced with a short description of when it can be useful to consider using it. The aim is not to apply as many of them as you can on any given task. **This is not a bingo chart.**

2.2 List of Patterns

Pattern	Description
ROI metrics	Keep track of cost/benefits ratios and apply these to your work and reporting

2.3 Return on investment

2.3.1 Applicable Context

There is great difficulty is in expressing the "Value" and "Cost" of activities and projects. In a simplified business environment, these terms are usually substituted by "money earned / money invested". While this is a useful metric for sure, I feel that it lacks some of the non-monetary gains. There are those that try to calculate the value of everything by converting it into an amount of money. While there is value in this, I find it a difficult exercise to align with my personal ethics.

A question to ask here is: *"How would you value a human life? How would you value an improvement to* *someones self-image or daily struggles?"*

We shan't go into this much further, as we would deviate from the main point I wish to transfer and dive too deep into philosophical debate. The point I wish to make here is: there is often more to be gained than just money. And even if we would be able to quantify those "intangible" gains, we can also not predict the future. Which means that the definition of "value" will always be a subjective concept.

Even though this subjectivity is not something we can get rid of, it does not mean an empirical approach on top of this holds no value. Statistics as a field are a prime example of this statement. Their usefulness is not in the absolute numbers they provide, but by offering a formalized way of comparing different situations and contexts. As we all know, they are mere tools to help people and organizations define their strategy. Their usefulness is limited to how we interpret them, and what actions we take based upon them.

It is the same for a metric such as ROI. It's use is to be one of many metrics that can help guide you to make more informed decisions. But as with all metrics, it is up to you to interpret the data and make decisions. If you go one step further in this reasoning: it is the person who defines what "value" and "cost" means that decides what to include in their strategic decision making process. This in turn means that the metric itself is only as good as the contextual knowledge and expertise of the person (or group) defining the input.

2.3.2 Description of Pattern

ROI or "Return on investment" is a very useful metric for strategic planning. The idea is to maximize for value over time. This helps you to stay focused on the activities or task that bring the most value to your organization/project/customer, or even your own life.

The basic formula to calculate ROI is: **Value gained / Cost of activity**. I will leave the exact formula's to go from this calculation into percentage based metrics, etc.

Embrace the fluffiness of this metric, and focus on what is important to you and your context. Try and find a way to go from a qualitative to a quantitative evaluation of tasks. Once you and your context agree on those, the ROI metric becomes a useful tool for tracking the impact of the work you do to reach your goals. It is to be used as a tool for comparison to a previous period in your path. This way, you can see if you are progressing in the way you wish to progress or if you need to adjust your course.

2.3.3 Key Performance Metrics

- Your reports and analysis documents contain an ROI indication
- You think about cost/benefit ratio's while working

2.3.4 Related Patterns and Resources

Item	Description	Action
Good Enough Code	A programmer mindset pattern that aims to match code quality recuirements to intended use	Read the GEC pattern in the Programming section and reflect on how it relates to this.

3 Leadership

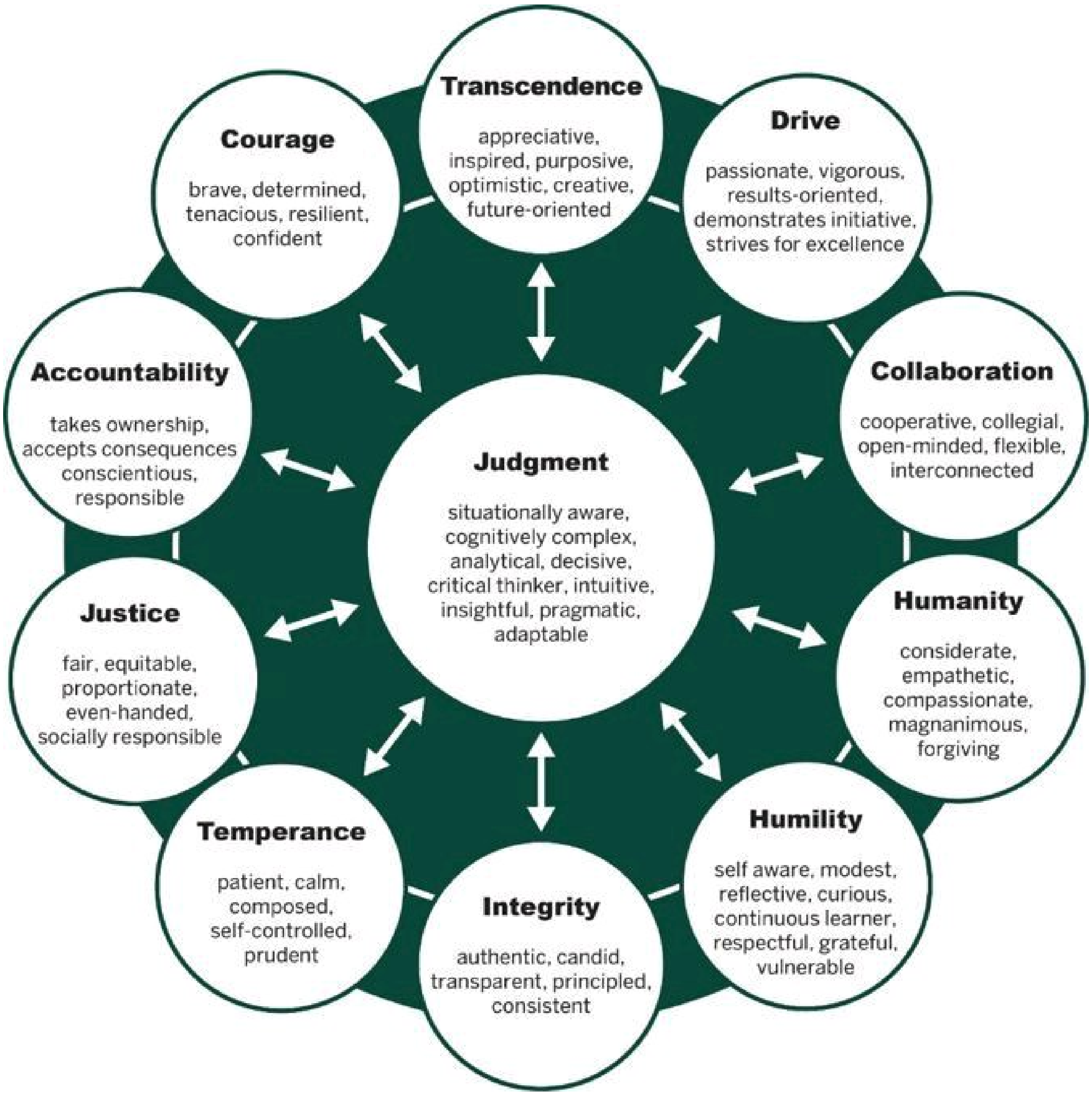
Transcendence
appreciative, inspired, purposive, optimistic, creative, future-oriented
Courage
brave, determined, tenacious, resilient, confident
Drive
passionate, vigorous, results-oriented, demonstrates initiative, strives for excellence
Accountability
takes ownership, accepts consequences conscientious, responsible
Collaboration
cooperative, collegial, open-minded, flexible, interconnected
Judgment
situationally aware, cognitively complex, analytical, decisive, critical thinker, intuitive, insightful, pragmatic, adaptable
Justice
fair, equitable, proportionate, even-handed, socially responsible
Humanity
considerate, empathetic, compassionate, magnanimous, forgiving
Temperance
patient, calm, composed, self-controlled, prudent
Humility
self aware, modest, reflective, curious, continuous learner, respectful, grateful, vulnerable
Integrity
authentic, candid, transparent, principled, consistent

Figure 1: Depiction of the core values of leadership

3.1 Further reading

4 Programming Patterns

4.1 Overview

This section is a collection of Programming related patterns. The patterns are meant as a guideline for your day-to-day development activities and will hopefully offer you a mental framework to reason about the tasks you are asked to perform.

For consistency, the patterns follow a similar structure. As we all know, **context matters**. This is why each pattern is prefaced with a short description of when it can be useful to consider using it. The aim is not to apply as many of them as you can on any given task. **This is not a bingo chart.**

4.2 List of Patterns

Pattern	Description
Good Enough Code	Avoiding the urge to overdesign your code by sticking to a good-enough implementation

4.3 Good Enough Code

4.3.1 Applicable Context

Issue: You get told that you overcomplicate simple tasks.

A mistake passionate programmers tend to make is to over-design simple things to make them theoretically and aesthetically more beautiful than they need to be at that point in time. In doing so, they often end up spending much more time and mental effort on a piece of software than is needed (or will ever be valuable).

Writing clean code is admirable, but it also has to make sense for the problem at hand. Creating a specific design by applying a pattern is to be done when it solves the problem at hand and makes the code more readable, robust, extensible or reusable.

Figure 2: Sometimes it is okay to keep it simple

4.3.2 Description of Pattern

Ask yourself: *"Is this code likely to be changed/expanded in the future?""Is my design solving an issue that is here NOW, or am I solving an issue that might never happen?""If this expected issue occurs in the future, can it be fixed easily at that time?"*

The idea of Good Enough Code is to write code as well designed as it needs to be AT THIS POINT IN TIME. If an idea for a more generic solution comes to mind during your implementation, take note of it. If in the future the problem you anticipated actually happens, or the code you wrote now is reused, it will be solved at that time.

Make sure the code you write at this point in time adheres to the basic principles of clean code and design, but do not solve future problems that might never happen.

4.3.3 Key Performance Metrics

- Throughput time of changes
- Regression introduced during tasks
- Function point count of changes
- Cyclomatic complexity
- Readability

4.3.4 Related Patterns and Resources

Item	Description	Action
Enterprise Quality FizzBuzz	A prime example of overdesigning something that can be done in a way more simple manner.	Go through the codebase, and ask *“Why would you want to do this? And why is it overkill here?”*
The bowling game kata	A programming kata by uncle Bob. Appart from learning how he thinks, the excercise also focusses on supressing your personal need to overly beautify a simple project.	Do the excercise and stop yourself from creating too many classes. Repeat the mantra: *“This is fine for now”* to supress your urges to add indirection or OO concepts to the design.

5 Software Architecture

5.1 What is Software Architecture?

The nuance between “architecture” and “design” is difficult to grasp. For me one is an extension of the other, but the nuances of the borders of these “blocks” are elusive.

5.1.1 Definition

In simple words, software architecture is the process of converting software characteristics such as flexibility, scalability, feasibility, re-usability, and security into a structured solution that meets the technical and the business expectations. This definition leads us to ask about the characteristics of a software that can affect a software architecture design. There is a long list of characteristics which mainly represent the business or the operational requirements, in addition to the technical requirements.

5.1.2 Characteristics

As explained, software characteristics describe the requirements and the expectations of a software in operational and technical levels. Thus, when a product owner says they are competing in a rapidly changing markets, and they should adapt their business model quickly. The software should be “extendable, modular and maintainable” if a business deals with urgent requests that need to be completed successfully in the matter of time. As a software architect, you should note that the performance and low fault tolerance, scalability and reliability are your key characteristics. Now, after defining the previous characteristics the business owner tells you that they have a limited budget for that project, another characteristic comes up here which is “the feasibility.”

A list of Software characteristics, known as *“quality attributes”* can be found on wikipedia.