Cheat Sheet Clean Code Development

1. Isoliere Aspekte

+ Don’t repeat yourself: Wiederholungen zum Wohle von Konsistenz und Fehlerfreiheit vermeiden!

+ Separation of Concerns: Jede Klasse hat eine klare Aufgabe und Funktionalitäten werden sinnvoll getrennt!

+ Single Responsibility Principle: Jede Klasse hat genau eine Aufgabe!

+ Interface Segregation Principle: Jedes Interface ist räumlich getrennt, unabhängig und nur von der Klasse benutzbar, für die das Interface vorgesehen ist!

+ Entwurf und Implementation überlappen sich nicht: Das zugrundeliegende Design sollte mit der Umsetzung weiter angepasst werden, sodass Design und Implementierung konsistent sind!

+ Integration Operation Segregation Principle: Es gibt Klassen, die nur Integrations- und keine Operationsmethoden haben und umgekehrt. Operations- und Integrationsmethoden sollten getrennt sein!

2. Minimiere Abhängigkeiten

+ Law of Demeter: Objekte können nur mit Objekten in ihrer unmittelbaren Umgebung interagieren!

+ Open Closed Principle: Das Verhalten einer Klasse sollte erweitert werden können, ohne dass man die Klasse modifizieren muss.

+ Tell, Don’t Ask: Objekte sollten ihre eigenen Daten halten, anstatt dass die Anwendung diese für sie verwalten muss!

3. Tue nur das Nötigste

+ Vorsicht vor Optimierungen: Zeit für die Implementierung von fertigen Funktionalitäten nutzen, anstatt zu viel an einer Funktionalität zu optimieren! Perfektionismus kann eine Hürde sein!

+ You Ain’t Gonna Need It: Nur Funktionen implementieren und behalten, die wirklich in den Klassen gebraucht werden!

+ Keep it simple and stupid: Möglichst einfach Funktionen verwenden, um den Code unkompliziert und effizient zu halten!

4. Halte Versprechen ein

+ Principle of Least Astonishment: alle Methoden funktionieren so, wie es ein Jedermann für den Kontext erwarten würde!

+ Implementation spiegelt Entwurf: siehe Entwurf und Implementation überlappen sich nicht

+ Favour Composition over Inheritance: Vererbung nur dann verwenden, wenn nötig und sinnvoll!

5. Halte Ordnung

+ Pfadfinderregel beachten: am Ende einer Coding-Session aufräumen und Docstrings an neuen Funktionen verfassen, die in der Zukunft das Verständnis und Erinnern erleichtern!

+ Komplexe Refaktorisierung: durch Refaktorisierung Konsistenz nach der Umbenennung von Funktionen und Variablen beibehalten!

+ Statische Codeanalyse: Hinweise von Sonarqube Metriken beachten und Code verbessern!