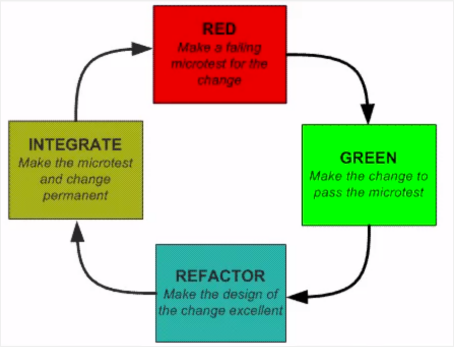
TDD Patterns:	**Specify IT:** Essence First, Test First, Assert First **Frame IT:** Frame First **Evolve IT:** Do The Simplest Thing That Could Possibly Work, Break It to Make it, Refactor Mercilessly, Test Driving
Pattern - 1 - Essence First	Was ist die benötigte Funktionalität, zerlegt auf seine Essenz? Heisst auch: Core Funktionalität, Essentielles, Architektur-Kritisches zuerst entwickeln.
Pattern - 2 - Test First	Was testen wir genau —> Namen sollte das aussagen. D.h. Namen zuerst schreiben
Pattern - 3 - Assert First	Welches verhalten wollen wir prüfen? —> Das Assert Statement sollte uns helfen die Methoden für unser neues Feature zu erstellen.
Pattern - 4 - Frame First	Wir erstellen Klassen, Konstruktoren und/oder Methoden die vom Assert-Statement (Assert First) gebraucht werden. Dies ist unser Gerüst/Frame.
Pattern - 5 - Do the simplest thing that could possibly work	Unser Code soll möglichst einfach sein, einfach ist oft auch robust. Und einfach braucht wenig Zeit. Hier spielt es keine Rolle ob wir Werte hardcoden. Wir wollen mit minimalem Aufwand den Test erfüllen. Damit vermeiden wir over-engineeredte Lösungen.
Pattern - 6 - Break It to Make It	Wir haben Tests und Funktionalität die diese Tests abdeckt aber wir wollen noch mehr/andere Funktionalität. Jetzt brechen wir den Code indem wir einen neuen Test schreiben der unsere neue Funktionalität prüft welche wir noch nicht implementiert haben. D.h. der Test wird rot und wir implementieren nun die neue Funktionalität.
Pattern - 7 - Refactor mercilessly	Wir verbessern den geschrieben Code unablässig und aggressiv durch Design Improvements. Da wir vorher das “simplest thing that could possibly work” geschrieben haben ist dieser Schritt sehr wichtig, sonst ist unser code scheiss, aka “held by duct-tape”.
Pattern - 8 - Test Driving	Wir wollen uns nicht zu weit von der grünen Bar entfernen. D.h. wir müssen die Tests oft ausführen, und sobald etwas rot wird dies wieder reparieren. Die Tests dürfen nicht lange rot sein sonst sind wir “offroad” was gefährlich ist.
Test Driven Development - Detaillierter Ablauf	1. **Test hinzufügen:** Für das Feature an dem wir arbeiten schreiben wir zuerst einen Test 2. **Alle Tests ausführen:** zum kontrollieren das der “Test harness” korrekt funktioniert. Unser Test sollte jetzt “rot” sein. 3. **Code schreiben:** Für unseren Test schreiben wir jetzt das Feature. So einfach wie möglich, mit dem Ziel dass der Test grün wird. KISS (Keep It Simple Stupid) 4. **Alle Test ausführen:** Unser Test sollte jetzt grün sein, die anderen (falls wir welche haben) Tests sollten ebenfalls immer noch grün sein. 5. **Refactor Code:** Jetzt machen wir den Code von unserem Feature schön/verbessern ihn/refactorn etc. 6. **Repeat:** Wir schreiben wieder einen neuen Test.
Test Driven Development - Heartbeat/Pulse	1. **RED**: Eine neue Testmethode die kompiliert aber noch nicht erfolgreich durchläuft *(Dauer von RED nach GREEN sind ein paar Minuten, sonst ist der getestete Code zu kompliziert)* 2. **GREEN**: Mit möglichst einfachen Mitteln soll erreicht werden dass der Test erfolgreich durchläuft 5-10Minuten 3. **REFACTOR**: Die getestete Methode/Code-Stelle wird verbessert/elegant gemacht. *(Viel Refactoring lässt sich automatisch/semi-automatisch durch das IDE erledigen !!)* 4. **INTEGRATION**: Der geänderte Code + Test wird ins Repo committed. **Essential First:** Zuerst Architekturkritische Funktionalität erweitern und erst nachher “Schnickschnack”. **Inverval:** Der Zyklus sollte möglichst klein gehalten werden. —> 6 “Runden” pro Stunde (1 Zyklus ca. 10Minuten) Integriert wird jeweils nach Abschluss eines Features (ebenfalls möglichst oft). Sicher mindestens am Ende des Tages (Backup). **Welches Feature zuerst?:** Kritische UseCases sollen zuerst implementiert werden
TDD - Übliche Refactorings	Inline Method, Compose Method, Rename Method, Extract Method, Move Method, Inline Temp, Pull Up Method
TDD - Seltene Refactorings	Collapse Hierarchy, Replace Conditional with Polymorphism, Extract Class
TDD - Grafik	
TDD - Inverted Design	Man könnte sagen TDD kehrt den Entwurf um. Weil: Normaler (Alter) Vorgang —> Design —> Code —> Test. Bei TDD dagegen: Test —> Code —> Design (Refactoring).