Test-driven development. *Emphasis on applying the rhythm and using/understanding the values and TDD principles.*

Rytmen

1. Skriv en test
2. Kør alle tests (fail)
3. Implementer
4. Kør alle tests (success)
5. Refaktoriser

Lav en testliste

Se specifikation af metoden/klassen
Lav specifikke tests der sørger for at specifikationen overholdes

Benyt rytmen

Skriv en JUnit testcase og assert noget
    Udviklingen drives af testen
    Alt testes hele tiden, så længe tests passerer er systemet i orden
    Ingen frygt i at refaktorisere og forbedre
Obvious implementation - implementer hurtigt
    Fake it
    Triangulering
Refaktorisering er ikke farligt da man har testcases

Test cases

Brug *indlysende data* for at beskrive hensigt (formel fremfor tal)
Repræsentativt data (en 17-cent mønt er ligeså repræsentativ som en 41-cent mønt)

Test stub - erstatter *DOU* så *indirekte input* til *UUT* kan styres af testen

Systematic black-box testing. *Emphasis on applying and understanding equivalence partitioning techniques and boundary value analysis.*

Lav en EC tabel

Condition	Invalid EC	Valid EC
Hvidt træk	*fromRow+1, fromCol, sort brik på to* *fromRow+1, fromCol, hvid brik på to* *toRow != fromRow+1* *toCol ∉ [fromCol, fromCol+/-1]*	*fromRow+1, fromCol, ingen brik på to* *fromRow+1, fromCol+/-1, ingen hvid på to*
Sort træk	*fromRow-1, fromCol, hvid brik på to* *fromRow-1, fromCol, sort brik på to* *toRow != fromRow-1* *toCol ∉ [fromCol, fromCol+/-1]*	*fromRow-1, fromCol, ingen brik på to* *fromRow-1, fromCol+/-1, ingen sort på to*

Grænseværdier - vigtige da der ofte er "off by one errors" og fejl omkring nul

Myers heuristikker
Benyt maksimalt antal gyldige EC (alle gyldige testes i så få testcases)
Lav testcases med en ugyldig EC i hver (brug kun en ugyldig for at undgå masking)

Variability management. *Emphasis on applying the four different techniques for handling variability and analysing their benefits and liabilities.*

1. **Source code copy**
   + simpel, hurtig, uafhængige varianter
   - høj vedligeholdelsesomkostning, versioner udvikler sig forskelligt
2. **Parameterisering**
   + simpel,
   - lav pålidelighed, store switches (svært at forstå koden), bloat
3. **Polymorfisk løsning**
   + change by addition, nem at forstå
   - compile-time binding, problem med multidimentionel varians
4. **Kompositionel løsning**
   + god kode genbrug, høj kohæson og lav kobling, runtime binding (kan skiftes på runtime)
   + seperation af tests, centraliseret variantbestemmelse
   - mange klasser, klienten skal kende til strategierne

Høj kohæsion - hvor stærkt relaterede og fokuseret, klassens responsibilitet er.
Lav kobling - hvor stærkt en klasse er koblet til andre klasser (afhængighed)

3-1-2 metoden
3. Find *variabilitetspunkt*
1. Udtryk variabiliteten som et *interface*
2. *Uddeliger* opgaven til en klasse der implementere interfacet

Test stubs and unit/integration testing. *Emphasis on applying test stubs and understanding the testing levels of unit/integration/system testing.*

Test stub - erstatter *DOU* så *indirekte input* til *UUT* kan styres af testen

UUT - **Unit Under Test** - softwareenheden der testes
DOU - **Dependent On Unit** - enhed der leverer *indirekte input* til *unit under test*

Unit test - test af en enkelt klasse eller metode.
Integration test - test af samarbejdet mellem klasser
System test - test af det fulde system. Explorative testing er ofte brugt

White box testing - det er muligt at læse koden, så mere specifikke testcases kan fremstilles
Black box testing - *UUT* behandles som en black box (koden er ikke tilgængelig). Tests skal laves ud fra kendskab til specifikationen, gængse fejl og generel viden om programmeringsteknikker.
Black og white box testing komplementerer TDD godt.

Explorative testing - modsat systematisk test. Der testes efter mavefornemmelse og heuristikker. Ofte brugt ifm. *black box testing* og *system test*.

Design patterns. *Emphasis on finding the proper design pattern for a problem at hand and applying it.*

Behavior - Agere på en observerbar måde
Responsibility - Ansvar for at svare pålideligt på en request
Protocol - Beskriver rækkefølgen af interaktioner og handlinger som forventes af et sæt roller
Role - Ansvar og dertilhørende protokoller

Strategy - anvend forskellige algoritmer
State - strategy pattern, dog skiftes mellem algoritmer baseret på intern state. Ofte *delegering*.
Abstract factory - leverer konkrete objekter så andre klasser ikke er koblet med disse
Facade - få en samling af klasser til at fremstå som en enkelt klasse udadtil
Decorator - tilføj funktionalitet til en klasse. Metoder delegeres til konkret klasse. Dekoratorer kan dekoreres
Adapter - tilpasser et interface til et andet. Eksempelvis 3. parts software med andre parametre
Builder - laver produkt af mindre enheder. Metode til hver enhed. Forskellige builders laver forskelligt resultat
Command - enhver commando er et objekt. Det er let at lave macroer (*composite*) og implementere undo
Iterator - gennemløb af datastruktur uden viden om underliggende implementation
Proxy - placeholder for konkret objekt. Kan udskyde load af konkret objekt. Brug eks. ved billeder
Composite - samling af objekter der implementer samme interface. Enkelte og samlinger behandles ens.
Null object - class that does nothing in methods. Used in automated testing when algorithm takes long time
Observer - hold liste med objekter og notificer alle ved event. Objekter kan tilmelde sig signalet (broadcast)
Model-View-Controller - opdeling af ansvar i 3 logiske områder. Øger kohæsion, flere views (State-UI-logik)
Template method - opdeling af algoritme i mindre dele. Uddeleger løsning af mindre dele

Compositional design. *Emphasis on applying compositional design principles and relating it to concepts behavior, responsibilities, roles, and multi-dimensional variance.*

3-1-2 metoden

3. Find *variabilitetspunkt*
1. Udtryk variabiliteten som et *interface*
2. *Uddeliger* opgaven til en klasse der implementere interfacet

Delegation

Responsibilitet uddelegeres til forskellige klasser hvilket giver høj kohæsion
Roles - klasser of interfaces har roller i patterns. Forklar disse roller
Multidimentional variance er simpelt at håndtere vha. *delegering*

Frameworks. *Emphasis on designing frameworks and understanding framework theory.*

Definition - en fleksibel struktur der gør det let at udvikle applikationer indenfor et bestemt domæne. Genbrug af kode og design

Hotspots/hookpoints - hvor det er muligt at adaptere frameworket til det aktuelle behov
Frozen spots - frameworkkode der ikke må pilles ved

Inversion of control - Hollywood princippet
Dependency injection - abstract factory injecter dependencies til andre moduler
Dependency inversion principle - Alle moduler skal afhænge af abstraktioner. Dependency injection bruges til at koble de egentlige moduler sammen.

Opførsel er prædefineret (skelet) (inversion of control)
Kan bruges indenfor et veldefineret domæne
Definerer protokoller mellem frameworkets komponenter
Fleksibel struktur
Genbrug af kode og design (***meget gennemtestet kode***)