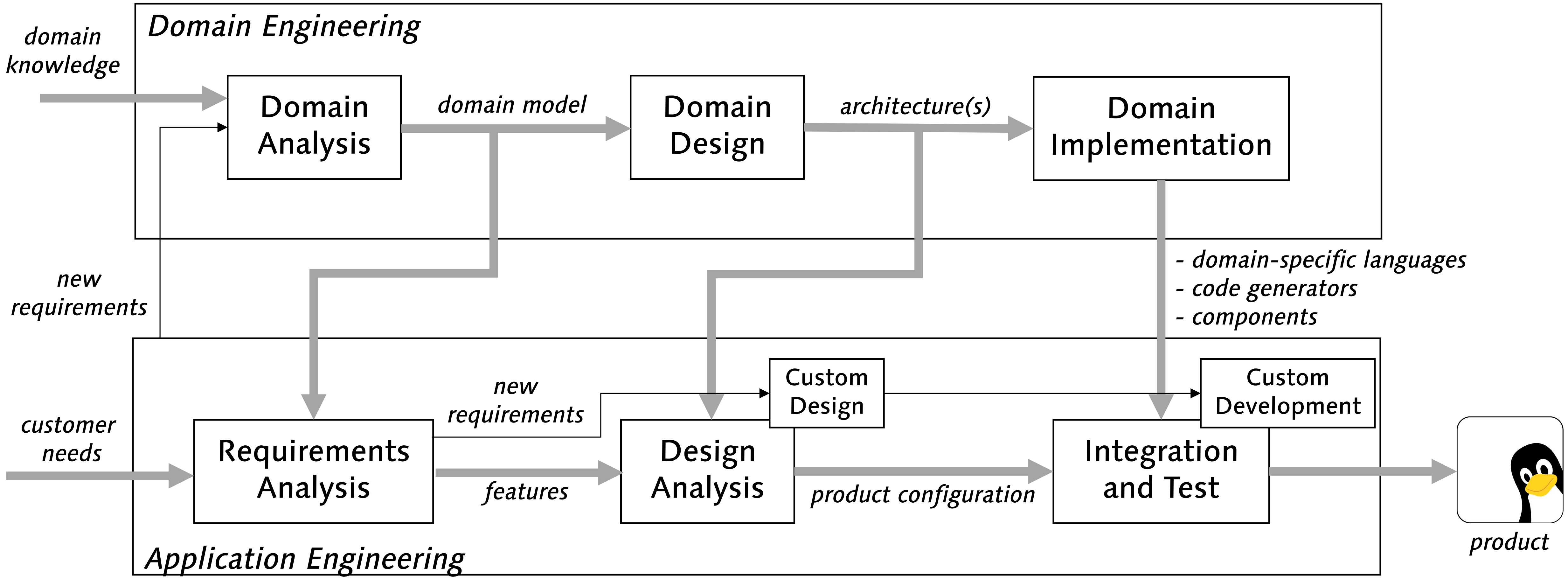
Software Product Line Engineering

Lab Class 3

Outline

- Presentation Assignment 2

Task 1a: Domain and Application Engineering

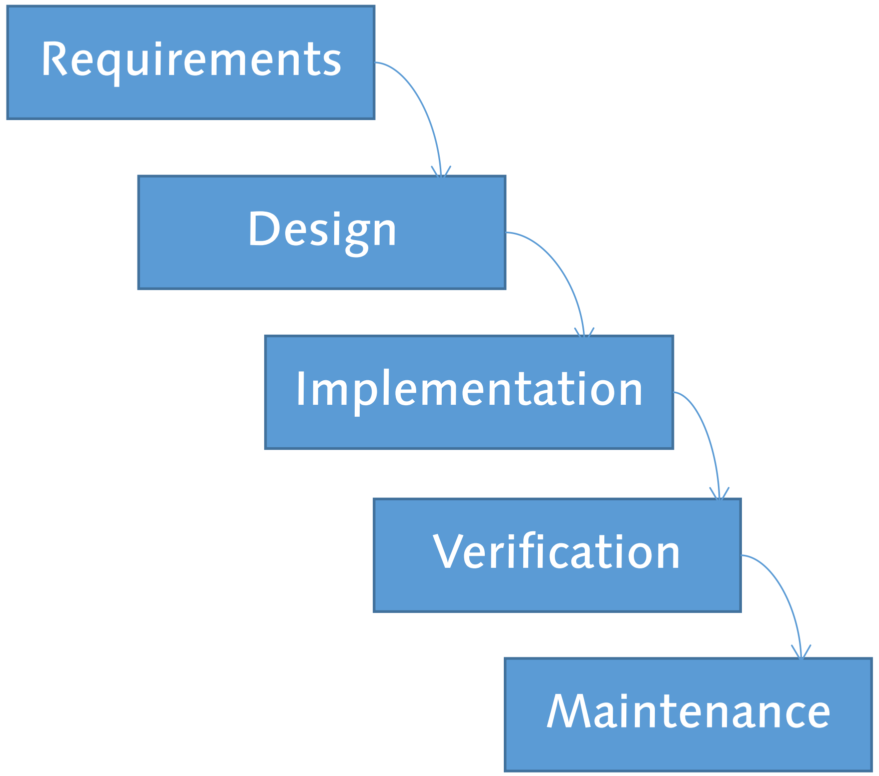


Task 1b: Waterfall model vs AE/DE

- Classical process models are linear, focus on product delivery.

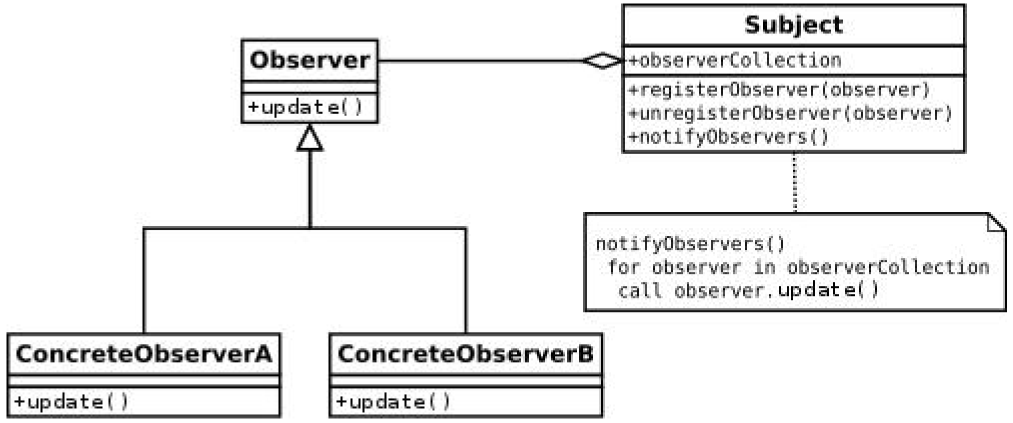
- AE/DE: emphasizing reusability
    - DE: “[…] is the activity of collecting, organizing, and storing past experience in building systems […] in a particular domain in the form of **reusable assets** […], as well as **providing an adequate means for reusing these assets** (i.e., retrieval, qualification, dissemination, adaptation, assembly, and so on) when building newsystems."

(Czarnecki/Eisenecker: Generative Programming)



Task 2a: Observer Pattern

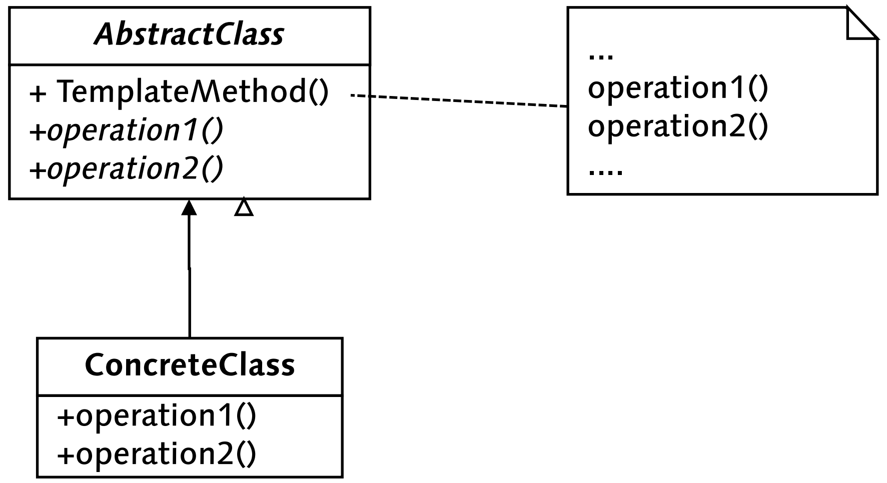
- Behavioral Pattern by the GoF
- Observers can subscribe to a subject.
- Subject can notify observers by calling update() method



https://en.wikipedia.org/wiki/Observer_pattern

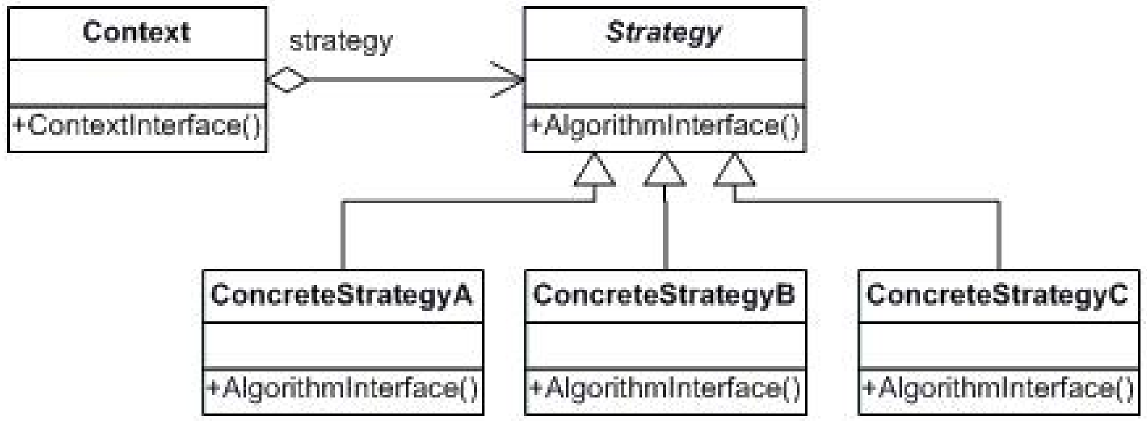
Task 2b: Template Method Pattern

- Behavioral pattern by the GoF
- A template method in an abstract class uses methods that intentionally unimplemented.
- A concrete class specifices and provides unimplemented methods for customization.



Task 2c: Strategy Pattern

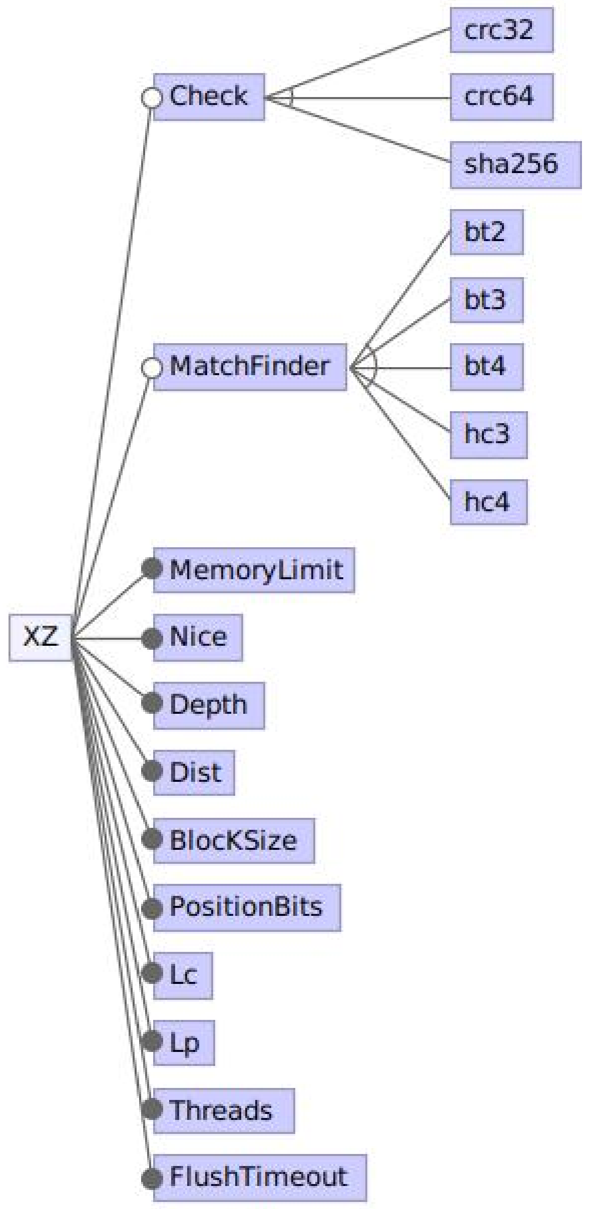
- Behavioral Pattern by the GoF
- Alternative implementations of an algorithm (concrete strategies) are hidden between an Strategy interface.

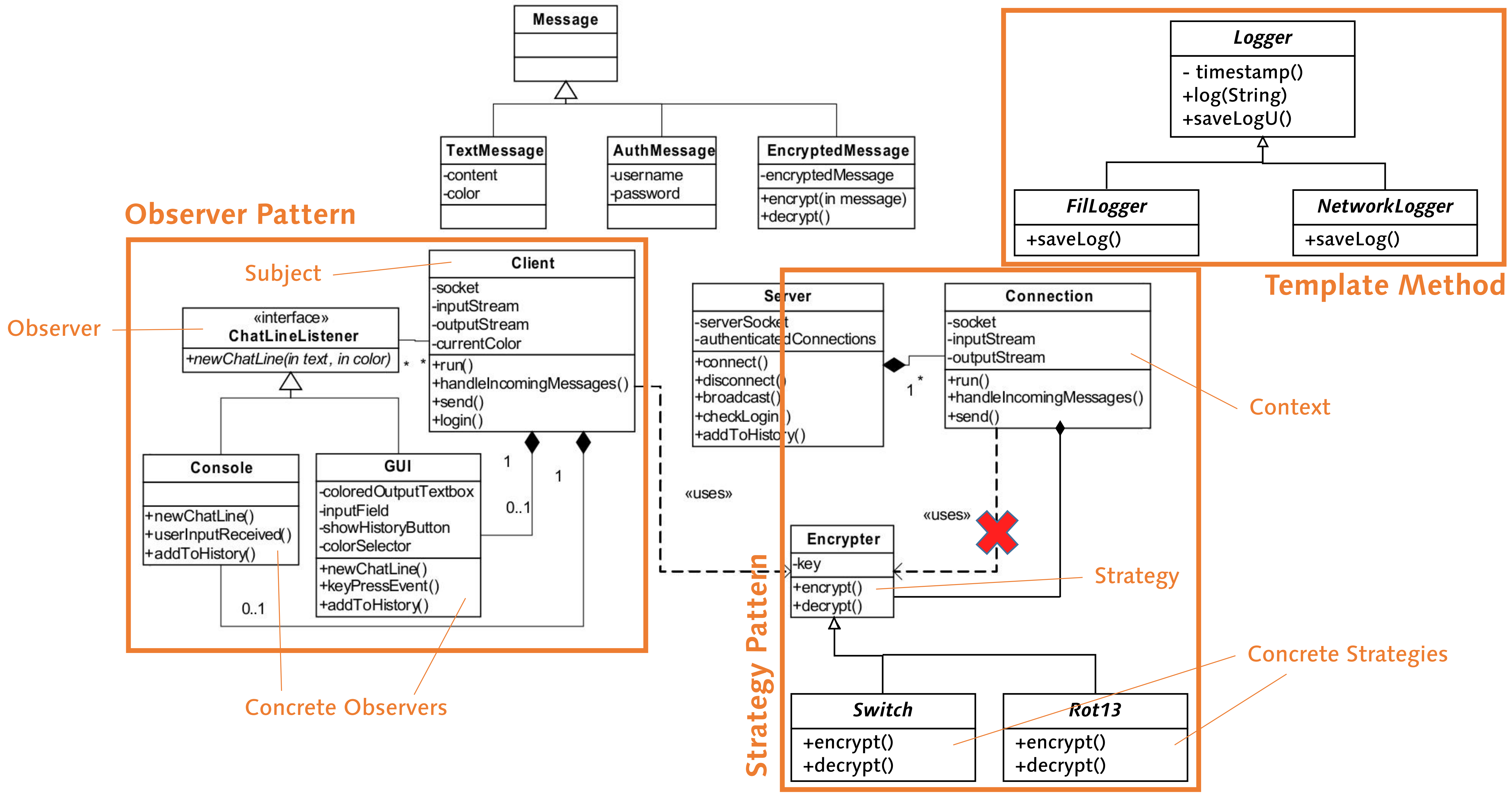


https://www.dofactory.com/net/strategy-design-pattern

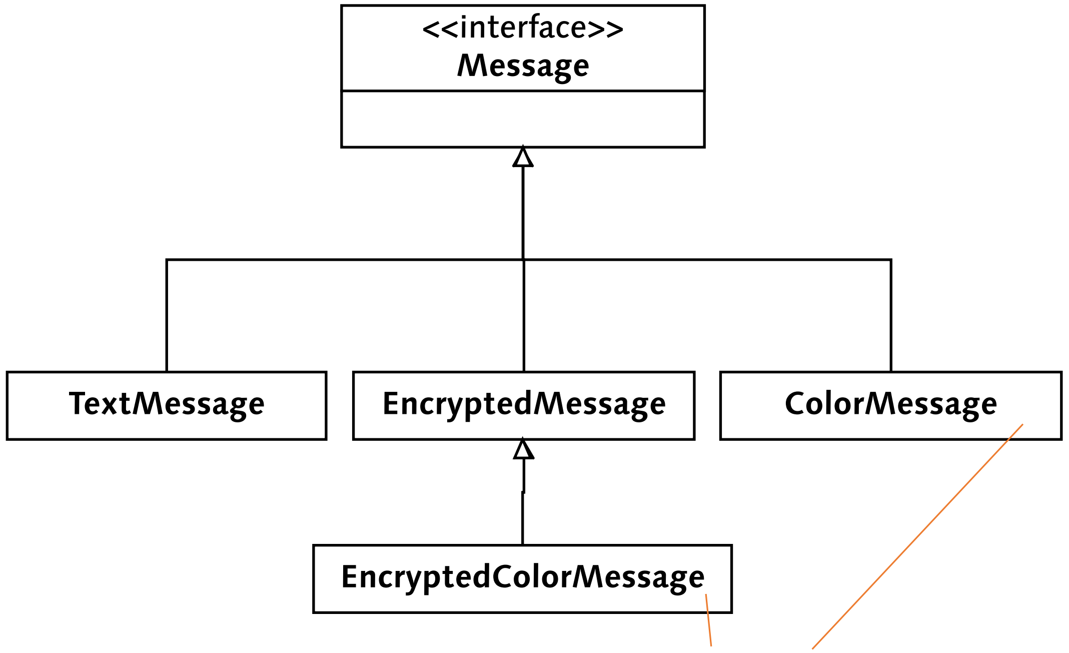
Task 3: Feature Model Extraction

- Options for compression
  - Check (crc32, crc64, sha256), MatchFinder (bt2, bt3, bt4, hc3, hc4), MemoryLimit, Nice, Depth, Dict, Dist, BlockSize, Threads, PositionBits, Lc, Lp, FlushTimeout, …
- Cross-tree constraints
  - $lc + lp \leq 4$
  - $bt2 \Rightarrow nice \geq 2$
  - $(hc3 \vee bt3) \Rightarrow nice \geq 3$
  - $(hc4 \vee bt4) \Rightarrow nice \geq 4$



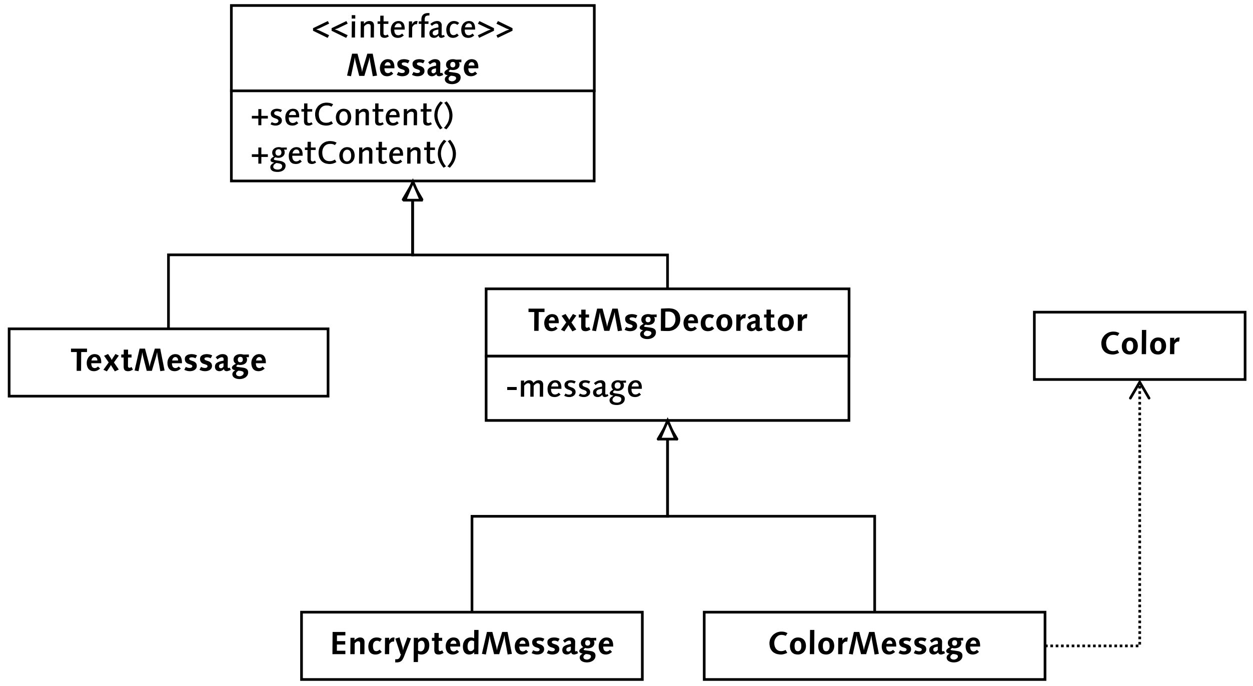

Message
TextMessage
-content
-color
AuthMessage
-username
-password
EncryptedMessage
-encryptedMessage
+encrypt(in message)
+decrypt()
Observer Pattern
Subject
Observer
«interface»
ChatLineListener
+newChatLine(in text, in color)
Client
-socket
-inputStream
-outputStream
-currentColor
+run()
+handleIncomingMessages()
+send()
+login()
Console
+newChatLine()
+userInputReceived()
+addToHistory()
GUI
-coloredOutputTextbox
-inputField
-showHistoryButton
-colorSelector
+newChatLine()
+keyPressEvent()
+addToHistory()
1
1
0..1
0..1
* *
Concrete Observers
«uses»
Server
-serverSocket
-authenticatedConnections
+connect()
+disconnect()
+broadcast()
+checkLogin()
+addToHistory()
1 *
Connection
-socket
-inputStream
-outputStream
+run()
+handleIncomingMessages()
+send()
Context
«uses»
Encrypter
-key
+encrypt()
+decrypt()
Strategy
Strategy Pattern
Switch
+encrypt()
+decrypt()
Rot13
+encrypt()
+decrypt()
Concrete Strategies
Logger
- timestamp()
+log(String)
+saveLogU()
FilLogger
+saveLog()
NetworkLogger
+saveLog()
Template Method

Inflexible Extension Mechanism



We cannot combine the features *Encryption* and *Color* in this inflexible hierarchy without code replication!

Decorator Pattern



Message msg = new EncryptedMessage(new ColorMessage(new TextMessage( )))

Task 4: Feature Modeling

a) Feature diagrams are easier to comprehend and communicate.

b) Product Line *A* ...

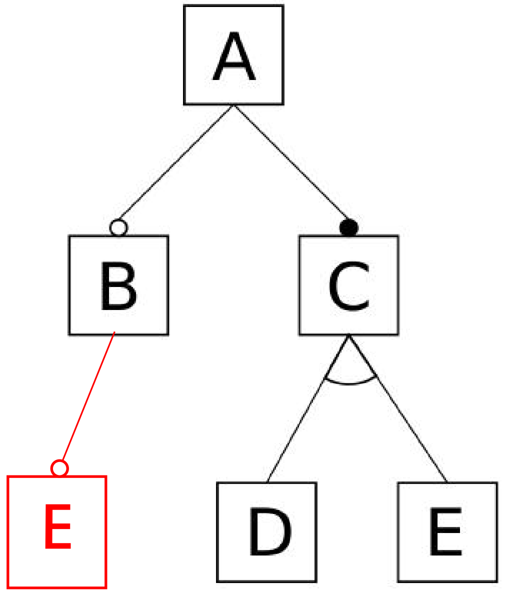
i. Valid configurations: *ACD*, *ACE*, *ACDB*, *ACEB*

ii. Naive solution: Create all combinations $\{0, 1\}^{\{A,\ B,\ \ldots\}}$, validate each one, retain only valid ones.

iii. $A \wedge (B \Rightarrow A) \wedge (C \Leftrightarrow A) \wedge [((D \vee E)) \Leftrightarrow C \wedge \neg(D \wedge E)]$

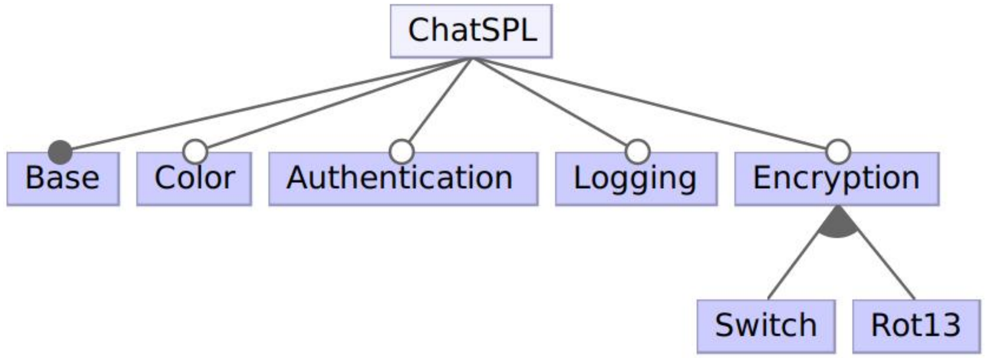
iv. The implication $E \Rightarrow B$ reduces the number of valid configurations: ACE becomes invalid

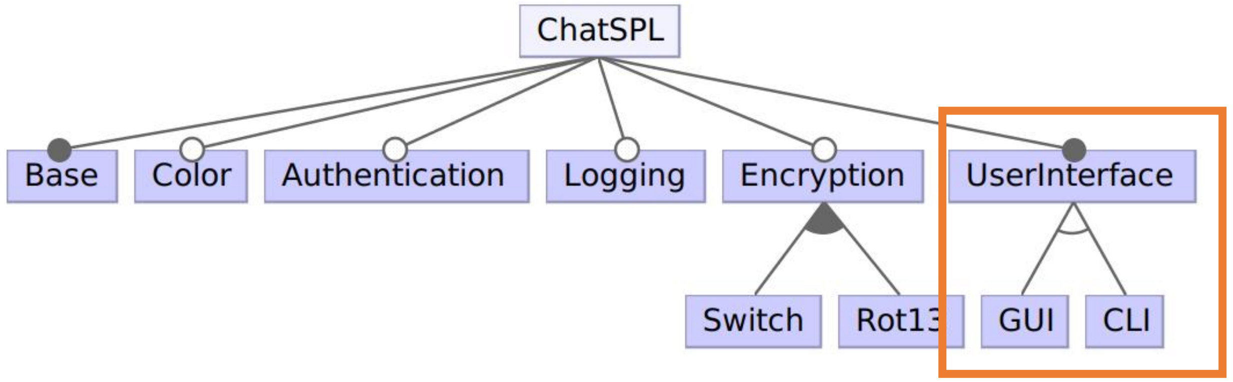
v. $(E \Rightarrow B)$ cannot be modeled as an optional feature, but as a cross-tree constraint.



E ⇒ *B*

Task 4c-e)


ChatSPL
Base
Color
Authentication
Logging
Encryption
Switch
Rot13


ChatSPL
Base
Color
Authentication
Logging
Encryption
UserInterface
Switch
Rot13
GUI
CLI