Group 2

Achievement A1 - GRASP
Advanced software design
Uppsala University

Mikael Hagafors Sara Back Daniel Owen-Berghmark Henrik Bihlar

December 6, 2016

GRASP Principles

- Controller

  Controller is the first non-UI object which is responsible for receiving and controls system events. For example in our class diagram the `Requester` and the `Responder`.

- Creator

  Creator applies to every object solves the problem of who creates a specific object. The basic idea is to find a creator that needs to be connected to the created object or has the initializing data for the object etc. In our class diagram `Matcher` creates match objects because it has all the necessary information to do so and `Requester` creates `Jobs` for the same reason.

- Indirection

  Indirection creates low coupling between two elements by assigning the responsibility of mediation between them to an intermediate object. Which is done in our class diagram by the `Match` object to create the connection between `Requester` and `Responder`.

- Information expert

  Information expert is the principle of assigning responsibility to the class that has the information to fulfill that responsibility. In our class diagram we see this for example in that the `Requester` has the responsibility to create `Jobs` as the former has the information/knowledge needed to create the latter.

- High cohesion

  Keep methods in a class highly related to each other. For example, having database logic together with UI stuff is bad. Example in class diagram: `Matcher` has three methods, all highly related to each other

- Low coupling

  Coupling is a measure of how closely connected two classes/elements are. To acheive as low coupling as possible we do not want connect modules unless we have to, for example

the class `Notify` which is only connected to the `InvoiceCreator` and the objects it needs to notify.

- Polymorphism

  Polymorphism in OOP is about how a class can share its functionality to subclasses and the subclasses can use the main functionality and modify the behaviors for a unique functionality to the subclass.

  For Example, we can see polymorphism in the `Invoice` interface where it is implemented as the classes `Bill` and `Wages`. Where `Requster` is used in `Bill` and the `Responder` is used in wages.

- Protected variations

  Protected variations is a way to avoid the impact of variations of some elements on other elements by creating assigning responsibilities to a stable interface and use polymorphism to create various implementations of this interface. For example in our class diagram the `Skills`.

- Pure fabrication

  A pure fabrication is a class that does not exist in the domain model and created to acheive low coupling and high cohesion. In our class diagram we see this for example in the class `InvoiceCreator`.