Theme 4.1 SE, SA Learning Task 1, M. Kindermans & M. van Mastrigt, group 10b.

Assesment 1 - Quality Attributes

a) Have a look at chapters 16 and 17 from MS Application Architecture Guide. What is the difference between quality attributes and crosscutting concerns?

They both affect the entire application. The difference between them is that quality attributes affect things like run-time performance, system design, and user experience. So they affect the decisions that will be made. The crosscutting concerns are more about actions that are performed throughout the application (eg. validation, logging, and authentication).

b) What is the difference between a Perspective and a Viewpoint as defined in Software Systems Architecture?

A viewpoint is a pattern or generalization of a view. In OO terms, the viewpoint is the class (or template) and the view is the instance of that class. A view is a representation of the whole system, as seen through the prism of specific concerns. Where these concers come from the stakeholders. It is more about explaining the system in an understandable way for that certain group of stakeholders.

The perspectives are a collection of patterns, templates, and guidelines to ensure the system has the right quality properties.

c) Describe two quality attributes that typically conflict, i.e. more of one attribute means less of the other. In what way can you make tradeoffs between them? (Don’t choose the examples mentioned at the lectures.)

Two attributes that often conflict are securability and usability. A more secure application often requires more interaction by the user. E.g. using a phone for two steps verification. This reduces the usability of the application, because it takes a user more time to log in. There can be made compromises between the two quality attributes by e.g. using a secure protocol, or by encrypting data. This does not affect the usability, but does improve the security.

Assesment 2 - Architectural Patterns

We beantwoorden vanaf hier de vragen in het Nederlands.

a) Describe in your own words what the difference is between Design Patterns as taught in the previous period (the Gang-of-Four patterns) and architectural patterns.

Het probleem dat een architectural pattern oplost ligt op een hoger/abstracter niveau dan die van een design pattern. Architectural patterns bieden een oplossing voor problemen omtrent architectuur en de kwaliteitsattributen (quality properties), zoals bijvoorbeeld keuzes omtrent indeling van systeemcomponenten. Waarbij Design Patterns - veel voorkomende - problemen oplossen die onstaan bij het ontwikkelen van software. Bijvoorbeeld hoe er duidelijke variabelnamen gekozen kunnen worden.

b) Design Patterns and architectural patterns are often based on the same ideas. Which Design Patterns are most similar to the architectural patterns Pipes and Filters and Indirection? (Pipes and filters is described on page 140-142 of Software Systems Architecture).

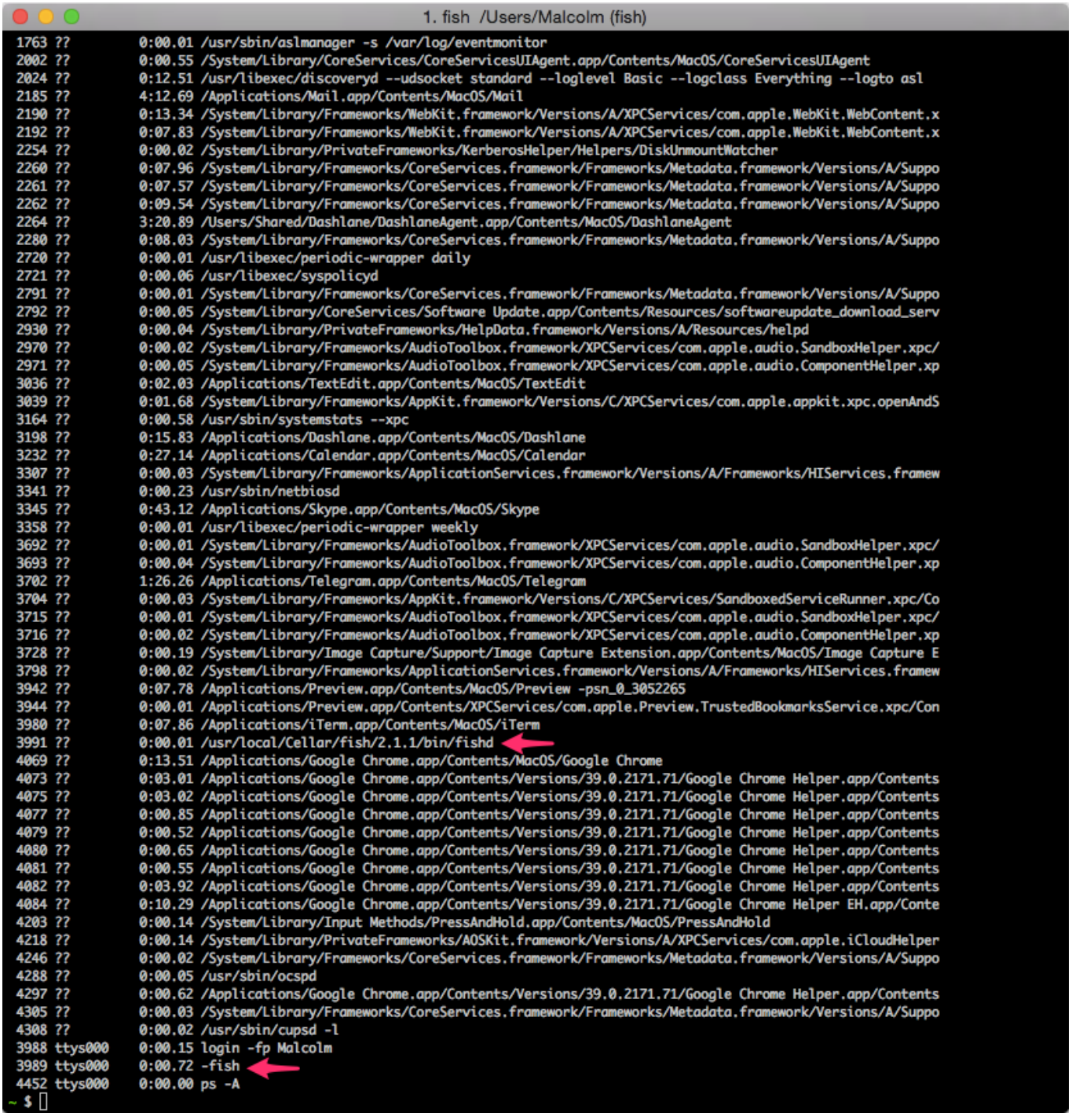
Op sommige input zijn meerdere bewerkingen noodzakelijk. Als de volgorde van die bewerkingen onafhankelijk van elkaar moet zijn, dan moet iedere bewerking dus hetzelfde input en output type hebben. Dat is precies wat het Pipes and Filters design pattern probeert te bewerkstelligen. Het Design Pattern dat daar op lijkt is Fluent Interface.

De Proxy en Delegation Design Patterns lijken veel op de Indirection pattern. Bij deze patterns is de locatie van het object bekend. Er kunnen via dit zogenaamde tussencomponent instructies worden gestuurd naar het object. Deze tussenlaag kan onder andere dienen voor beveiliging of simplificatie van functionaliteit.

c) Give an example of the application of the Pipes and Filters-pattern.

Pipes and Filters worden in Linux veel gebruikt. Ook letterlijk. Je kunt functies namelijk doormiddel van pipelines combineren. Ieder commando in Linux heeft minimaal drie channels: *STDIN*, *STDOUT* en *STDERR*. De output (die wordt geschreven naar *STDOUT*) kan weer als input dienen voor de *STDIN* channel. Door bijvoorbeeld het commando `ps` te gebruiken, zie je een hele lijst met de statussen van alle processen, zoals hieronder.

`$ ps –A` , geeft:



Zoals te zien zijn de processen die met Fishshell te maken hebben gemarkeerd.

De output (*STDOUT*) van dit commando kun je als input (*STDIN*) gebruiken voor het commando `grep`. Dus als je wilt filteren op alle processen met verband houden met Fishshell, kun je het volgende uitvoeren: `$ ps –A | grep fish`.

```
~ $ ps -A | grep fish
 3991 ??         0:00.02 /usr/local/Cellar/fish/2.1.1/bin/fishd
 3989 ttys000    0:00.86 -fish
 4622 ttys000    0:00.00 grep fish
```

d) Lookup an architectural pattern of your choice and describe it in your own words. You are not allowed to copy/paste any text from existing sources. Use the same format as the example description of the Shared Repository-pattern given on Blackboard. Also, have a look at the article A Pattern Language for Pattern Writing (especially patterns B.1 and B.2) for an explanation of the pattern elements in the template.

We hebben het ***Peer-to-Peer Architectural Pattern*** gekozen.

Pattern naam: *Peer-to-peer pattern*

Beschrijving: Een peer-to-peer netwerk bestaat uit minimaal twee componenten, ook wel 'peers' genoemd, maar kan ook veel meer componenten of peers bevatten. Ieder component/peer in dit netwerk heeft dezelfde rechten en kan dezelfde rollen vervullen. Deze rollen kunnen ook door alle componenten/peers tegelijk vervuld worden. Ieder component of peer kan zowel de rol van node als die van server vervullen en kan connecties opstarten en afsluiten.

Context: Een peer/component heeft bestanden en deelt deze via een netwerk waarbij de data niet via een centrale server loopt, maar direct tussen de componenten wordt uitgewisseld.

Probleem: Het delen van bestanden via een netwerk met een centrale server heeft als nadeel dat ieder component afhankelijk is van de server voor de beschikbaarheid van de bestanden en de snelheid waarmee deze getransporteerd kunnen worden. Bij uitval van de server, gaan bij alle componenten binnen het netwerk de connecties met de server en overdracht van de bestanden verloren.

Krachten:

- *Performance:* Doordat het bestand vanuit meerdere plekken (in delen, door de verschillende componenten) beschikbaar wordt gesteld, is de snelheid niet afhankelijk van de capaciteiten van een enkele server. Hierdoor kunnen de dichtsbijzijnde componenten worden gekozen voor het binnenhalen van bepaalde bestandsdelen voor een zo efficiënt/snel mogelijk transport.
- *Availability:* Een bestand staat niet op één enkele locatie opgeslagen. Daardoor is de beschikbaarheid niet afhankelijk van één component. Hierdoor wordt de beschikbaarheid dus in zijn geheel vergroot (d.m.v. redundantie).

Oplossing: Ieder component binnen het netwerk bevat een gedeelte van (of alle delen van een bestand). Wanneer het bestand door een bepaald component opgevraagd wordt, kan deze aan de hand van alle componenten die over gedeelten van het bestand beschikken, opnieuw worden opgebouwd.

Consequenties:

- Omdat bestanden tussen de peers worden uitgewisseld, moet het IP-adres van de componenten openbaar zijn. Aan de hand van een IP-adres kan de locatie van een peer bij benadering worden bepaald.
- Een ander nadeel is dat een virus ook heel snel kan worden verspreid. Wanneer een bestand (en dus ook een kwaadaardig bestand) eenmaal in een peer-to-peer netwerk zit, is het heel lastig om deze weer uit het netwerk te krijgen.

Assignment 3

Zie bijlage `SAD NOVA LVG.pdf`.