به نام خدا

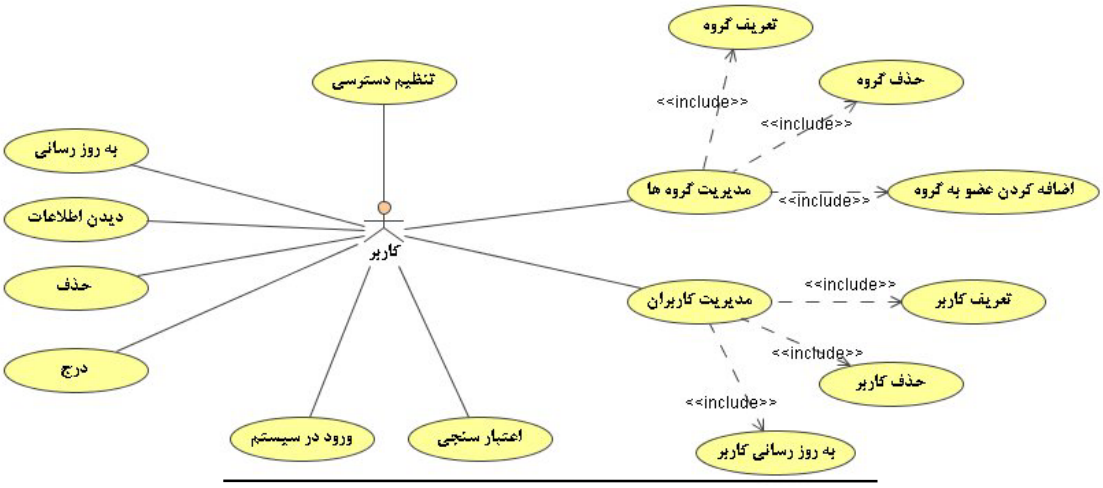
تمرین سری دوم
مهندسی نرم‌افزار۲

نعیم اصفهانی
۸۴۲۰۱۰۰۳

سئوال ۱:

(a) اکتورها و نمودار موارد کاربرد

طراحی سیستم به گونه‌ای صورت گرفته است که هر کاربر امکان پذیرفتن هر نقشی را دارد بنابراین به طور کلی تنها یک اکتور خواهیم داشت. این اکتور بالقوه امکان دسترسی‌ها را دارد ولی با توجه به نقش لحظه‌ای خود امکان بالفعل را به‌دست می‌آورد. در مورد موارد کاربرد در سمت چپ شکل هم می‌توان از در بر داشتن[1] استفاده کرد ولی چون سطح این موارد بالاتر است و کاربر معمولا به طور مستقیم با یکی از این موارد می‌تواند سر و کار داشته باشد، این کار انجام نشد.



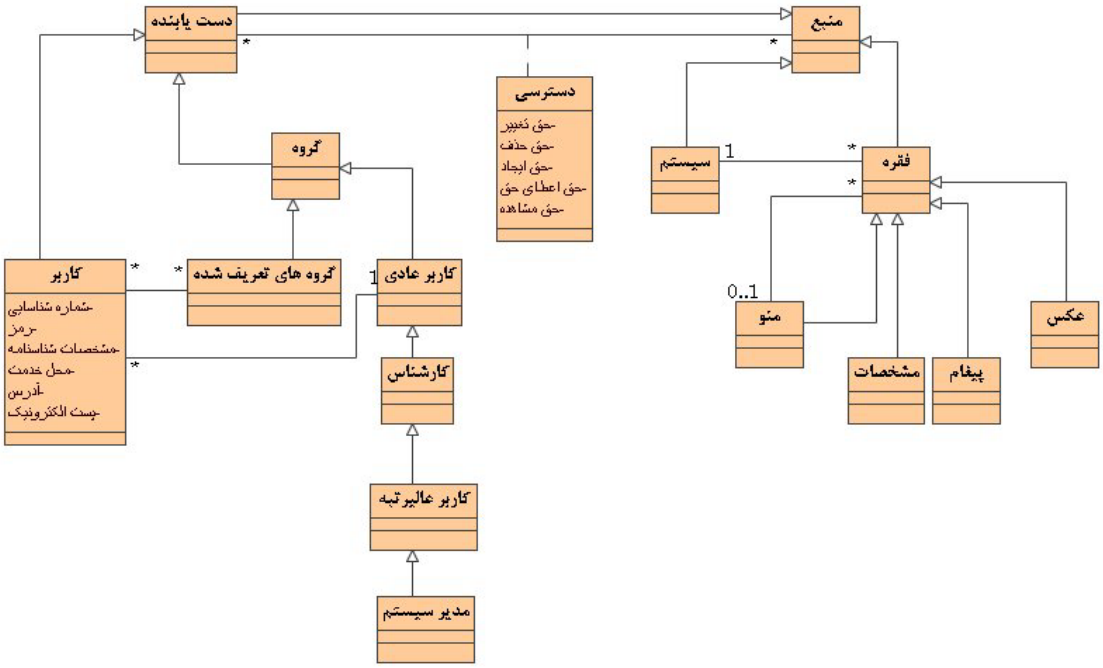
نمودار موارد کاربرد

(b) نمودار کلاس‌ها

طراحی در این سیستم به گونه‌ای صورت گرفته است که خود گروه‌ها و کاربران هم یک منبع تلقی شده و می‌توان به آن‌ها دسترسی در سطوح مختلف داشت. در این سیستم هر کاربری که حق دسترسی دارد می‌تواند گروه تعریف کند. نکته‌ی قابل توجه این است که لزومی برای تعریف گروه‌های از پیش تعیین شده به عنوان کلاس دیده نمی‌شود و می‌توان حضور آن‌ها را به صورت دیگری تامین کرد. برای یکنواختی سیستم این گروه‌ها به صورت کلاس تعریف شدند. البته باید توجه داشت که این کلاس از نوع کلاس‌های یک نمونه‌ای[2]

[1] include
[2] Singleton

هستند. در ضمن اطلاعات مربوط به صفات کلاس‌ها با توجه به صورت سئوال پر شده و بدیهی است که در حد همان صورت سئوال اطلاعات دارد و کامل نیست.

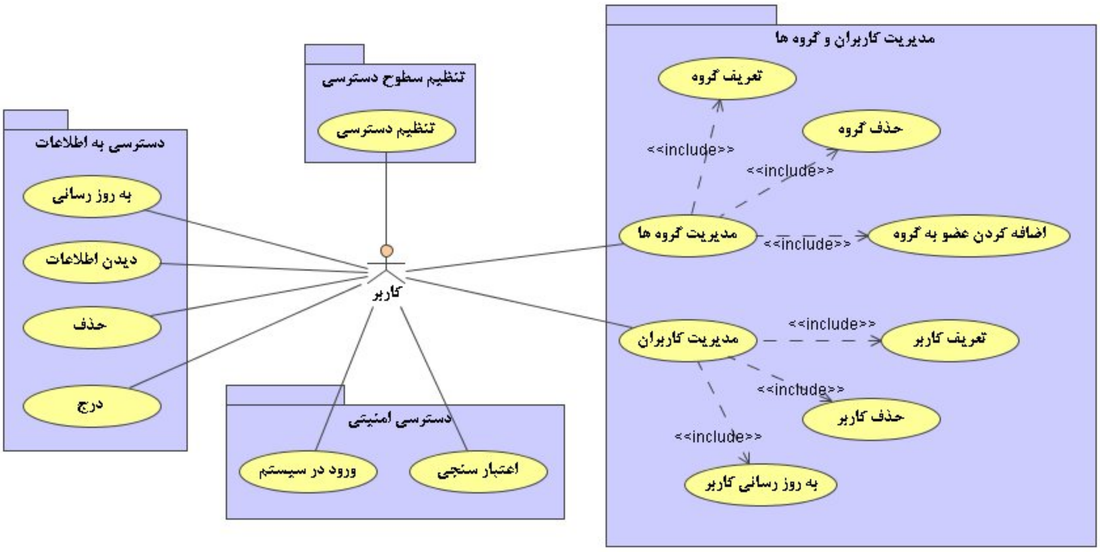


نمودار کلاس‌ها

(C) تقسیم بندی به زیرسیستم‌ها و طراحی واسط کاربری

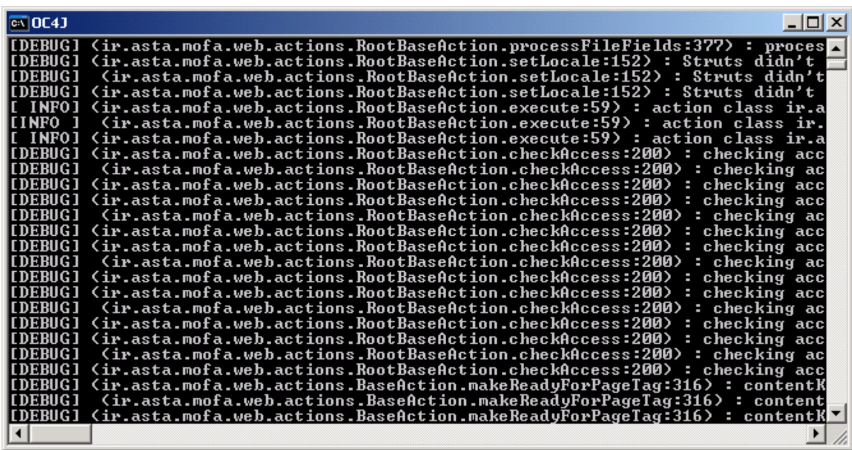
همان‌طور که از شکل بر می‌آید سیستم به چهار زیرسیستم تقسیم می‌شود. یک زیر سیستم برای کارهای اولیه که سیستم بدون داشتن مکانیزم کنترل دسترسی دارد(این کارها بسیار اولیه و در حد عملیات پایه‌ای دیده شده‌اند بنابراین نیازی به چندین زیر سیستم نیست) یک زیر سیستم برای مدیریت کاربران و دو زیر سیستم دیگر یکی برای اعمال سطوح دسترسی و دیگری برای بررسی این دسترسی‌ها در زمان اجرا.

فرض شده است که این سیستم مبتنی بر وب است و بنابراین یک نمونه واسط کاربری با زبان HTML ارائه شده است. این نمونه از واسط کاربری برای شخصی است که تمام دسترسی‌ها را دارد. در قسمت بالا نام زیر سیستم‌ها دیده می‌شود، در قسمت سمت راست منوهای سطح یک زیر سیستم انتخاب شده دیده می‌شوند. در وسط نیز امکان عملیات اولیه بر روی اطلاعات وجود دارد. باید توجه داشت که اگر یک کاربر امکان دسترسی به یک گزینه را نداشته باشد اصولا آن را نمی‌بینند. مثلا امکان انتخاب را ندارد و یا دگمه‌ی حذف را نمی‌بینند.



زیر سیستم‌ها

نمونه‌ی واسط کاربر



نمونه‌ای از چگونگی اجرا

(d) سطح آزمون

به طور کلی چهار سطح در آزمون مطرح می‌شود(برگرفته از مستندات RUP ضمیمه شده). آزمون واحدها[3]، آزمون سیستم[4] ، آزمون یکپارچگی[5] و آزمون پذیرش[6]. به طور کلی همیشه با توجه به اندازه و قیمت پروژه بهتر است که هرچه بیشتر آزمون انجام شود ولی اولویت این آزمون‌ها با توجه به نوع سیستم‌ها انجام می‌شود؛ به طور مثال در سیستمی که تعداد اجزا کم می‌باشد و اندازه‌ی اجزا بزرگ است اهمیت آزمون یکپارچگی پایین می‌آید چون اصولا واحدها ارتباط کمی با هم دارند و در عوض اگر سیستم از تعداد زیادی واحد که با هم کار می‌کنند تشکیل شده باشد، اهمیت این آزمون زیاد می‌شود. در زیر به اهمیت هر کدام از این آزمون‌ها اشاره می‌شود:

آزمون واحدها:

اجزای این سیستم به دلیل امنیتی بودن باید حتما درست کار کنند و به همین دلیل باید آزمون در این سطح مفصل باشد.

آزمون سیستم:

این آزمون هم در مورد هر سیستمی مورد نیاز است و در صورت انجام مرحله‌ی اول با توجه به سناریوهای خاص از پیش تعیین شده می‌توان این مرحله را هم انجام داد.

آرمون یکپارچگی:

از آنجا که واسط‌های سیستم‌ها به دقت تعریف شده و این‌که تعداد سیستم‌ها و درنتیجه ارتباطات آن‌ها نسبتا زیاد نمی‌باشد این آزمون خیلی در مورد این سیستم اهمیت پیدا نمی‌کند.

آزمون پذیرش:

با توجه به ماهیت سیستم کاربران در مورد درون سیستم به نسبت دیگر سیستم‌ها کم‌تر آشنایی دارند، بنابراین این آزمون در صورت اجرا بیش‌تر ظواهر را می‌پوشاند نه خود سیستم. البته این نیز به اندازه‌ی پروژه بر می‌گردد، به طور مثال ممکن است کاربر برای این آزمون کارشناسی را استخدام کند.

[3] Unit Test
[4] System Test
[5] Integration Test
[6] Acceptance Test

سئوال ۲:

باید توجه داشت که در متن ابتدایی(قبل از مقایسه بر اساس موارد) برداشت از سیستم سنتی همان سیستم آبشاری است. در ضمن به دلیل ضمیمه کردن منابع برای ارجاع به آن‌ها بردن نام آن‌ها کفایت می‌کرد.
تفاوت اساسی بین روش‌های سنتی و روش‌های شیءگرا در این است که در روش‌های سنتی شکستن پروژه براساس فعالیت‌هاست حال آن‌که در روش‌های شیءگرا این شکستن بر اساس وظیفه‌هاست. در نتیجه‌ی این دید در هر مرحله‌ی پروژه یک جزء کامل نتیجه می‌شود که به آن Release گفته می‌شود؛ کامل از آن جهت که تمام فعالیت‌های مربوط به یک کار انجام شده و درستی آن اثبات شده است. این روش با توجه به توضیح داده شده Iterative نام دارد. در مقابل روش‌های سنتی با توجه به تقسیم‌بندی ذکر شده در پایان است که همه‌ی سیستم‌ها به اجرا می‌رسند و اگر در این مرحله مشکلی یافت شود هزینه‌ی رفع آن بسیار بیش‌تر از هزینه‌ی رفع آن در صورتی است که در یک مرحله‌ی از روش شیء گرا کشف شود. برای اطلاع بیشتر به فصل دوم کتاب UML Distilled نوشته‌ی مارتین فاولر که در سی دی ضمیمه موجود است مراجعه شود. با این وجود می‌توان هر کدام از موارد آنالیز، طراحی، پیاده‌سازی، تست و نگهداری را نیز جداگانه با هم مقایسه کرد:

۱. آنالیز
آنالیز شیء گرا نسبت به آنالیز سنتی آموزش بیشتری می‌خواهد و برای یک متخصص آنالیز سنتی سخت است که از روش قبلی خود به روش جدید مهاجرت کنند[2].

۲. طراحی
به دلیل پیچیدگی بالای طراحی شیء گرا قابلیت کنترل آن پایین‌تر می‌آید و در نتیجه احتمال طراحی بد بالاتر می‌رود.[1]
طراحی شیء گرا نسبت به طراحی سنتی آموزش بیشتری می‌خواهد تا به سطح طراحی خوب برسد[1].

۳. پیاده‌سازی
روش شیء گرا مقدار کد کمتری نیاز دارد[3].

۴. آزمون
روش‌های شیء گرا امکان آزمون جعبه‌ی سیاه[۷] را بهتر فراهم می‌کنند[4].

[7] Black Box Testing

۵. نگهداری

در نگهداری سیستم‌های شیء گرا اگر برنامه‌نویس تجربه‌ی کمی داشته‌باشد توانایی نگهداری بیشتری در مقایسه با سیستم‌های سنتی دارد[1].

اگر از قوانین شیء گرایی برای طراحی خوب استفاده شود. امکان تغییر به دلیل قابلیت فهم بیشتر سیستم افزایش می‌یابد[1].

[1] An Experimental Comparison of the Maintainability of Object-Oriented and Structured Design Documents.pdf
[2] Object Oriented and Conventional Analysis and design methodologies comparison and critique.pdf
[3] Object-oriented vs. waterfall software development.pdf
[4] Testing of ObjectOriented Programming.pdf

در نهایت جدولی برای مقایسه‌ی روش‌های رویه‌ای و شیء گرا ارائه می‌شود که این جدول در فایل OO.doc ضمیمه قابل مشاهده است.

Element	Procedural	Object Oriented
Basic unit is defining characteristic	Algorithm, code is all how to do it	Object, an encapsulated abstraction
Design focus	How its done	Responsibility allocation
Major artifact	Usually none, sometimes Flow Chart	Various Models, especially Object Model
File size	Large, I heard of a 10,000 line file	Small, my goal is 200lines max
Real world counterpart	A letter with instructions on what to do	Entity who responds to requests
Scalability	Medium	Infinitely high
Performance	High	Medium
Engine complexity	Medium	High
Model size	Larger	Smaller
Richness of design artifacts (models)	Low	High
Ease of learning	Easier	Harder, unless no procedural habits
Ease of reuse	Very low unless table driven	High

Ease of extensibility	Medium	High
Ease of maintenance (defect fixes)	Low	High if good design
Ease of tiny system	High, especially scripting languages	Low
Ease of distributed system	Low	High
Ease of Artificial Intelligence	Impossible	Medium for average OO language
Ease of Unit Testing	Low unless designed for it	High especially if designed for it
Appropriate for event driven system	Low	High
Modularity	Low due to large files, tight coupling	High
Encapsulation	Tends to be low, can be high	Much easier
Appropriateness for CPU bound algorithm	High, since procedural is algorithm oriented	Low until a miracle occurs
Ease of understanding, self documenting	Low, gotta chase a method call to find out what it is doing	High since all you have to know is the interface and understand the model. Inheritance also makes understanding what methods are available easier.
Data complexity handling	Anything can be done	But is easier here!
Variable list explosion	Can easily happen	Tends to rarely happen
Development efficiency	Average, falls off with system size	With training, big improvement. Without training, worse than procedural.

در نهایت با توجه به بیانات مختلف در ادبیات به نظر می‌رسد که این حرف تا حدی درست است؛ هرچه به پایان کار نزدیک می‌شویم روش‌های شیء گرا ساده‌تر می‌شوند و به‌تر عمل می‌کنند و این در حالی است که در ابتدا روش‌های سنتی ساده‌تر و کاراتر هستند.