Реферат

по

Среди за разработка

на тема

Обектноориентиран анализ и проектиране



**Съдържание:**

[I. Увод 3](#_Toc405290587)

[II. Същност 3](#_Toc405290588)

[III. Видове методологии 4](#_Toc405290589)

[1. Обектно-ориентирано програмиране 4](#_Toc405290590)

[2. Обектно-ориентираният анализ 5](#_Toc405290591)

[3. Обектно-ориентираното проектиране 5](#_Toc405290592)

[IV. Сравнителна характеристика на методологиите 5](#_Toc405290593)

[1. Обектно-ориентираното програмиране 5](#_Toc405290594)

[a) Абстрактни типове данни 6](#_Toc405290595)

[b) Ограничаване на достъпа 6](#_Toc405290596)

[c) Модулност 6](#_Toc405290597)

[d) Йерархичност 7](#_Toc405290598)

[e) Типизация 7](#_Toc405290599)

[f) Паралелизъм 8](#_Toc405290600)

[g) Устойчивост 8](#_Toc405290601)

[h) Предимства на обектния подход 8](#_Toc405290602)

[2. Обектно-ориентирания анализ 9](#_Toc405290603)

[a) Определяне на ключовите абстракции 10](#_Toc405290604)

[b) Определяне на механизмите 11](#_Toc405290605)

[3. Обектно-ориентирано проектиране 12](#_Toc405290606)

[a) Идентифициране на класовете и обектите 13](#_Toc405290607)

[b) Идентифициране семантиката на класовете и обектите 13](#_Toc405290608)

[c) Идентифициране връзките между класовете и обектите 15](#_Toc405290609)

[d) Реализиране на класовете и обектите 15](#_Toc405290610)

[V. Софтуерни среди за разработка 16](#_Toc405290611)

[VI. Извод 18](#_Toc405290612)

# Увод

В първите дни на обектно-ориентираните технологии преди средата на 1990-те години, е имало много различни, конкуриращи се методологии за разработка на софтуер и обектно-ориентирано моделиране, често обвързани с доставчици на конкретен CASE[[1]](#footnote-1) инструмент. Основните опасения за този период не са били стандартните обозначения, съобразените срокове и технологичните напътствия, който намаляват комуникационна ефективност.

“Some of the well-known early object-oriented methodologies were from and inspired by gurus such as Grady Booch, James Rumbaugh, Ivar Jacobson (the Three Amigos), Robert Martin, Peter Coad, Sally Shlaer, Stephen Mellor, and Rebecca Wirfs-Brock.”[[2]](#footnote-2)

През 1994 г. тримата амигоса на рационалния софтуер (Rational Software) започват да работят заедно за да разработят единен език за моделиране - UML[[3]](#footnote-3). По-късно, заедно с Philippe Kruchten и Walker Royce повеждат една успешна мисия да слеят своите собствени методологии OMT, OOSE и Boochs, с различни познания и опит от други индустриални лидери в едно цялостно постоянно нарастващо ръководство и рамка за учене, най-добрите практики в индустрията за разработка на софтуер и управление на проекти, наречен RUP[[4]](#footnote-4). Оттогава RUP семейството е станало може би най-популярната методика и референтен модел за обектно-ориентиран анализ и дизайн.

# Същност

Обектно-ориентиран подход - всички обектно-ориентирани Case среди се базират на обектно-ориентиран подход, който определя техните основни цели и предназначения. Обектно-ориентирания подход е съвкупност от принципи, на които е подчинено цялостното разработване на КИС. Основни принципи:

* декомпозиране
* абстракция
* йерархичност
* ограничаване на достъпа
* и други.

Принципите на обектно-ориентирания подход се реализират от конкретни обектно-ориентирани компютърни методи. В широк смисъл методите обхващат (обобщават) натрупания практически опит в дадена практическа област на мисловна или физическа дейност. Те са целево ориентирани по дисциплина или практика.

История на възникване на обектно-ориентираните методи - хронологично на първо място възниква **методът на обектно-ориентирано програмиране**. След това **обектно-ориентиран метод на проектиране** и най-накрая **методът на обектно-ориентиран анализ**. Обектно-ориентираните методи за анализ и проектиране са тясно свързани помежду си. Обектно-ориентиран метод за анализ до голяма степен се явява подмножество на метода за проектиране. Ето защо тези два метода се обединяват в един общ обектно-ориентиран метод, чието предназначение е моделирането на КИС.

# Видове методологии

## **Обектно-ориентирано програмиране**

Методология за програмиране, която се основава на представяне на програмите във вид на съвкупност от обекти, всеки от които е реализация на определен клас, а класовете образуват йерархия на принципа на наследствеността.

В това определение могат да се отделят следните три части:

* обектно-ориентираното програмиране използва като конструктивни елементи обекти, а не алгоритми
* всеки обект е екземпляр на определен клас
* класовете са организирани в йерархия

Програмата ще бъде обектно-ориентирана само при спазване на горните три изисквания. В частност, програмиране, което не е основано на йерархични отношения, не е обектно-ориентирано, а се нарича програмиране на база абстрактни типове данни.

## **Обектно-ориентираният анализ**

В основата на обектно-ориентираното проектиране стои обектният подход. Обектно-ориентираният анализ е насочен към създаване на модели, по-близки до реалността, с използване на обектно-ориентиран подход. Това е методология, при която изискванията се формират на основата на класове и обекти, съставящи речника на предметната област. В резултат на обектно-ориентирания анализ се формират модели, на които се базира обектно-ориентираното проектиране.

## **Обектно-ориентираното проектиране**

Методология за проектиране, съчетаваща в себе си процеса на обектната декомпозиция и начин за представяне, както на логическия и физическия, така и на статичния и динамичния модели на проектираната система.

Това определение съдържа две важни части:

* Обектно-ориентираното проектиране води до обектна декомпозиция;
* Използва се многообразие от начини за представяне моделите, отразяващи логическата[[5]](#footnote-5) и физическата[[6]](#footnote-6) структура на системата.

Поддържането на обектно-ориентирана декомпозиция различава обектно-ориентираното проектиране от структурното проектиране.

Обектно-ориентираното проектиране от своя страна създава основата за окончателната реализация на системата с използване на методологията на обектно-ориентираното програмиране.

# Сравнителна характеристика на методологиите

* 1. Обектно-ориентираното програмиране

Концептуална основа на обектно-ориентирания стил е обектният подход. Той се характеризира със следните четири главни елемента:

### Абстрактни типове данни

Абстракциите, това са такива съществени характеристики на някакъв обект, които го отличават от всички други видове обекти, като по този начин точно определят особеностите на дадения обект от гледна точка на по-нататъшния анализ. Абстрахирането концентрира вниманието на системния аналитик върху външните особености на обекта и позволява да се отделят най-съществените особености в поведението от подробностите при осъществяването им. Обектът представлява модел на съществените страни на предметната област. При описване поведението на обекта се включват операции, които самият обект изпълнява над други обекти. Този подход концентрира вниманието върху външните особености на обекта. Пълният набор от операции, които могат да се осъществяват с други обекти, се нарича протокол. Протоколът отразява всички действия, които обектът може да изпълни и с които влияе върху други обекти, определяйки външното поведение на абстракцията от статична или динамична гледна точка. Трябва да се има предвид, че понятията операция, метод и член-функция са еквивалентни и означават едно и също нещо.

### Ограничаване на достъпа

Това е процес на защита на отделните елементи на обекта, без да се губят съществените му характеристики като цяло. По този начин се определят границите между отделните абстракции и се осъществява защита, както структурата на обекта, така и на реализацията на неговите методи. Практически това означава наличие на две части в описанието на класа: интерфейсна и локална. Интерфейсната част отразява външното проявление на обекта, създавайки абстракция за поведението на всички обекти от даден клас. Локалната част скрива от другите обекти всички подробности, които нямат отношение към взаимодействието с обектите.

### Модулност

Това е свойство на системата свързано с възможността за нейното декомпозиране на множество от тясно свързани модули. Това предполага разделяне системата на фрагменти, които могат да бъдат отделно компилирани, като между тях съществуват средства за предаване на съобщения. Стремежът трябва да бъде към построяване на модули, за да се обединят логически свързаните абстракции да се намалят взаимовръзките между модулите. По този начин принципите на абстрахиране, ограничаване на достъпа и модулността са взаимно допълващи се. Обектът определя явните граници на определена абстракция, а ограничаването на достъпа и модулността създават граници между абстракциите.

### Йерархичност

Значително опростяване при описване на сложни системи се постига чрез изграждане на йерархична структура от абстракции. Йерархията това е подредена система от абстракции. Наследяването означава такова отношение между класовете, когато един клас използва структурна или функционална част на един или няколко базови класове. Принципът на наследяването позволява да се опрости представянето на абстракциите.

Горните четири елемента са главни в смисъл, че ако някой от тях липсва, подходът няма да бъде обектно-ориентиран. Освен главните, обектният подход има три допълнителни елемента. Те са полезни, но не са задължителни.

### Типизация

Това е ограничение, поставено върху обектите от даден клас, което възпрепятства взаимно замяната на различните класове. Типизацията позволява да се изпълни описанието на абстракциите по такъв начин, че да се осъществят проектните решения на ниво език за програмиране. Обектните и обектно-ориентираните езици за програмиране могат да бъдат силно типизирани, слабо или не типизирани. Предимства на силно типизираните езици са:

* липсата на контрол на типовете може да доведе до труднооткриваеми прекъсвания в програмите по време на тяхното изпълнение
* декларирането на типовете опростява документирането на програмите
* много компилатори позволяват да се генерира по-ефективен код при явно определяне на типовете.

Концепциите за силна и статична типизация по своята същност са различни. Силната типизация има отношение към контролиране съответствието на типовете, а статичната – има отношение към времето когато имената се свързват с типовете. Статичното свързване означава, че типът на променливите и изразите не се изменя по време на компилация. При динамично свързване типът на всички променливи и изрази се определя само по време на изпълнение на програмата.

Полиморфизмът е свойство, при което определено име може да означава обект от всеки клас, произхождащ от определен базов клас. Полиморфизмът възниква при съчетаване принципите на наследяване и динамично свързване. Това свойство отличава обектно-ориентираното програмиране от повечето традиционни методи на програмиране с използване на абстрактни типове данни. Освен това полиморфизмът е важна концепция в обектно-ориентираното проектиране.

### Паралелизъм

Това е свойство на обектите да се намират в активно или пасивно състояние. Обектът обединява две концепции – всеки обект може да представлява отделен канал за управление. Такъв обект се нарича активен. За системите, построени на базата на обектно-ориентираното проектиране, реалността може да бъде представена като съвкупност от взаимодействащи обекти, част от които са активни и влизат в ролята на възли на независими действия.

### Устойчивост

Това е свойство на обекта да съществува във времето и/или в пространството. Всеки обект в програмата заема определено място и съществува в продължение на определено време. Устойчивостта е свързана не само с времето на съществуване на данните. В обектно-ориентираните бази от данни се съхранява не само състоянието на обекта, но и начина на интерпретация на класа от всяка друга програма трябва да бъде определен еднозначно.

### Предимства на обектния подход

Обектният подход принципно се различава от подходите, свързани с традиционните методи на структурния анализ, проектиране и програмиране. Той създава множество от съществени удобства, които при други условия не могат да бъдат осигурени. Най- важното е, че обектният подход позволява да се създават системи, които съчетават представените по-горе атрибути. Могат да се дефинират следните предимства, свързани с използване на обектния подход:

• той позволява в пълна степен да се използват възможностите на обектно-ориентираните езици за програмиране

• използването на обектния подход съществено повишава качеството на системата като цяло. Обектно-ориентираните системи са по-компактни. Това означава не само намаляване обема на кода на програмите, но и поевтиняване на проекта и по-голямо удобство при планиране на разработката

• той води до разработване на система, базирана на стабилни междинни състояния, което опростява процеса на внасяне на изменения. Това дава възможност системата да се развива постепенно и не довежда до пълно преработване в случай на съществени изменения в началните изисквания

• той е ориентиран към естественото възприемане на света от човека.

## Обектно-ориентирания анализ

В обектно-ориентирания анализ определянето на общите свойства на обектите позволява да се намерят общите ключови абстракции и механизми, което от своя страна води до по-прост проект на системата. Това дава възможност за изграждане "речника" на проблемната област. Класификацията е средство за подреждане на знания. Целта на класификацията е да се намерят общите свойства на обектите, като в една група се обединяват обекти, имащи еднаква структура и еднакво поведение. Класификацията помага да се изгради обобщена, специализирана и събирателна йерархия на класовете. Тя осигурява правилно построяване модулната структура на системата.

Класифицирането е итеративен процес. Последователният итеративен подход непосредствено определя процедурата за построяване йерархията на класовете и обектите при разработване на информационни системи. На практика обикновено за основа се взема някаква определена структура от класове, която постепенно се усъвършенства. Едва на по-късните стадии на разработка, когато вече е получен някакъв опит от използването на такава структура, може критично да се оцени качеството на получената класификация. Основавайки се на натрупания опит, може да се създадат нови подкласове от съществуващите вече или да се слеят няколко съществуващи класа в един. Възможно е в процеса на разработване да бъдат открити нови общи свойства, които по рано не са били забелязани, могат да се дефинират нови абстрактни класове.

Класификацията е изключително сложна задача, тъй като

• "съществуват толкова начина за разделяне на окръжаващия ни свят на обекти, колкото учени решават тази задача "

• разумната класификация изисква голяма творческа енергия и проницателност.

Исторически са се наложили следните три класификационни подхода:

♦ Класическо разпределяне по категории. При този подход наличието на свойства е основен критерий за сходството между обектите. Обектите могат да се разделят на не пресичащи се множества, определяйки за всеки обект, дали притежава конкретни свойства или не. Най-подходящият набор от свойства за такава класификация се характеризира с висока независимост на тези свойства едно от друго. Това обяснява популярността на набора от свойства, като размер, цвят, форма и материал. Тъй като тези свойства са независими, те могат да се използват в произволна комбинация. В по-общ смисъл не е задължително свойствата да могат да се определят само от измерими характеристики.

♦ Концептуално обединение. Това е по-съвременен вариант на класическия подход. Той е възникнал при опита за формално представяне на знанията. При този подход отначало се формират концептуалните описания на класовете и след това обектите се класифицират според описанието, образувайки класовете.

♦ Теория на прототипите. Съществуват някои абстракции, които нямат нито точни свойства, нито ясна дефиниция. При този подход, класът се определя от един обект-прототип и обектът може да се включи в класа при условие, че той по определен начин прилича на прототипа.

Необходимо е да се има предвид, че принципите на структурния анализ и проектиране са противоположни на принципите на обектно- ориентираното проектиране. Структурният анализ може да бъде полезен в процеса на обектно-ориентираното проектиране при условие, че системният аналитик не предпочита структурния тип мислене. Друга опасност при анализа се състои в това, че много аналитици чертаят диаграми на потоците от данни, които отразяват по-скоро проекта, отколкото модела. Много е трудно да се построи обектно-ориентиран модел, ако той по лесен и очевиден начин се поддава на алгоритмична декомпозиция. Затова се предпочита обектно-ориентирания анализ и анализа на предметната област като подготвителен етап към обектно-ориентираното проектиране. Това намалява риска проектът да се насочи към елементите на алгоритмичния анализ. Ако е необходимо да се приложи структурен анализ, не е необходимо да се съставят диаграми на потоците от данни, ако те започват да представляват проект на системата, а не съществена част от модела. Разумно е да се избягват елементите на структурния анализ, когато проектирането се намира на последния етап. Може да се използва продукт от структурния анализ само на високо ниво на абстракция, когато се обхваща модела на проблема.

### Определяне на ключовите абстракции

Ключовата абстракция е клас или обект, който определя част от речника на предметната област. Най-важната роля на ключовите абстракции се състои в това, че те определят границите на проблема. Определянето на ключовите абстракции включва два процеса: откриване и създаване на нови абстракции. Нови класове и обекти, които не са съществена част на предметната област, се създават в случай, че те са полезни инструменти при реализация на проекта. Най-общият начин при определяне на ключовите абстракции е намирането в проблема на абстракции, аналогични на съществуващите класове и обекти. Тъй като това е проблем на класификация, могат да се използват описаните по-горе класически начини на класификация.

Трябва да се определи някакъв набор от класове, които могат да бъдат ключови абстракции. Това означава, че системният аналитик трябва да зададе следните въпроси

• Как се създават обектите от съответния клас?

• Могат ли да се копират или унищожават обекти от съответния клас?

• Какви операции могат да бъдат изпълнени с тези обекти?

Ако отговорите на тези въпроси са неясни, то възможно е общата концепция да е неточна и по-добре е отново да се дефинират ключовите абстракции.

Определяйки новите абстракции, трябва да се намери тяхното място в контекста на вече съществуващите класове и обекти. Няма точно дефинирано правило при решаване на този проблем, както при проектиране “отдолу-нагоре” или “отгоре-надолу”. Някои автори потвърждават, че не е необходимо йерархията на класовете да се изгражда, започвайки от корена на йерархията и след това да се допълнят произхождащите класове. Често се създават няколко независими класа, осъзнават се общите им характеристики и се отделят в базови класове. Няколко такива преминавания по йерархията са напълно достатъчни за създаване проекта на системата - процесът на проектиране е последователен и итеративен.

Трудно е класовете веднага да се разположат на правилното ниво на йерархия. Понякога в построената класификация могат да се открият подобни класове и те могат да се придвижат нагоре в йерархията на класовете, по този начин се увеличава степента на разделяне на общите свойства. Аналогично могат да се намерят класове с толкова общи свойства, че това води до семантичен разрив между тях и техните подкласове. В двата случая се прави опит да се намерят зависимите и независимите абстракции.

### Определяне на механизмите

Ключовите абстракции описват модела на реалността. Терминът “механизъм” се използва, за да се опише някаква структура, с помощта на която обектите взаимодействат помежду си, осигурявайки необходимото поведение на системата. Проектът на класовете съдържа в себе си знания за поведението на класовете. Проектирането на механизмите съответства на решаването на задачата за взаимодействие на обектите.

Ключовите абстракции определят речника на предметната област, а механизмите определят същността на проекта. В процеса на проектиране системният аналитик трябва да отдели внимание не само на структурата на класовете, но и на това как обектите от тези класове взаимодействат един с друг. Ако аналитикът е разработил някакъв конкретен механизъм на взаимодействие, той се реализира чрез дефиниране методите на съответните класове. Това трябва да се извърши точно, в противен случай може да се получи голям брой независими обекти, изпълняващи свои функции без да се отчита взаимодействието с останалите обекти от системата.

## Обектно-ориентирано проектиране

Обектно-ориентираното проектиране е възвратен процес. Обединяването на понятията обектно-ориентирано и възвратно проектиране осигурява големи предимства. Еволюционният характер на процеса на проектиране дава възможност да се използва този модел, който е най-необходим в момента за решаване на поставената задача. В началото може да се състави схема на структурата на класовете чрез диаграмата на класовете, след това да се разработят някои механизми, използващи тези абстракции, и да се документират в съответните диаграми на обектите. Проектирайки подробностите на тези механизми, отново може да се осъществи връщане към структурата на класовете, установявайки протоколите за всеки клас, което може да доведе до реорганизиране на наследствената схема, така че по-добре да се използва приетата общност от изразителни средства. При завършване процеса на проектиране се получават редица диаграми, съответстващи на различни, но точни и съгласувани модели на системата, получени в резултат на еволюцията на предишните устойчиви, макар и не толкова подробни модели.

Стъпаловидният итеративен процес на обектно-ориентираното проектиране се различава от традиционния процес на разработване на програмно осигуряване, при който лавинообразно нараства сложността в процеса на създаване на системата. Традиционният подход има редица неотстраними недостатъци и нарушава много инженерни принципи. Той задължително трябва да предполага еволюционност в развитието. Подобен подход съответства на спирален модел в развитието на програмното осигуряване. Обектно- ориентираното проектиране не е процес, който започва с определяне на изискванията и завършва със съставяне схема на използване на програмната система.

Обектно-ориентираното проектиране съответства на следната последователност от събития:

### Идентифициране на класовете и обектите

Първата стъпка се състои от две действия - създаване на основните абстракции, принадлежащи на проблемната област (най-важните класове и обекти), и разработване на основните механизми, осигуряващи необходимото поведение на обектите, които от своя страна осигуряват функционирането на системата. Това изисква анализиране на предметната област, усвояване на основните понятия и теоретични положения. Определят се границите на абстракциите. Предполага се, че тези граници могат да се променят при дефиниране общите свойства на абстракциите и да се модифицират при формиране механизмите на взаимодействие. Резултат от първата стъпка е съставяне на списък от имена на основните класове и обекти, така че тези имена да носят определено смислово съдържание в съответствие с мястото и функциите на класа или обекта в системата. Някои елементи в списъка могат да бъдат класове, други - обекти, а трети - свойства на обектите. Резултат от тази стъпка може да бъде запълване шаблоните на съответните класове и обекти, т.е. формално определяне смисъла на отделените абстракции и механизми. Основната цел при съставянето на такъв списък е изработването на терминология за общуване между участниците в анализа и проектирането на системата.

### Идентифициране семантиката на класовете и обектите

На втората стъпка се определя съдържанието на класовете и обектите. Всеки клас се анализира от гледна точка на перспективите на взаимодействието му с другите класове и се определят механизмите на взаимодействие между класовете и обектите. Дефиниране съдържанието на протокола за всеки обект е по-сложна задача. По тази причина от определен момент процесът на обектно- ориентирано проектиране става итеративен. Съставянето протокола на даден обект може да доведе до загубване съдържанието на друг обект. При това не се поставя въпрос само за съществуването на дефинираните при първата стъпка абстракции, а може да се стигне до преместване границите между тях. Затова е необходимо всеки обект да се опише, като се определи цикъла на неговото функциониране - от създаването му до изтриването и особеностите на неговото поведение. Резултатите от тази стъпка отразяват стъпаловидния характер на обектно-ориентираното проектиране. За всеки клас и обект се завършва с попълването на шаблоните, създадени на първата стъпка. Затова е необходимо по най-добър начин да се фиксират всички статични и динамични свойства на всяка основна абстракция и механизъм. Могат да се създадат прототипи на отделни части на проекта с цел анализиране на неговото текущо състояние и оценяване на различните подходи при решаване на тези подзадачи, които се намират в зоната на повишения риск.

**Общата схема на шаблона за описание на класовете може да бъде представена по следния начин:**

ИМЕ НА КЛАСА:

ТИП НА КЛАСА:

ВИДИМОСТ:

НАСЛЕДЯВА ОТ:

БАЗОВ ЗА:

ПРИЯТЕЛСКИ НА:

ИМА ПРИЯТЕЛСКИ:

ОПИСАНИЕ ХАРАКТЕРНИТЕ ОСОБЕНОСТИ НА КЛАСА:

ЧЛЕНОВЕ ДАННИ:

име:

съдържание:

тип:

ФУНКЦИИ-ЧЛЕНОВЕ (МЕТОДИ):

<прототип>

<общо описание действието на функцията>

<структура и тип на параметрите>

<резултати>

<особености при използването>

ОБЕКТИ НА КЛАСА:

име:

предназначение:

устойчивост:

ВРЪЗКИ МЕЖДУ ОБЕКТИТЕ:

### Идентифициране връзките между класовете и обектите

Третата стъпка може да се разглежда като продължение на предната. Тук точно се определят механизмите на взаимодействие вътре в системата. Трябва да се определят отношенията между основните абстракции - използване или наследяване. Необходимо е да се определят статичните и динамичните свойства на обектите и техните механизми. Трябва да се разработи структурата на класовете, което може да доведе до необходимостта от реорганизация на системата на класовете. Трябва да се разработи структурата на организацията на обектите, което може да доведе до обобщаване механизмите на взаимодействие, включени в проекта. Необходимо е да се определят границите на видимост между класовете и обектите. Това ще се отрази върху архитектурата на модулите в системата и може да доведе до усъвършенстване на протоколите за класовете, създадени на по-ранните етапи. На този етап завършва създаването на логическия модел на системата и може да се започне с построяването на диаграмите на модулите. На този етап може да се продължи с разработването на нови прототипи на отделните подзадачи, както и с редактирането на старите прототипи с цел постигане на дефинираните функционални възможности на цялата система. Действайки по такава схема винаги ще съществува някакъв работен вариант на системата.

### Реализиране на класовете и обектите

Четвъртата стъпка не винаги е последна. Трябва да се решат още две основни задачи: да се вземат решения относно включването в системата на конкретните класове и обекти и да се разпределят класовете и обектите по отделните модули. Всеки клас и обект се разглежда в дълбочина с най-големи подробности, така че да се намерят най-подходящите начини за реализация на неговите свойства. Дори ако абстракциите и механизмите са сравнително прости, от този етап може да се извърши връщане към първата стъпка и отново да се проходи целия път, разглеждайки детайлно съществуващите класове и отделните модули. По този начин процесът на проектиране се повтаря, като вниманието се фиксира на по-ниските нива на абстракция. Означава ли това, че обектно- ориентираното проектиране е пример за проектиране отгоре-надолу? Отговорът е “не”. На практика се проектират по-високите нива на абстракция и в тях се включват тези класове и обекти, които явно съответстват на терминологията на предметната област. На всяка стъпка често се налага да се извършва обръщение към абстракции от по-ниско ниво и свързаните с тях механизми. На тази стъпка се реализира конкретното взаимодействие между всеки клас и обект.

Обектно-ориентираното проектиране е постепенен процес - идентификацията на класовете и обектите налага модифициране на семантиката и връзката между съществените класове и обекти. В същото време това е итеративен процес - използването на класовете и обектите често довежда до необходимостта от разработване и създаване на нови класове и обекти, което улеснява процеса на проектиране.

Обектно-ориентираното проектиране започва със създаване на класовете и обектите, формиращи речника на предметната област. Процесът на проектиране ще завърши, ако се установи, че повече няма ключови абстракции и механизми, които не са включени в модела, и прилагайки ги при създаването на класовете и обектите може да се осигури очакваното поведение на системата.

# Софтуерни среди за разработка



**Rational Rose Modeler** - осигурява набор от възможности за разработването на софтуерни приложения. Тя позволява на архитекти и дизайнери да произвеждат UML модели за софтуерни архитектури, бизнес нужди, активи за многократна употреба и комуникация на управленско ниво. Предлага общ инструмент за моделиране и език, за опростяване на работната среда и да позволи по-бързо създаване на качествен софтуер. **По-бързо разработване на приложения** – функции за визуално моделиране на средата, която ни позволява по-бързо разработване. **Интегрира дизайна на приложението със средата** – унифицира екипа като им доставя единен UML модел за изпълнение и нотация.

http://upload.wikimedia.org/wikipedia/commons/thumb/e/e3/Borland_logo.svg/150px-Borland_logo.svg.png

**Together Borland** – инструмент за визуално моделиране, който позволява на екипите да анализират, проектират, имплементират софтуерните архитектури и да останат в синхрон, като споделят общо визуално възприемане. Средата подобрява възприемането на цялостните системи, за да ни помогне при възприемането на изискванията възможно най-прецизно.



**MagicDraw** е първокласен бизнес процес, архитектура, софтуер и система за моделиране инструмент, който подържа работа в екип. Проектиран за бизнес анализатори, софтуерни анализатори, програмисти, QA инженери, този динамичен и гъвкав инструмент за развитие улеснява анализа и проектирането на обектно-ориентирани системи и бази данни. Тя осигурява най-добрия механизъм за генериране на код (с пълена двупосочена подръжка за Java, C ++, C #, CL (MSIL) и CORBA IDL езици за програмиране), както и схема на база данни за моделиране, генерира DLL.



**Enterprise Architect** е изключителен инструмент с високи крайни възможности и богат набор от функции, за да се подпомогне управлението на информация и иновации в днешната сложна и трудна среда. На цена значително по-ниска, отколкото конкурентни инструменти. Enterprise Architect предлага на вашия екип и компания възможност да се възползва от висок клас за моделиране и функции за изграждане на една изгодна цена. Enterprise Architect предлага пълен жизнен цикъл на моделиране за: бизнес и IT системи, системно и софтуерно разработване, real-time and Embedded разработване.

# Извод

Всички от изброените среди са „Close source“. Rational Rose Modeler е интегриран в Eclipse, като с него могат да се генерират както UML така и MDA. За разлика от Rational Rose Modeler, Together Borland е интегриран освен в Eclipse и в Microsoft Visual Studio, също така подържа набор от шаблони. Може да генерира код на Java, C++ и CORBA. MagicDraw разширява тези възможности, но не се интегрира в MS Visual Studio, но за сметка на това се подържа в NetBeans. Освен това може да генерира и XML, разширява и езиците като допълва C#, DDL, XML, Schema. Усъщестява двупосочна връзка с езиците, като те могат да бъдат и импортирани в средата. Enterprise Architect предоставя подобни възможности, като MagicDraw, със съвсем малки допълнения, като той вече се подържа в MS Visual Studio. При него е възможен експорт и импорт и на езици, като PHP, Python, Delphi, Visual Basic, ActionScript.

По-големите възможности на MagicDraw и Enterprise Architect, са за сметка на по-висока цена, подходящи за по-сложни и по-големи проекти с по-висок бюджет, спрямо останалите две среди. В зависимост от изискванията на проекта, се избира и съотвентата среда.

**Източници:**

* <http://office.microsoft.com/>
* <http://www-03.ibm.com/>
* <http://www.borland.com/>
* <http://en.wikipedia.org/>
* <http://www.referati.org/>
* <http://www.umsl.edu/>

1. Computer Aided Software Engineering [↑](#footnote-ref-1)
2. <http://en.wikipedia.org/wiki/Object-oriented_analysis_and_design> [↑](#footnote-ref-2)
3. Unified Modeling Language [↑](#footnote-ref-3)
4. Rational Unified Process [↑](#footnote-ref-4)
5. Структурата на класовете и обектите [↑](#footnote-ref-5)
6. Архитектурата на моделите и процесите [↑](#footnote-ref-6)