Теми по Софтуерни Технологии на специалност Компютърни Науки
Лектор: Нели Манева
Летен семестър на 2010/2011г

Съдържание

1.Основни понятия - програмен продукт, софтуер. Характеристики.

Характеристиките на софтуера са следствие от особеностите на потребителите (интерфейс надеждност, лекота на усвояване и др.)

Проблеми и характеристики на програмите:
- всяка програма би трябвало да може да бъде поправяна, разширявана, подобрявана от своя автор или от друго квалифицирано лице;
- неясно дали други хора освен автора, ще ползват програмата; днес той, може никога да не ползва своето творение след написването му;
- последното обаче веднага води до необходимостта от преносимост на програмата от един компютър на друг.

Програмата е последователност от инструкции, които, когато бъдат декодирани от компютър (или от компютър и транслираща програма), водят до решаването от страна на компютъра на зададена задача.

Програмният(софтуерният) продукт е програма или съвкупност от взаимодействащи програми, записани върху технически носител и придружени от съответна документация.

Софтуерът - това са компютърни програми, процедури, правила и евентуално придружаваща документация, както и данни, отнасящи се до функционирането на компютърната система (днес общо понятие за програми и/или програмни продукти).

Характеристики на софтуера:
- необходими ресурси – време/хора; софтуерните продукти, в сравнение с много други, се отличават невероятно много по отношение на влаганите в тях ресурси, от където следва и цената им.
- **абстрактност на софтуера** - софтуерният продукт не може да бъде почувстван с нито едно от петте човешки сетива. С компютъра това не е така - той може най - малкото да бъде видян или пипнат. Софтуерът е на много високо ниво на абстрактност.
- **уникалност на софтуерното производство** - основните усилия при производството на програмен продукт са преди появата на първото работещо копие, при колите за всяка кола се влага много труд; - ако има грешка в даден програмен продукт, то тя се отнася до всички негови копия, при колите е малко вероятно; - когато програмният продукт излезе от употреба, все повече това става относително едновременно и поради една и съща причина - обикновено идва нова версия, с някоя и друга възможност повече и най-вече - съобразена с променения хардуер, при колите причините за „умирането" на даден екземпляр са индивидуални(катастрофа, повреди, възможности на собственика...).
- **мултидисциплинарност на разработването на софтуер** - Като изключим специфичните инструментални средства (компилатори, свързващи редактори, средства за тестване на програми, ОС), разработвани от изключително от софтуеристи, не е възможно програмният продукт да бъде създаден само от програмисти. Трябва много повече групи специалисти(статистици, застрахователи, икономисти). При такова сътрудничество почти винаги възникват проблеми на общуването, породени най-вече от различния тип на мислене и различните професионални езици на участниците.
- **специфични проблеми на надеждността** - абсолютно безпогрешен софтуер няма, причина - прекалено много са възможните пътища през програмата, допустимите (и недопустими) съвкупности от данни, действията от страна на потребителя, които наистина не могат да бъдат изцяло предвидени. Има разработени и технологии за тестване и отстраняване на грешки, които обаче поне от теоретическа гледна точка не могат да гарантират пълна липса на грешки. Ако програма се докаже, че не прави грешки, при определени условия, нито едно копие на тази програма няма да направи грешка при тези условия.
- **рискове** - неизпълнени изисквания, неизпълнен срок на доставка - опитът тук е от решаващо значение - „90% синдром" - 90% от програмния продукт се завършва за определения период от време, а за останалите 10% се оказва, че е необходимо в най-добрия случай още толкова време.
- **софтуерът - средство, а не цел** - софтуерът е средство което задейства (управлява) някаква система, а тя самата довежда до искания резултат(цел). Пример: потребителят иска да получи документ в определена форма и с определено съдържание, което става с помощта на компютъра, задействан от текстообработваща програма. В случая документът е цел от първи ред. Компютърът е средството, с което се достига до резултата и поради това е от втори ред, а текстообработващата програма е от трети ред.
- **Мащаби на производителността** - Едва ли има друга област, в която да се забелязват такива огромни

разлики в производителността на отделните специалисти от един и същи тип, особено сред програмистите, като основни изпълнители.

2.Софтуерни технологии – дефиниции, предмет, особености.

„Software engineering" – конференция на НАТО 1969г., „Софтуерни технологии", а не „Софтуерно инженерство“ на български.
„Софтуерната криза“ - няколко години преди горе споменатата конференция са се появили компютрите от т.н. 3-то поколение. В този период не било ясно как да се произведе софтуер с толкова голяма сложност и мащаб, който да отговаря на новите компютърни мощности . Така възниква естествената необходимост от специална дисциплина, която да се опита да помогне за преодоляването на софтуерната криза.
Проблеми отбелязани в доклада от 1969:
- липса на разбиране на изискванията към системите от страна на потребителите;
- големи разлики между предварителните оценки за разходи и време и реално изразходваните поради липса на адекватни методи за оценка;
- бързо нарастване на обема на софтуерните системи;
- склонност на програмистите да пишат оптимални и безупречни във всяко отношение програми, като загърбват спешността на чисто практическите нужди на потребителите;
- бързо нарастващи нужди от програмисти и хроничен недостиг на добре обучени програмисти с добри практически умения;
- лоша комуникация между отделните разработващи даден проект групи, обмен на излишна или недобре координирана информация;
- трудности при постигане на достатъчна надеждност (липса на грешки) на големите софтуерни системи;
- разходи за съпровождане, които често превишават разходите, направени за разработването на софтуерната система.
Определение: Софтуерните технологии представляват систематичен подход към разработването на качествен софтуер, както и към неговото предлагане на пазара, експлоатация и съпровождане.
Цели - всеки програмен продукт следва да може да бъде:
- лесно съпровождан - в него трябва да могат относително лесно да се внасят изменения и подобрения, както и лесно да се отстраняват грешки. За да може това да се осъществява, е необходимо програмният продукт да бъде добре документиран.
- надежден- грешките в него трябва да сведени до възможния минимум, а продуктът да изпълнява очакваните от потребителя функции.
- ефективен - оптимално експлоатиране на ресурсите на компютъра, преди всичко по отношение на икономичното използване на ОПамет и достигане на голяма бързина на изпълнение.
- удобен и лесен за усвояване потребителски интерфейс - все по-актуално,поради нарастването на броя на потребителите и разширяването на спектъра им. Не са малко случаите, когато поради недобре проек тиран или реализиран интерфейс, много от възможностите на продукта остават неизползвани от повечето потребители. Което води до намаляване на броя на продажбите му.
- цена - минимизиране разходите по разработването, но и особено на съпровождането, доколкото последните са неочаквано големи.
Не е възможно да бъдат постигнати едновременно най-добри стойности за всички изброени атрибути. Просто защото някои от тях направо си противоречат. Веднага се вижда например, че ако интерфейсът се направи прекалено „дружелюбен" (естетичен, ергономичен, лесно разбираем, снабден с много помощни указания и пр.), ефективността на програмния продукт ще пострада. Нещата се усложняват и от факта,

че цената на подобренията в много случаи не расте линейно, т.е. малки подобрения в определен атрибут могат да изискват значително (дори експоненциално) нарастване на вложените ресурси, следователно и на цената.
Всъщност, погледнато най-общо, задачата на софтуерните технолози е да покаже как да се произвежда софтуер, който да притежава в оптимално съотношение упоменатите характеристики. Това е смисълът
на преодоляването на софтуерната криза, станала причина за появяването на тази дисциплина. Общо е обаче мнението, че софтуерната криза още не е преодоляна. Независимо от значителните постижения, подобряващи методите и технологиите на разработване на софтуер или довели до създаването на мощни инструментални средства, изглежда че нуждите от софтуер нарастват по-бързо, отколкото се подобрява производителността на разработчиците на софтуер.

Наука и практика:
Докъде всъщност е науката в създаването на софтуерни продукти? В много случаи софтуеристите не могат без формални методи(проба-грешка) или биха постигнали по-лоши резултати без тях. Във всеки случаи с такива методи трябва да се процедира, когато задачата е добре формулирана и разбрана или пък е с рутинен характер. Когато обаче е поставен сложен проблем или пък такъв, изискващ творческо решение, евристичният подход е по-добрата възможност.

3. Жизнен цикъл на ПП. Модели на ЖЦ, класификация.

ЖЦ на ПП обхваща целия период на неговото създаване и използване: начало - моментът на възникване на идеята за създаването му; физически край - моментът, в който се преустановява използването на последното копие на ПП; логически край – не са необходими никакви усилия по съпровождането му.
Цел на конструктивните дисциплини(моделите)е: какво правим, как го правим, в какъв ред го правим.
Какви свойства има моделът:
- Изоморфизъм - на всеки обект от реалността съответства обект от модела и на всяка операция върху един или няколко обекта от реалността съответства операция върху съответните обекти от модела.
- Абстрактност - не е възможно да се постигне пълен изоморфизъм – сложната. В сила е и за ПП и ЖЦ - не е възможно да се построи модел, който да отрази всичките им особености.
- Език за описание - езика на някой дял от математиката, графични, диаграмни, блокови езици, естествен език, повече от един език. Моделите на ЖЦ на ПП обикновено се отнасят към последния случай.
Приложност на модела –определя се от адекватността между модела и реалната система. Валидност на модела – определя се от вътрешните характеристики и способността му да дава адекватна информация. Полезност на модела – определя се от потребителя.
Следствията от един успешен модел са:
- Методологически - моделът дава отговори на общи въпроси - следва ли съпровождането да бъде разглеждано като изцяло отделна дейност, какъв тип специалисти да го извършват, възможно ли е управлението на мерките по осигуряване на качеството да се разглеждат отделно от цялостния производствен процес, следва ли за отделните извършвани функции да се правят отделни планове и кой да ги прави, кой да ги обсъжда и кой да ги одобрява и т.н.
- Организационни - как да се комплектова и ръководи екипът-разработчик, какъв тип йерархия да се възприеме (ако се приеме йерархична орг. структура), дали всеки проект да има собствен ръководител и ако да докъде да се простират управленските му пълномощия, допустимо ли е даден специалост или група от специалисти да работи едновременно по повече от един проект и тн.

- Технологически - какъв тип спецификации и техники на програмиране са препоръчителни, какви метрики да се ползват за опр. на качестовото на разработвания продукт, възможно ли е ползването единствено на обектно-ориентиран подход от началото на ЖЦ до завършване на тестването на ПП.

Пълни Едномерни Модели:

- Стандартни хронологични модели - основа е развитието на ПП във времето, основен обект **фазата**.

* **Модел на Фрийман:** 1. Анализ на необходимостта от създаване на ПП 2. Специфициране на ПП:създаване и описване условията за реализацията му.Оформяне на „Спецификация на ПП"; 1. Проектиране архитектурата на софтуерната система (външен проект) 2. Детайлно проектиране
(вътрешен проект); Реализация на ПП 1. Програмиране 2. Тестване; Поддържане: осъществяване на връзки с потребителя за инсталиране на ПП, обучение, регистриране на грешки.

***Модел на Метцгер:** Определяне на софтуерната система; Проектиране; 1. Кодиране (програми-
ране) 2. Вътрешно тестване (по модули и системно) 3. Приемни изпитания от независими експерти
- може ли системата да се пусне на пазара; Инсталиране Експлоатация на ПП.

* **Модел на Боем:** 1. Определяне на изискванията към ПП 2. Определяне на изискванията към средата
на използване и на разработване; 1.Предварително проектиране 2.Детайлно проектиране; 1Програмиране
2. Тестване и експериментиране (експериментално внедряване, проверка на работоспособността на ПП); Използване на ПП в реални условия.

- **Функционален модел(USEIT)** - Хамилтън и Целдин - най-съществения признак е типът извършвани дейности, група от приличащи си дейности - функция. Управление сочи към всички дейности разгледани по доло – въплъщава идеята за непрекъсната управляемост и контролируемост на процесите.

*Дефиниране на проблема - би следвало да дойде от взаимодейгвието на потребителя с разработчика.

*Анализ - представлява изясняване на проблема и избиране на решение в най-общия му вид.

* Разпределение на ресурсите - в другите модели не се среща. В нея се включва планирането, разпределението на всякакви ресурси, необходими за изпълнението.

* Документиране – резултат - различни видове документация - вътрешна , потребителкса, съпровождаща.

* Изпълнение - реализацията на планираните и фиксирани във вече изброените функции дейности.

- **(Хронологичен)Разклонен модел на Фокс** - особено внимание на дейностите по съпровождането, непрекъснатото усъвършенстване на веднъж разработения ПП. (Фиг. Разработване сочи към 2 успоредни прави – използване, продължаващо разработване(съпровождане).

* Разработването (деф. на изискванията, проектиране, написване и сглобяване на програмите, тестване, документиране) - най-тежката фаза. Безпроблемното използване зависи силно от вложените усилия тук.

* Съпровождането (добавяне и усъвършенстване на функции, дори свързани с промени в хардуера, отстраняване на грешки) - може да отнеме при големи ПП толкова усилия, колкото и разработването.

Възможни модификации - патологични ЖЦ -някоя от фазите не е застъпена или пък непрекъснатостта, особено на връзката между разработването и съпровождането е разкъсана(различни колективи).

Пълни Многомерни Модели:

- Тримерен модел на Питърс-Трип – три аспекта(време, логика, формализъм).

* Време - вече срещаната хронологическа компонента, отразява развитието във времето на ПП. 4 фази - анализ на системата (ПП), проектиране, реализация и експлоатация.
* Логика - категоризират се дейностите, които се извършват през отделните фази, дефинирани в първото измерение. Те са: определяне на проблема и оценъчно проектиране, синтез на системата (ПП), анализ на системата, оптимизация, вземане на решение, планиране на действията. Смисълът на синтеза е да се създадат няколко алтернативни идейни проекти, на анализа - да се идентифицират и систематизират изводите и оценките на всяка от алтернативите, а на оптимизацията - да се подредят алтернативите по един или повече критерии. След тази дейност остава само да се вземе решение за избор на алтернатива.
* Формализъм - включва необходимите за разработването формални модели на ПП. Започва се с чисто мислен модел, отразяващ първоначалните идеи на разработчика за ПП. Този модел се преобразува в структурен. Препоръчително е той да се базира на някакъв математически формализъм от рода на графи (евентуално дървета, мрежи и пр.). Последният модел е лингвистичен и той придобиваформата на последователност от текстови оператори с определен синтаксис и семантика - проект на ПП на определено ниво, описан формално, било ПП, реализиран във вид на определен програмен език.

Частични Модели:

Липсват части от ЖЦ на ПП. Reuse техника - използване на вече създадени компоненти от други ПП.

Ниво (3)(2)(1) – оценка на изискванията(3) -> използване на заготовки(3), настройка(2), изготвяне на скица(1) -> сглобяване(3); настройка(2), изготвяне на скица(1) -> проектиране и изготвяне на заготовки(0) -> оценка на изискванията(3). Във в с и ч к и случаи прилагането на модела започва с оценка на изискванията. В зависимост от резултата първоначално става насочване към ниво 3 или 2.Насочването към ниво 3 означава, че разработването на новия ПП е възможно изцяло на основата на заготовки.

Пълен Хронологичен Модифициран Каскаден модел:

Модификацията се състои в това, че отделните фази се припокриват частично във времето. Това е една стъпка на подобрение на адекватността на модела на ЖЦ на ПП по отношение на стандартните модели.

Изследване > деф. на изискванията > външно проектиране > вътрешно (детайлно) проектиране > програмиране > тестване > използване.

Пълен Хронологичен Модифициран Прототипен модел:

Използва се твърде интензивно в практиката. Реализира умалена версия (прототип) на крайния ПП, върху която се правят експерименти за установяване на съответствие с изисквания на потребителя. В зависимост от получените резултати се правят корекции с различна степен на връщане към предходните стъпки. След окончателното доказване на правилността на функциониране на прототипа и съответствието му с изискванията за разработката се пристъпва към реализацията на крайния ПП. Предимствата на такова прототипиране са очевидни те спестяват различни ресурси на разработчика. Недостатъкът е липсата на гаранции, че прототипът ще съдържа всички съществени свойства на крайния ПП. В най-лошия случай може да се окаже, че поради неумело създаден прототип, крайният ПП е разработен със значително повече ресурси. Анализ на системата > дефиниране на изискванията > създаване/промяна на прототип > оценка на прототип > предните 3, реализация > тестване > използване.

Пълен Еволюционен модел:

Спецификации – създават яснота, но и никой не е в състояние от самото начало да фиксира своите изисквания. Еволюционен модел - опитва да интегрира специфициране, проектиране и реализация.

Етапите в един еволюционен процес: - Формулиране първи вариант на изискванията към ПП. - Разработване на ПП възможно най-бързо на основата на така формулираните изисквания. - Оценяване на ПП съвместно с потребителите или възложителя и внасяне на изменения по функционалността;

Този модел води до ПП, съответстващ в най-висока степен на нуждите на потребителя.
Проблеми:
- Тъй като се фокусира върху крайния потребител, моделът не дава достатъчно приоритет на организационните аспекти на разработването.
- Постоянното изменяне но време на разработката влошава структурата на ПП до такава степен, че съпровождането му може да стане изключително трудно и скъпо.(евентуално пренаписване).
- Цялостният процес няма желаната прозрачност и ръководството на проекта не е в състояние да оцени развитието му. Не се прилага при големи проекти, най-тежкият проблем е управлението.
Пълен Спирален Модел: Интергриране на еволюционен модел с изискванията на управлението - спирален модел. В този модел разработването се движи спираловидно основавайки се на спецификации. На всеки сегмент на спиралата се прави оценка на риска и се провежда редукция- създават прототипи с оглед редуциране неопределеността на спецификациите, установяване на потребителски интерфейс и тн. Това води или до обогатяване на спецификациите и до внасяне на по-голяма определеност в техен ПП, или пък до еволюция на прототипа до крайния ПП. Допустимо е различни компоненти – разработени с различни методи. Еволюцуонния и спираловидния модел са най-подходящи за разработване на интерактивни и авангардни системи, с трудно формулируеми изисквания в началния момент.

4.Модел на Гънтър за жизнения цикъл. Фази и функции, съответствие.

В групата на двумерните модели. Две измерения – хронологично и функционално. Съгласно хронологичното измерение, разработването на ПП преминава през 6 етапа, наречени фази. По време на тези фази с различна интензивност се осъществяват 7 функции. Фиг.3.1 на отделен пищов. **Фази:**
Изследване: Начало – началото на ЖЦ за ПП, моментът на възникване на идеята. **Същност** – опр. се предназначението, основните функции и изискванията към разработвания ПП(маркетингово проучване).
Резултат – трябва да бъдат формулирани всички основни изисквания към ПП, всеки подобен резултат се оформя в писмен документ, в случая той се нарича Съглашение за Изискванията (СИ). **4-10 седмици.**
Анализ на осъществимостта: 1-10weeks Начало - моментът на назначаване на ръководител на проекта. **Същност** - може ли да се създаде ПП; анализ на – **техническа осъществимост**(достъпен хардуер, както за осъществяване на разработката така и за ползване на готовия ПП; анализират се и другите необходими технически средства); **икономическа осъществимост** (анализират се и се оценяват в цената на разработване, цена на която би се продавал ПП и цената за експлоатация на ПП, от къде идват парите); **експлоатационна осъществимост** (анализират се преимуществата и недостатъците на замисления ПП от гледна точка на потребителите му – спец. квалификация, бързина на обработките); **пазарна осъществимост** (на основата на маркетингово проучване се прави опит да се прогнозира дали новият ПП ще се търси, може ли да бъде конкурентноспособен и какви ще са силните му страни в сравнение с аналогичните ПП). **Резултат -**след евентуални промени се утвърждават формулираните в СИ изисквания.
Проектиране: Начало – след края на фазата за изследване. **Същност** - целта е обхващането и отразяването на потребителския възглед за ПП – външен проект на ПП. Обсъжда се с потребителя и се проектира UI-а, връзки на ПП с ОС и други Софт. средства. **Резултат.** Създаденият външен проект се оформя като документа - Външна спецификация.
Продължителност - 10 седмици.
Програмиране: Начало - започва, когато външната спецификация е представена в някакъв вид, макар да не е преминала окончателно формално утвърждаване. СИ трябва да са били

утвърдени. **Същност -** През тази фаза се създава детайлен (вътрешен) проект. В него е описано как ще се реализира ПП -архитектурата на ПП с пълно описание на всяка програмна част, описание на потока на данните и потока на управлението. Модулно тестване. Създадените програмни части се интегрират в единна система. **Резултат –** работоспособен ПП, който може да бъде предоставен за независимо тестване и приемни изпитания. **Прод. –** зависи от обема и сложността, не повече от 10 месеца. (разделяне на малки части).
Оценка: Начало – когато ПП е сглобен от готовите програмни модули**. Същност** – независимо тестване . Тества се от специалисти, които не са го изработвали, след това приемни изпитания(съответствие м/у ПП и СИ. Да се документира експлоатационната годност на ПП в реални потребителски условия; да се проверят качествата на документацията. **Резултат -** документ, удостоверяващ експлоатационната годност на ПП – позволява предаване за разпространение и експлоатация. **Продължителност** – оптимистично 1/3 от фазата на програмирането, може да достигне нея по продължителност.
Използване: Начало - издаването на документа за годност. **Същност -** инсталиране и последващата експлоатация на ПП. Извършва се обучение на потребителите с различна продължителност. Всички видове съпровождане (усъвършенстване, отстраняване на грешки, добавяне на нови функции). **Резултат -** Експлоатацията на ПП. **Продължителност -**Определя се от конкретния ПП. Физичес и логически край.
Функции: функция – съвкупност от сходдни дейности. За всяка от 7-те функции има отделна група хора.
- **Планиране** - обхваща дейностите по съставяне и проследяване на изпълнението на всички видове планове. Според обхвата плановете се делят на: а) такива , които се отнасят до организацията производител - целева програма, стратегически план, тактически план; б) свързани с всеки конкретен ПП: бюджет, план за работата на отделна група, индивидуален план за всеки участник, мрежов график...
Групата по планиране следва да се състои от хора с икономическо образование и с някаква специализация в областта на информатиката и на софтуерните технологии. Най-интензивно групата по планиране работи по време на фазите Изследване и Проектиране.
- **Разработване** - Групата по разработване проектира, съставя външната и вътрешна спецификация, извършва програмирането и тестването (разбира се без независимото), отстранява грешки, сглобява готовите модули, консултира съставящите съпровождащата документация. Тази група включва специалисти информатици и е основната.
- **Обслужване** – не е разумно на високо квалифицирани специалисти да се възлагат дейности от чисто технически характер. Разделя се на групи: а) административно-правни – съставяне на – договори, мрежови графици, финансови отчети, данъчна документация; включва прависти. б) технически – осигуряват: изчислителната техника и поддържането й, заявените инструментални софтуерни средства, стандарти и други нормативни документи. в) изпитания от клас С те са крайни, състоящи се в случаен избор сред готовите за разпращане комплекти от ПП и проверката им за качество и комплектност.
- **Документиране** - създаването и поддържането на потребителската документация, специалисти с филологично образование, чужд език, в големи фирми и софт. технолози(проектират документацията).
- **Изпитания** – установяване на дефекти. 3 вида: клас А - вътрешни, извършвани от самите разработчици; клас В – независими, от групата по изпитанията; клас С – случайни, от групата по обслужването.
- **Поддържане** - проучване на потребителското мнение във фазата анализ на осъществимостта; защищаване интересите на потребителя още при избора на проектантски или програмистки решения;
обучение на потребителя; осъществяване наобратна връзка с потребителя: получаване и систематизиране на съобщенията за грешки и на заявките за изменения от потребителите; връзки с обществеността.

- **Съпровождане** - поправяне на грешки; добавяне на нови възможности; адаптиране към нова ОС или хардуерна среда. Върши се от разработчиците. Изисква най-високата възможна квалификация. Извършва се на различни равнища: гаранционно съпровождане – отстраняване на грешки за около 12 месеца; развитие на ПП и остраняване на грешки; остраняване на грешки; регистриране на съобщения за грешки и на заявките за подобрения, с оглед вземането им под внимание при разработката на сходен ПП. Фиг 3.1

5.Софтуерен проект, инициализация, управление.

Управление на проекти: извършване на съвкупност от разнородни дейности с уникално съдържание за решаване на сложен, нестандартен проблем, при наложени ограничения относно време, разходи, качество и специфични изисквания към организацията на работата.
Ключови идеи: Сложна и неизвършвана до момента работа за постигане на промяна, и то към по-добро;
Предварително определени цели и изисквания за продължителност, разходи и качество; Може да се наложи преструктуриране на организацията–изпълнител, промени в стила на работа и управлението на човешките ресурси.
Разлика между проектите и операциите:
- (**проекти : операции)** – (уникални : повторяеми), (преходни : дълготрайни), (съществена промяна : еволюция), (неустойчивост : устойчивост), (променящи се цели : постоянни цели), (динамични ресурси : постоянни ресурси), (гъвкавост : стабилност), (цели : роли), (неопределеност и риск : опит,рутина).
- основни разлики: средата, която е определена от операциите, е относително стабилна, докато проектната среда е гъвкава, неустойчива и се определя от условията на конкурентно обкръжение. Операциите са насочени към перфектност на изпълнението, докато при проекта насоката е постигане на дългосрочните цели, което предполага свобода на изискванията към изпълнение на дейностите. Проектният екип е мотивиран за достигане на крайния резултат, а не към перфектно изпълнение, синхрон и постигане на някакъв резултат При операциите членовете на екипа имат длъжностни характеристики и изпълняват предварително дефинирани роли. Проектният екип е целево ориентиран, в това число всеки може да изпълнява няколко роли, променящи се с течение на проекта. Проектите са изложени на значителен риск. В началото на проекта не можем да сме сигурни в постигането на крайната цел, тъй като не притежаваме предишен опит за изпълнение на такава уникална съвкупност от дейности. Затова управлението на проекти може да се разгледа като управление на риска, докато управлението на операциите − като управление на текущото състояние.
Деф. Проект: Управленски подход, при който чрез специфична организация на ресурсите – (човешки, финансови и материални) се изпълнява уникална съвкупност от дейности при ограничения за време и разходи и зададени изисквания за качество.
Основни цели на проекта - картинка: предназначение > обхват > организация > време, разходи, качество; време, качество(с права линия не стрелка) > разходи; обхват (с права линия не стрелка) > време, разходи, качество. **Проекти, средства и крайни продукти картинка:** проект(дейности) > средства(цели) > предназначение/полза(продукти/услуги). **Обощение: а)Проектът**: е определен от уникален обхват дейности; изисква нова организация; насочен е към съществяване на промяна (към по-добро). **б)Проектът по необходимост**: има неопределеност и риск; изисква формиране и интегриране по нов начин на организационните ресурси; е ограничен по отношение на време, разходи и качество.
Класификация на софтуерните проекти: по предназначение - **системен, приложен**; по използвана технология на разработване - **Структурна, ОО, покомпонентна**; по големина (размер, обхват, сложност, продължителност) - **големи, средни, малки**; по разработване чрез нова или усвоена вече технология –**иновативен, стандартен**; **възложен/автономен;**

Инициализация на софтуерен проект: А. Възложен - има конкретен Възложител с идея за ПП и финанси; процедура - технико-икономическо задание (от Възложителя); формулиране на изискванията; задание за разработка (ЗР) – три етапа; проучване, разработка, експериментално внедряване; експертна оценка след Р1 и решение за – прекратяване/преработване на ЗР/сключване на договор.
Б. Автономен
Структурен подход за управление на проекти: (Основни цели на проекта – картинка – без правите линии само стрелките): прави линии между - обхват и (структуриране по нива: интергриращо, стратегическо, детайлно); организация и (йерархичко структуриране на организ., схеми за разпределение на отговорностите); време и (мрежови графици, диаграми); качество и (осигуряване и контрол на качеството); разходи и (йерархично структуриране на разходите, контролен куб на разходите).

6.Формулиране и анализ на софтуерните изисквания. Управление на изискванията към софтуера през ЖЦ (Requirements Engineering).

Управление на изискванията:
1. Идентифициране на изискванията
Подходи:Обсъждания, метод на различните гледни точки
Резултат: Пълен списък на изискванията
2. Анализ на изискванията
Резултат: Нареден списък на непротиворечиви и осъществими изисквания
3. Специфициране на изискванията - СИ
4. Валидиране на изискванията
5. Управление на изискванията – проследяване, промяна

7.Технологии за разработване на софтуер. Конвенционални (структурни) технологии.

Най-общите, фундаментални подходи се наричат **парадигми** - традиционен (конвенционален, възможно решение на софтуерната криза – висока цена, незадоволително качество) и обектно-ориентиран.
Диаграма на Kuhn – научно знание > криза > радикална смяна > научно знание. Хронолог. модел на ЖЦ:
Определяне на софтуерната система: - **Определяне на изискванията**: а) **формулиране на изискванията** - всички възможни изисквания въз основа на определените вече предназначение и обхват на предлаганата софтуерна система(FAST - потребителите и разработчиците подготвя предварително множество от изисквания; анализ на различните гледни точки – brainstorming на потребителите, опр. се изискванията евентуално use cases). б) **анализ на изискванията** – класифициране(функционални, не-, задължителни, препоръчителни), формиране на непротиворечива система за всяко изискване, осъществимо?, рискови фактори, ресурси, проследимост, Спецификация на изискванията(СИ). в) **утвърждаване на изискванията** - СИ се проверява и утвърждава от потребители и разработчици и стават основен док. за софтуерната разработка. г) **проследяване на изискванията** - планиране на дейностите по проследяване на изискванията, като в контролните точки се включат и проверки за постигнатата степен на удовлетв. на избраните изисквания. - **Аналитичен Модел** – формален, езултат от проведения структурен анализ и предназначението му е да улесни следващите дейности по проектиране. Състевен е от: а) **модел на данните** представя основните обекти, атрибутите, които ги описват и връзките на обектите помежду им. Обект – всяко нещо, което създава/използва информация в софт. система. Характеристики на отношенията – кардиналност(1:1, 1:N, N:N), модалност(задължително 0, незадължително 1). Диаграма, представяща обектите и отношенията между тях, се нарича диаграма елемент-

връзка. б) **функционален модел** представя основните функции на софтуерната система чрез проследяване на преобразуването на информацията в нея. **Data Flow Diagram** представя трансформациите на данните така че от входните данни в системата да се получат изходните. Правоъгълник за външни обекти, кръг за ф-ция и стрелки за входните и изходните данни. Широко използвано. **Нарича се спецификация на процеса**. в) **поведенчески модел** - системата се описва чрез различимите си състояния и начина на преминаване от едно в друго. Диаграмата на преобразуване на състоянията представя наблюдаваните състояния (правоъгълници) и събитията (стрелки). Основно преимущество на структурния анализ е простотата и нагледност.
Речни на данните - име на обекта, синоними, къде и как се ползва, същност, допълнително инфо.
Проектиране: - **Основни понятия** - обхваща всички дейности за превръщане на утвърдените изисквания в описание, определящо точно съдържанието и функциите на програмите. Основен принцип на проектирането е изследване и разбиране на проблема и последователното му декомпозиране на подпроблеми. Действия: индентифициране на проблема, съставяне на решения, избор на оптимално. Два основни подхода към проектирането: функционален(Състоянието на системата е централизирано и се разделя между функции, опериращи върху това състояние) и обектно-ориентиран(Състоянието на системата е децентрализирано и всеки обект управлява собственото си състояние, комуникация чрез съобщения). - **Методи и средства за проектиране** -два начина: **ad hoc или стандартизиран процес**. Ад хок - въз основа на утвърдените изисквания се изготвя неформализирано описание на естествен език , служи за ръководство при съставяне на програмите. По-систематичен е подходът с прилагане на структурни методи, които предлагат начини за описване и процедури за създаване на софтуерни проекти.
- **Етапи на проектиране** - два основни етапа на функционалното проектиране:предварително (външно) проектиране и детайлно (вътрешно) проектиране.Външното създава архитектурен проект. Обхваща:- логическата организация на данните; - структурата на системата; - интерфейс на системата; Вътрешното (component-level) софтуерната система се представя като йерархия от програмни части(модули – основна,
самостоятелно разработвана, тествана и док. единица; 4 атрибута – вход/ изход, функция, механизъм за реализация, вътрешни данни;). - **Методи за описване на проекти** - предпочитат се неформалните методи поради съпротивата и на потребителите, и на проектантите(нямат теоритична подготовка, твърде сложни класове – паралелно изпълнение и тн.). Използваните техники за описания са:а) графично описание; б) таблично представяне; в) текстови описания: този вид описания са с различна степен на формализираност и най-общо се делят на две: псевдокодове и езици за проектиране на програми (PDL - Program Design Language). Псевдокодът е неформален език - речникът на езика включва глаголи, имена от речника на данните и запазени думи за описване на логиката(цикъл, условие). **Критерии за сравняване** на методите за **описване на проекти** - модулност; простота; леснота на редактиране; възможност за автоматично създаване и обработка; съпровождаемост; удобство на представяне на данните; възможност за верификация; сложност на прехода „проект-програма".
Организационни аспекти на проектирането: проектирането трябва да се осъществява целенасочено и систематично,с регламентиране на основните процедури. Най-сложният проблем е определянето на подходящо ниво на декомпозиране и на подробност на проекта.
Оценяване на качеството на проекти, характеристики: **Пълнота** - утвърдените изисквания да са отразени в проекта на софтуерната система. Проектът да бъде разбираем, ясен, точно и недвусмислено описан, и да не зависи от платформата, на която ще се реализира. **Лесно проверяем** и **документиран**. **Спецификация на проекта:** I. Цел и обхват на дейностите по проектиране; II. Описание на проектите а) външен проект;б) детайлен проект; III.Съответствие на проектите с утвърдените изисквания; IV. План за модулно и системно тестване; V.Допълнителни условия и ограничения за проектирането; VI. Приложения (описание на алгоритми, алтернативни процедури,таблични данни, извлечения от други документи, и т.н.).

Програмиране: Основната идея на структурното програмиране (без GOTO") е използването на три класически конструкции: линейна последователност,конструкция за разклоняване IF-THEN-ELSE и за цикъл. Техниката за написване на структурни програми съчетава низходящото проектиране и постъпковото уточняване. Започва се с най-обща схема на програмата, като всеки цикъл или проверка на условие се представят чрез съответните оператори, а всяка вложена част постепенно се разширява, докато се получи окончателната програма. Основно преимущестшо на структ. проектиране е повишаване на разбираемостта и тестируемостта на програмите и намаляване на усилията за тяхното разучавано и съпровождане. Недостатък: необходимостта от усвояване на тази специална програмистка техника и прилагането й при създаване на всички програми; изискват по-големи изчислителни ресурси. **Интергриране и Тестване:** стратегиите за тестване определят кои части на с-мата да се тестват, в какъв ред, с какви методи и средства, в каква среда и от кого. Обикновено се започва с модулно тестване, след което се преминава към тестване на компонентите от по-високо ниво.
- Модулно (поелементно) тестване: проверява коректността на най-малките програмни компоненти - модулите. Препоръчва се тестването да започне с проверка на интерфейса (на входните и изходните данни) на модула и да продължи с проверка на логиката,обработката на данните и реализираните функции. Модулното тестване изисква специална среда. Необходимо е да се създадат допълнителни програми драйвери (drivers) за извикване на тествания модул и опори (stubs) за представяне на модулите, извиквани от тествания модул. **- Интеграционно тестване:** техника за конструиране на програмна структура от тествани вече елементи и организиране на тестване за откриване на интерфейсни грешки. Съставянето на програмната структура за тестване става по два начина: интегрално или инкрементално. **Интегралното** - след тестването на всички модули се сглобява цялата програмна с-ма и започва системно тестване. Предимство: не се разработват допълнително драйвери и опори, недостатък: откриването и локализирането на грешки е много по-трудно поради възможността за наслагване на последиците от няколко интерфейсни грешки. **Инкрементално тестване** започва с два тествани модула и на всяка стъпка се добавя един допълнителен модул.
Реализират се възходящо или низходящо. **Възходящ подход** се започва от най-долното ниво на йерархията (модула), образуват се подсистеми и накрая се достига до най-високото ниво - системата. **Низходящ подход(драйвери)** започва се с тестване на системата като цяло и се продължава с тестване на елементите от по-ниските нива. **„Сандвичово тестване"(стъбове) -** до определено ниво в йерархията се прилага единият, а оттам нататък - другият подход.
Особено важни са т.н. **приемни тестове**. Те са последна проверка на завършената софтуерна система преди разпространяването й. Алфа-тестването се провежда с участието и на потребители, но в средата
на разработване. Бета-тестването е изцяло в реална потребителска среда.

8.Обектно-ориентиран подход за разработване на софтуер.

Осн понятия: обект -познаваем предмет, елемент или същност, имащ важно функ. предназначение в разгл приложна област. Вс обект има състояние, поведение и индивидуалност. Сходни обекти определят общ за тях клас. Състоянието на обекта се характеризира с изброяване на всичките му възм свойства(атрибути). Вс атрибут има област на ДС. С всеки обект могат да се свържат операции. Понятието клас е осн за ОО-подхода. Той обхваща данните и алгоритмите ,описващи съдърж и поведението на „нещо" от реал свят. Суперклас е съвкуп от кл-ве, а подклас - екземпляр на клас. Същ йерархия на класовете, като подкл насл атр. и оп-те на суперкласа, но могат да имат и специф за тях. Взаимодействието чрез съобщения, изпълнява се посочената в съобщ операция и връща управл на обекта подател. Така се описва поведението на обектите и на ОО-система като цяло.**Три осн характеристики** на ОО-подхода: **Капсулиране** => преимущества: вътр реализация остава скрита; данните и операциите са обединени в едно цяло - класа, (повторното му използване); връзките са опростени(не зависи от вътр структ от данни). **Наследяване** => Вс подкл

придобива автоматично всички атр и операции на съотв суперкл(повторно използване,лесно внасяне на изменения,нов клас чрез дописване на специф елементи, множествено наследяване). **Полиморфизъм** - дава възм разл операции да имат едно и също име=> настроеваемост и гъвкавост, защото с унифицирано извикване могат да се реализират специф обработки.

ОО анализ и проектиране: Целта на ООА е създаването на модел, представящ статични и динамични характеристики на класовете и взаимоотн м/у тях. Различни методи за ООА преминават през едни и същи стъпки: **1)**Изясняване на потр изисквания за системата **2)**Идентиф на потр сценарии (use-cases) **3)**Избор на кл-те и обектите въз осн на изискв **4)**Идентиф на атр и операциите за вс обект **5)**Деф на структурите и йерархиите на класовете **6)**Постр на модел, описващ обектите и връзките м/у тях (object-relationship model) **7)**Постр на поведенчески модел **8)**Проверка на ОО аналитични модели за съответствие с потребителските сценарии. Проектирането на 2 нива, съответстващи на външното и детайлното проектиране при конвенц подход - проектиране на системата(1) и проектиране на обектите(2). (1) - опр се архитектурата на ОО-приложение. Аналитичения модел се декомпозира на подсистеми, като се описва предназн на всяка подс-ма и връзките м/у тях. Проектират се UI и лог организация на данните. (2) - за вс обект се съставя интерфейсно описание и описание на реализацията. Интерфейсното опр вс съобщения, получавани от обекта, и операциите. О на Реализацията - свързаните с обекта структ данни и алг за вс операция. За стандартизирано описание на моделите в ОО - разработване на софтуер е създаден Unified Modeling Language. Осн части на **UML** са: а) **Представяния** (аспекти — функционални, нефункц, организационни и др.) - потребителско (use-case view); логическо; компонентно; конкурентно; обвързващо (deployment view) б) **Диаграми** - (use-case |class|object | state| sequence| collaboration| activity| component| deployment diagram) в)**Елементи на модела** -използваните в диагр концептуални обекти - клас, обект, съобщение и връзките м/у тях. г) **Общи механизми** - осиг допълн инф-я за семантиката на елементите на модела или как да се разшири моделът за специф процеси, системи или организации. UML се използва за моделиране във вс фази на разработването на софт с-ми. **ОО програмиране и тестване**: Прогр представлява превръщане на ОО-проект в програмен код. Класовете, дефинирани при проектирането, трябва да се опишат като класове в съотв ОО-език за програмиране. Резултатът е програма, която може да се изпълнява. Целта на ОО-тестване - да се открият колкото се може повече грешки в рамките на определените за тестването бюджет и време. Тестването трябва да започва от най-ранните фази и първите обекти за тестване да бъдат моделите на системата, създадени след ОО-анализ и ОО-проектиране. ОО-тестване включва формалните проверки за правилност,пълнота и непротиворечивост в синтаксиса, семантиката и използването на моделите, създадени след ОО-анализ и проектиране. модулно тестване тук е тестване на клас.Проверка на алгоритъма тук е проверка на операциите и промените в състоянията на класа. интеграционно тестване тук е тестване на съвкупност от класове. **Интеграционното** ОО-тестване може да бъде: проследяващо (thread-based) -тестват се вс класове, свързани с едни и същи събития в системата; пластово - разгл на йерархията от кл и разделяне на кл на независими и последователни слоеве от зависими класове; клъстерно - сътавяне на групи от взаимодействащи класове и тестването им. **Системното** ОО-тестване съвпада с традиционното. Генерирането на тестови данни зависи от метода на тестване (стресово, случайно, сценарийно) и от обекта на тестване — отделен клас или група от класове.

Управл на ОО разработване: Прилагане на осн управленски техники за разработването на ПП, за управление на: процеса на създаване на софтуер, създавания продукт, софт проект и човешкия фактор. - - - **Управл на процеса на разработване** - рекурсивно - паралелен модел, близък до спираловидния и до еволюционния модел, същността му е в разработването на последователно разрастващи и усложняващи се прототипи до окончателното изграждане на целевата с-ма. За вс прототип се извършват едни и същи дейности: планиране, анализ, проектиране, извличане на компоненти от библиотеката за повторно използване, конструиране на прототипа с налични и новоразработени компоненти, тестване и оценка от

потр, определящ изискв към следващия прототип. Ефективно е планирането и измерването на развитието на проекта да се извършва за всяка независима компонента.
- **Управл на проекта и продукта** - Станд техники за управл на софт проекти са приложими и при ОО-разработване - разлики: а) допълване на стандартните методи за оценяване на цената на разработване, трудоемкостта и продълж на софт проект с разработените ОО-метрики, измерващи специфични за подхода елементи — сценарии на използване, осн и поддържащи класове и връзките м/у тях б) за проследяване на развитието на - станд техника с контролни точки, но достигането им се определя по специф начин в) управл на проекта и на продукта се преплитат поради специф покомпонентно разработване при ОО-подход. Всяко приложение може да се изгражда чрез компоненти - клас, който е създаден и съхранен, с опр ниво на качество и степен на общност, която позволява да се преизползва.
- **Управление на персонала** - Разграничени са 3 групи изпълнители на ОО софт проект: програмисти на приложения1), програмисти на компоненти2) и стратези3). 1) определят кои обекти да се групират, за да се изгради приложението. 2) участват активно в началото на всеки проект, за да дадат инф-я, какви налични комп има и как могат да се използват. Осн функции на 3) са координиране и съветване - опит в повторното използване като осн подход в софт орг-я, решават и какво да бъде съдърж на библ от комп. **Въвеждане на ОО подход**: Препоръки, следването на които би улеснило процеса на преход:1) Обучение на персонала на софт фирма на вс нива.!ясни цели, подкрепа и финансиране за периода от време, необх за постигане на технологично ниво. Ръководителите на проекти тр да бъдат спец обучени. 2)Стартиране на пилотен проект за проверка. < петима разработчици (3- 5 месеца). За планиране и оперативно управл - - опитен специалист, който да ръководи проекта и да осигури документирането на възникващите проблеми и успешни решения, така че да се натрупва know-how инф. За следв проекти прилагането на метрики за измерване на подобренията в процеса на разработване и производителността. 3) Вс осн дейности трябва да се преразгл и ако е необх, да се преформулират съдърж и начинът на извършване на някои от тях 4) Избиране на подх методи за разработване на софтуер и адаптирането им към нуждите на орг-ята. Критерии за избора: история на създаване и използване на метода, ниво на зрелост и стабилност,степен на унив-ст, адекватност с ОО-подход, сложност на усвояване и прилагане, гъвкавост, степен на автоматизираност, ползваемост и др.

9.Гъвкави (agile) софтуерни технологии. Extreme Programming (XP).

AGILE: Причини: Бързо променящи се пазарни условия; Променено отношение на разработчиците към стила на работа. **Принципи:** Активно включване на потребителите в процеса на разработване на Софтуера; Реализация на инкрементално разработване, реализация на „разрастващо се" (инкрементално) разработване, като изискванията за всяка следваща итерация се преформулират въз основа на оценяване на текущата; Регламентиране на стил на работа, основан на компетентност и мотивираност, екипност и стимулиращ творчеството психологически климат; Приемане на неизбежността на промените, съпътстващи софтуерните проекти и адаптиране на техниките на разработване към тях; Придържане към опростени конструкции и схеми на работа с цел минимизиране на ресурсите за осъществяването им.
Извод: Гъвкавите методи не предлагат универсашо решение на проблемите в софтуерното производство.
Прототипиране: - Дефиниция - съвкупност от дейности, осигуряващи ранно демонистриране на някои свойства на софтуера, който трябва да бъде създаден.
Цел: Да се определят изискванията към разработваната софтуерна система.
Стъпки: Проектиране на прототипа - избор на функциите и свойствата, които прототипът ще изяви (вертикално - реализират се само някои ф-ии, но в техния предполагаем окончателен вид / хоризонтално - всички функции присъстват, но не са са реализирани напълно); Създаване на

прототипа - в прототипа се изберат само с-ва, важни за конкретното приложение; Оценяване на прототипа - важна стъпка, защото осигурява обратна връзка; Използване - Прототипът или се отхвърля, или се използва като база за разработването на целевата система. **Недостатъци**: За да се разработи бързо, може да е избрана неподходяща за целевата система операционна или хардуерна среда; Някои съществени за системата х-тики могат да бъдат игнорирани в прототипа, но да са задължителни за крайния продукт; Поради многократното модифициране на прототипа, той може да е с ниска съпровождаемост; Изисква допълнително време и средства. **Видове**: **Изследователско** - изяснява изискванията към целевата система чрез представяне на няколко алтернативни решения; **Експериментално** - провери дали избрано решение на даден проблем е подходящо; **Еволюционно** - разработването се осъществява в динамична среда и непрекъснато променящи се изисквания (най-отдалечено от оригиналното разбиране). Крайният продукт се създава като последователност от инкременти, всеки от които е прототип за следващия.
Методи за създаване: Езици от четвърто поколение - дават възможност на разработчика бързо да генерира изпълним код и затова са подходящи за създаване на прототипи; Използване на готови софтуерни компоненти; Среди за формално специфициране и прототипиране.
Разработване с участието на потребителя: - Дефиниция - Съвкупност от методи, техники и целенасочени действия, осигуряващи активното участие на потребителя в проектирането и създаването на софтуерната система, която той ще използва. **Цел:** Да се повиши активността и съответно отговорността на потребителя за качеството на програмния продукт, като той се включи активно и в някои междинни фази. **Автономни СП:** Проекти без конкретен възложител. Обикновено до идеята за създаването им се достига след маркетингово проучване. **Възложени СП:** Конкретен възложител, който финансира разработката след завършването й става неин собственик.
Ex3m Programming: - Същност - Лека методология за малки и средни колективи, разработващи софтуер при неясни или бързо променящи се изисквания. Определени принципи и практики се довеждат до възможния си предел. **Ангажименти:** Към програмистите - ще работят непрекъснато съществени неща, няма да им се налага да се справят с тежките ситуации сами, ще имат възможност да допринасят максимално за успеха на проекта, ще им се налага да вземат решения от своята компетентност и никога такива, за които нямат квалификация; Към клиентите - ще получават максималната възможна стойност от всяка изработена седмица, на всеки няколко седмици ще получават конкретни използваеми резултати, каквито предварително са поискали (такава завършена подсъвкупност на крайния програмен продукт се нарича рилийз), ще имат възможност при необходимост да променят насоката на проекта по време на реализацията му, без това да бъде свързано с прекалено големи разходи.

10.Съвременни софтуерни технологии – мултиплициране, разработване на надежден софтуер

Мултиплициране: Същност - Съвкупност от планирани и систематични дейности, насочени към максимално използване на съществуващи софтуерни елементи в процеса на създаване на нов софтуер.
Видове преизп. компоненти: (по същността им) **Идеи или концепции** - В тази група са алгоритми, методи, техники или формални модели; **Софтуерни компоненти** - Това са най-често използваният тип елементи и могат да бъдат: приложни системи, подсистеми, програмни модули функции или други обособени програмни части, процедури или умения. // (по обхват) **Вертикално повторно използване** - създават се основни модели, които се прилагат във всички разработвани за тази област софтуерни системи; **Хоризонталното ПИ** - създават се универсални компоненти, които могат да се вграждат в програмни системи за различни приложни области. // (по осъществяване) **Планирано и систематично ПИ**; **Инцидентно** - търсят се съществуващи софт. елементи. // (по използвана техника) **Сглобяващо**; **Генериращо** - чрез генератори на програми автоматично се създават съответните програмни части; (по начин на използване) **Без**

промяна - Основава на проверено качество и минимизира усилията за съпровождане; **С модифициране**. // (по вида на СП) Могат да се използват спецификации, проекти, фрагменти от програмен текст, документация, тестове...
Стъпки за ползване: идентифициране на компонентите за ПИ, документиране на всяка компонента и включване в общодостъпна библиотека, обучение на всички участници в разработката как да използват библиотеката.
Компонент: Софтуерна единица с функционалност и зависимости изцяло определени от интерфейсите му. Може да е: стандартизиран, независим, вградим (връзките с него се осъществяват чрез известни протоколи)
Разработване на надежден софтуер: характеристика, която е свързана с честотата на сривове в системата (следствие от дефекти).
Избягване на дефекти: Основава се на следните принципи: създаване на прецизни спецификации; Озползване на проектиране, което позволява капсулация; Използване на механизмите за осигуряване на качеството със систематична верификация и валидация на системата, планиране и осъществяване на подробно тестване; Скриване на информацията, Избягване на рекурсии, указатели, паралелизъм, където може.
Софтуер с приемливо ниво на дефекти: проектиран и програмиран така, че продължава работата си и при наличие на дефекти. **Трябва да се:** Прогнозират или откриват дефекти по възможност преди да са довели до срив на системата; Оценят на пораженията след срив на системата; Възстановяване след срив; Локализиране и отстраняване на отговорните дефекти

Тема 11. Качество на софтуера. Модели на качеството. Софтуерни метрики.

Софтуерни метрики

Класификация в зависимост от **предназначението**

а) оценяване на разходите и усилията за разработване на софтуера;

б) измерване на производителността на разработчиците;

в) моделиране на качеството на софтуера и оценяването му;

г) измерване и прогнозиране на надеждността на създаваните софтуерни системи

Класификация в зависимост от **целта** на прилагане на метриката

А)за характеризиране - да разберем изследвания обект и да установим база за сравнение при бъдещи оценки;

Б)за оценяване- регистриране на текущо състояние на изследвания обект

в) за прогнозиране- предварителна оценка за бъдещо състояние на изследвания обект.

г) за подобряване - натрупване на количествена информация за определяне на факторите, от които зависи качеството на продукта или ефективността на процеса.

Класификация в зависимост от **типа на изследвания обект**

Процеси - дейности, свързани със създаването и използването на софтуера;

Продукти - резултати от процесите;

Ресурси - елементи, входни данни за процесите.

Класификация в зависимост от **начина на получаване на информацията**.,

а) регистрационен метод - получаване на информацията по време на разработване или функциониране на ПП чрез регистриране на отделни събития

б) измерителен мегод - информацията се получава чрез прилагане На Специално разработени инструментални средства;

в) възприятиен метод - информация, получена след анализ на зриТелни или слухови възприятия

г) изчислителен метод - използване на теоретични и/или емпирични зависимости изведени въз основа на статистическите данни, натрупани през различни фази на жизнения цикъл на ПП.

Класификация за **идентифицирането на програмни свойства, които да бъдат излседвани с определена цел:**

а) в зависимост от начина на изследване статични и динамични. При статичните метрики се анализира само текстта на програмата, докато за динамичните метрики е необходимо изпълнението й;

б) в зависимост от изследваната програмна характеристика - сложност (структурна, текстуална или алгоритмична), модулност, тестируемост, независимост от средата, модифицируемост и др.;

в) в зависимост от нивото на декомпозиране - изследване на отделна програмна част или на програмна система като цяло. Метриките от втората група са много по-сложни, защото се изследва потокът на данните и потокът на управление в многокомпонентни системи.

Една метрика се нарича **адитивна**, ако резултатите от прилагането й за програмна система могат да се получат чрез сумиране на резултатите от прилагането й върху съставящите я програмни части.

Примери за софтуерни метрики

Ml. Метрика за размера на програма.

Метриката е статична и може да се прилага както за определен програмен модул, така и за програмна система. За мярка на размера на програма се избира броят програмни редове в изходния текст на програмата: *L = L*i + *L2,* където *L* е общият брой редове, L_1 - броят коментарни и празни редове, а L_2 – броят на същинските програмни редове. Метриката е адитивна.

Приложимост: 1. Броят на грешките, усилията за разбиране, поддържане и съпровождане са право пропорционални на размера на размера на програмите. 2. В зависимост от размера и на размера В зависимост от размера (на ниво програмна единица) и на размера и броя модули в програмна система, програмите могат да се класифицират като прости, средно сложни, сложни и свръх сложни и за всяка категория да се формулират мерките за осигуряване на качеството,

да се оценят обхвата и сложността на тестването, съпровождането и др. 3. L е най-проста мярка за извършената от програмиста работа.

М2. Метрика на Мак Кейб за структурна (цикломатична) сложност

Тя е статична и измерва сложността на потока на управление в програмата. Тази метрика е една от най-често използваните и цитирани в литературата. За прилагане на метриката трябва да се построи управляващият граф на програмата, в който върховете са отделните оператори и два върха са свързани с ребро, ако има предаване на управление между съответните оператори. Тогава *V(G) = e — n + 2*p,* където *V(G)* е цикломатичната сложност, представляваща максималния брой линейно-независими пътища в програмата, е - броят на ребрата в управляващия граф, n - броят на върховете в управляващия граф, а *p* броят на силно свързаните компоненти в графа. Тъй като обикновено *p* = 1, получаваме *V(G)* = e — *n +* 2. Друга интерпретация и начин на пресмятане на цикломатичната сложност може да се даде чрез теоремата на Ойлер за планарен граф. По дефиниция *планарен* е този граф, който може да се изобрази без пресичане на ребра. Ойлер доказва, че n — e + r = 2, където n и e са съответно брой върхове и ребра, а r е броят на областите, определени от ребрата на планарния граф. Преобразувайки горното равенство, получаваме r=e-n+2. Цикломатичната сложност съвпада по стойност и може да бъде пресмятана чрез броя на областите в планарен граф. Мак Кейб доказва, че броят на областите е с 1 повече от броя на разклоненията в програмата. Т.е. ако намерим общия брой *d* на операторите IF, на операторите за цикъл и на операторите CASE, можем да изчислим цикломатичната сложност чрез формулата *V = d +1.* Недостатък на метриката на Мак Кейб е, че не отчита дължината на линейните участъци и нивото на влагане на управляващите структури. Не се отразяват и междумодулните връзки, които влияят на структурната сложност на програмната система като цяло. Разширение на метриката на Мак Кейб е метриката на Гонг-Шмид, която предлага цикломатичната сложност да се изчислява по формулата C(G) = V(G) -f E, където E представя обобщената степен на вграждане на управляващите структури една в друга.

Приложимост: Метриката на Мак Кейб предлага мярка за вътрешно-модулната сложност, която определя леснотата на разбиране и усвояване на програмите и точността на внасяне на изменения в тях. Въз основа на емпирични данни се препоръчва да се проектират модули с цикломатична сложност *V* < 10 като се счита, че модулите със сложност до 10 са прости, до 20 - средно сложни, от 30 до 50 - сложни, а при сложност над 50 трябва да се обмисли преструктуриране на програмата

М3. Метрика на Холстед за текстуална сложност

Тази метрика е статична метрика за програма, написана на произволен език за програмиране. Програмата се разглежда като състояща се от елементи (лексеми), които се класифицират в две групи: пасивни елементи (операнди) и активни елементи (оператори). Onepandume имат стойност и могат да бъдаг константи или променливи, използвани при реализацията на алгоритъма в конкретна програма. Операторите са всички останали елементи влияещи върху стойностите или наредбата на операндите.За пролагане на метриката се определят следните оценъчни елементи: n1- брой различни оператори; n2-брой различни операнди; N1-общ брой оператори; N2-общ брой операнди; **Речник**: n =n2+n2 :броя на „градивните" елементи на изследваната програма. **Дължина на програмата** (в брой лексеми): N = N1 + N2. **Изчислена**

дължина на програмата:Низч=n1*log2n1 + n2*log2n2. Тази мярка позволява да се оцени максималният размер на програма, използваща този речник, т.е. точно тези оператори и операнди**. Обем на програмата:** V = (N1 + N2) * log2 (n1 + n2). Тази мярка се свързва с броя на битовете, необходими за двоичното представяне на изследваната програма. Доколкото за представянето на всеки от n-те различни елемента са необходими поне $\log_2$ n бита, то за представяне на последователност от *N* такива елемента ще са необходи- ми *N* * $\log_2$ n бита. **Потенциален обем**: V* = (2 + n2*) log2(2 + n2*).Потенциалният обем е обемът на минималната програма за решаване на разглеждания проблем. В такава програма има само два операторa- името на функцията и оператор за групиране (т.е. n1* = 2), а с n2* е означен минималният брой операнди. **Ниво на реализация**: L = V*/V Имайки реалния и потенциалния (минимален) обем, смислено е да разгледаме отношението им, защото то дава представа колко сме „услож-

нили" програмната реализация, отдалечавайки се от тривиалната. **Усилия за създаване на прогртмат***а: = V/L.* Тази мярка е за интелектуалните усилия, вложени за написването на програмата, като мерната единица е ***ernd*** - елементарни умствени импулси.

Приложимост: 1- Могат да се оценяват усилията за създаване на програмата. 2. Може да се оцени времето за програмиране чрез използване на индекса на Страуд за интензивност на интелектуалните занимания, който за програмирането има стойност *S* = 18. Така *T = E/S = E/18.* 3. Гордън доказва, че усилията за създаване на нова програма са сравними с усилията за разбиране на съществуваща. 4. формули за предварително оценяване **на броя** на откриваните грешки: **B_1 — V/3000** - брой грешки, откривани при изпитанията; B_2 = 4.B_1 = V/750 - общ брой грешки, открити през жизнения цикъл на програмата.5. възможността усилията за тестване да се оценяват чрез произведението *NX * E,* където *NX* е индекс за ниво на влагане , а Е е мярката на Холстед за усилията на разработване.

М4. Метрика на Рехенберг за технологична сложност

Метриката на Рехенберг е опит за преодоляване на едностранчивостта на съществуващите метрики за сложност. Тя е статична, комплексна метрика, която може да се прилага за програми или програмни системи, написани на произволен език за програмиране от високо ниво. Рехенберг предлага следната формула: *CC = SC + EC + DC,* където CC е комплексната сложност; *SC* - операторна сложност; EC - сложност на използваните изрази; *DC* - сложност на данните. Ако NS е броят на операторите в програмата, то съответните относителни мерки са: *RCC = CC/NS, RSC = SC/NS, R.EC = EC/NS, RDC =• DC/NS.* Операторната сложност *SC* на програмата е сума от операторните сложности на съставящите я оператори. За всеки тип оператор е въведе-

на мярка за сложността му. Сложността на изразите EC се пресмята чрез мерки за сложността на използваните в израза операции. Сложността на данните *DC* измерва разстоянието между деклариране и използването на всяка от данните. Мярката за сложност DC за програмата е сума от мерките за сложност на използваните променливи.

Приложимост: таблици за операторната сложност, сложността на изразите и сложността на данните лесно може да се настрои метриката така, че да може да се използва за програми, написани на кой да е език от високо ниво, като се отразят и конкретните потребителски виждания за всяка от разглежданите мерки.

M5. Метрика за информационния поток

Статична метрика, предложена от Henry и Kafura, която дава мярка за междумодулна сложност на първичния код. Този вид сложност се определя от информационния поток от и към даден модул, като се отчита броят на предаваните параметри, глобалните променливи и броят на входно/изходните данни. Оценъчните елементи са: ini- мярка за входящия информационен поток;; outi-мярка за изходящия информационен поток;weighti - теглови коефициент, който може да бъде 1 или в по-сложния вариант - мярка за размера на модула или друга мярка за сложност. Ізчисляваната мярка за сложност на информационния поток е: HKi — weighti * (ini * outi). Мярката е адитивна и сложността на програмната система е сума от изчислените сложности на всеки от модулите.

Приложимост: Мярката може да се използва за оценяване на междумодулната информационна сложност и оттам косвено да се оценят усилията за съпровождане и за повторното използване на отделните модули и на изследваната програмна система като цяло.

M6. Метрика за четимост

Обект на изследване са различни видове документи – ръководства за потребителя, материали за обучение, спецификации и др. Предлаганата мярка е $F = 0.4(lm + plw)$, където: lm е средна дължина на изречение, *plw* - относителен дял на дългите думи.

Приложимост Мярката на четимост *F* отразява трудността на текста. Боем предлага мярката *F* да изпълнява условието 0<F<12 за обикновените делови документи и 12 <=F<16 за спецификации документация и научни отчети.

Обектно-ориентирани метрики

A. Метрики за ОО програмиране

01. Среден брой методи за клас = Общ брой методи/общ брой класове за обект. По-големият брой методи за клас усложнява тестването поради увеличения размер и сложност на обекта. От друга страна, ако тази мярка е по-голяма, тъй като подкласовете наследяват повече методи от суперкласовете, се увеличава възможността за мултиплициране.

02. Средна сложност на метод—Сума от цикломатичната сложност На всички методи/Общ брой на методите в приложението. Тази мярка може да се използва за оценяване на усилията за тестване и съпровождане на 00 приложение.

B. Метрики за ОО проектиране

Съвкупността MOOD включва 6 метрики, които са свързани с основните характеристики на 00 подход като капсолиране (метрики MHF и AHF), наследяване (метрики MIF и AIF), полиморфизъм (метрика PF) и предаване на съобщения (метрика CF). MHF - частно на сумата от скритостите на всички методи, дефинирани във всички класове и общия брой методи, дефинирани в изследваната програмна система. Скритостта на метод е мярка, която се определя от частта на класовете, от които този метод не е видим, изразена в проценти. AHF - частно на сумата на скритостите на всички атрибути, дефинирани във всички класове и общия брой методи, дефинирани в изследваната програмна система. MIF- частно на сумата на

наследените методи във всички класове на изследваната система и общия брой налични методи за всички класове. AIF - частно на сумата на наследените атрибути във всички класове на изследваната система и общия брой налични атрибути за всички класове. PF - частно на реалния брой възможни различни полиморфни ситуации за клас Ci и максималния брой на възможните различни полиморфни ситуации за клас Ci;. CF - частно на максималния възможен брой на свързвания в системата и реалния брой на свързвания, които не са

резултат от наследяване.

B. Класически метрики, адаптирани към ОО разработване

По-дългият период на съществуване и експериментално доказване на полезността на класическите метрики провокира желанието да се приспособят някои от тези метрики към спецификата на ОО технология. Метриките за цикломатична сложност, брой на програмни редове и ниво на коментируемост могат да се прилагат и за програмните фрагменти, реализиращи методите. Нови метрики за ниво на свързаност между обекти и класове могат да се създадат по аналогия с метрики за междумодулните връзки. Чрез метрики за дълбочина на дървото на наследяване и брой на наследниците в това дърво могат да се изследва характеристиката „наследяване" в конкретна програмна реализация на ОО приложение.

КАЧЕСТВО НА СОФТУЕРА

Качеството е съвкупността от средства и характеристики на даден продукт или услуга, носители на способността му да отговори на явно или неявно указани нужди.

МОДЕЛИ НА КАЧЕСТВОТО

Модел на Боем

Качеството на софтуера Боем свързва преди всичко с неговата „***полезност***" и възможност за лесно ***съпровождане.*** Първият аспект се определя от няколко характеристики ***надеждност, ефективност*** и ***използваемост*** ог гледна точка на потребителя- човек. Вторият аспект зависи от други характеристики - ***тестируемост, разбираемост*** и ***модифицируемост.*** Освен това има още една, несвързана с двата аспекта характеристика, наречена ***портабелност (мобилност).*** Характеристиките от своя страна зависят от свойства на по-долно ниво. Тази зависимост вече не е чиста йерархия, защото не само че дадена характеристика се определя от две или повече свойства от по-долното ниво, но има свойства, които определят повече от една характеристика. Още тогава Боем е осъзнал, че по някакъв начин трябва да се отразява важността на метриката. Защото е ясно, че дадено свойство или характеристика е много важно за даден тип програмни продукти и не толкова - за други. Още един жалон в работата на Боем е разбирането за необходимостта от автоматизирано оценяване на свойствата и характеристиките. **Недостатъци -** не съвсем ясната структурираност, недостатъчната пълнота на множеството от характеристики, съсредоточаване почти изключително върху качеството на

програмния код, а не върху цялостния програмен продукт, сравнително тясната експериментална база.

Типичен йерархичен модел:

Структура

Качеството се разглежда като йерархична структура. То се нами-

ра на най-високото ниво - 0.

На следващото ниво - 1 - се намират факторите. Факторът се определя като потребителски ориентирано свойство, представящо даден аспект на качеството на софтуера от гледище на потребителя. В зависимост от конкретния модел факторите могат да бъдат от 6 до 16. В разглеждания от нас те са 6 и са следните: гъвкавост, коректност, надеждност, съпровождаемост, удобство на използване, ефективност.

Гъвкавост*:* лекота на адаптиране към нови функционални условия, включително при изменение на областта на приложение или други условия на функциониране.

Коректност*:* степен на съответствие на специфицираните алгоритми и други изисквания спрямо обработката на данни, както и спрямо потребителската документация.

Надеждност*:* способност на програмния продукт да изпълнява зададените функции, предвидени в програмната документация, при отклонения, възникващи в средата на функциониране.

Съпровождаемост*:* възможност за отстраняване на отклонения и грешки в процеса на експлоатация на програмния продукт и за поддържането му в актуално състояние.

Удобство на използване*:* възможност за усвояване и експоатация на програмния продукт с минимални усилия.

Ефективност*:* Пълнота и скорост на изпълнение на специфицираните функции в зададената изчислителна среда.

На ниво 2 са **критериите*.*** Критериите са софтуерно ориентирани свойства, представящи характеристики на програмния продукт. Всеки фактор се определя от няколко критерия.

На ниво 3 се позиционират **метриките**. В йерархичните модели **метриките са софтуерно ориентирани детайли на даден критерий**.

Последното ниво - 4 - е това на **оценъчните елементи*.*** Оценъчният елемент е елементарна характеристика на най-ниско ниво, която подлежи на количествено оценяване.

Определяне стойностите на оценъчните елементи

В рамките на йерархичните модели в процедурата за намиране на количествена оценка на качеството директно се оценяват само оценъчните елементи. Методите, по които става това, могат да се класифицират по два признака:

А. По **начина на получаване** на информацията за програмния продукт: измерителен, регистрационен, органолептичен, изчислителен.

Б. По **източника на получаване** на информацията: традиционен, експертен, социологически.

Измерителният метод се състои в използването на програмни инструментални средства за определяне обема на програмата на времето на изпълнението на цялата програма или определени нейни клонове, на времето на реакция на програмата на определени входове, както и на други показатели.

Регистрационният метод се основава на информация, получена по време на изпитания или функциониране на програмния продукт, когато се регистрират определени събития, например време и брой на грешки, време на начало и край на работата.

Органолептпичният метод е основан на използването на информация, получавана от човека в резултат на анализ на възприятията му чрез сетивните органи (зрение, слух) и се. прилага при определяне на такива показатели като удобство на използване, коректност и други подобни.

Изчислителният метод използва теоретични и емпирични зависимости, статистически данни, натрупвани при изпитанията, експлоатацията и съпровождането на програмния продукт.

Стойностите на оценъчните елементи по **традиционния** метод се определят в специализирани организации от звена по изпитания и изчисления.

Определянето на стойностите на оценъчните елементи чрез **експертния** метод се осъществява от група експерти, компетентни в решаването нададения тип задачи, на основа на техния опит и интуиция. Експертният метод се използва в случаите, когато оценката не може да бъде извършена чрез друг метод или другите методи са значително по-трудоемки.

Социологическият метод се основава на обработката на специално подготвени анкети-въпросници, на информация от изложби и конференции и т.н.

Стойностите на оценъчните елементи могат да попадат в следните видове **скали**: **интервална** скала, характеризираща се с относителни или абсолютни величини в даден интервал; **порядкова** скала, позволяваща ранжиране на стойностите на някои оценъчни елементи чрез сравняването им с определени опорни стойности; **номинална** скала, показваща само наличието на конкретното разглеждано свойство в дадения програмен продукт.

Процедура по оценяване Целта на модела е чрез използване на вече дефинираната структура и като се следва точно определена процедура, да се определи за всеки даден програмен продукт число, което да характеризира качеството. 1. Приема се, че всяка характеристика на всяко ниво може да приема стойности в интервала [0,1]. При това експертите ползват някои най-общи указания, изготвени предварително. Когато се касае за определяне на стойност чрез измерителен, регистрационен или изчислителен метод, се посочва точният начин за получаване на стойността и как получената стойност, ако не е в интервала [0,1], да се трансформира подходящо. В останалите случаи се дават по-общи указания. Стойностите на всички оценъчни елементи се определят от експерти по един от изброените по-горе методи. 2. Приема се също така, че на всяка характеристика на всяко ниво съответства тегло в интервала

[0,1]. При това, както е обичайно, сумата от теглата на характеристиките, отнасящи се до една характеристика от по-горно ниво, е 1. Всички теглови стойности се определят от експерти, да ги наречем базови, предварително и се отнасят до точно определен тип софтуерни продукти. Причината е, че дадена характеристика може да е изключително важна за даден тип софтуер и тогава тя трябва да получи високо тегло; същата характеристика може да не е особено съществена за качеството на друг тип софтуерни продукти и тогава базовите експерти ще й дадат относително по-ниско тегло. 3.При започване на оценката на качеството на конкретен програмен продукт от даден тип, експертите разполагат с вече готовите: процедура за оценяване, указания за намиране стойностите на оценъчните елементи, теглата към всички характеристики на всички нива точно за дадения тип софтуер. 4.Експертите определят стойностите на всички оценъчни елементи, като използват структурата на модела и указанията и изследват оценявания програмен продукт.5.Нека получените стойности на оценъчните елементи, определящи метриката *M* са e1, e2 , . . . , en, а съответните им предварително зададени

тегла са *w1,w*2,..., *wn.* Тогава стойността на метриката се изчислява по

формулата: ***M = e1 * w1 + e2 * w2 + …+ en * wn..*** 6. Същата схема на пресмятане се прилага за всяко от следващите нива. 6.1. След като всички стойност на метрики M са ни известни, за всеки критерии ***C*** прилагаме формулата: C=M1*w1+M2*w2+…+Mn*wn, където Mi са метриките, определящи критерия C. 6.2. Аналогично, след като всички стойност на критерии C са ни известни, за да получим стойността на всеки фактор F прилагаме формулата: F = C1 * w1 + C2 * w2 +.. + Cn* wn. Тук отново Ci са критериите, които определят фактора F. 6.3. И накрая, след като всички стойности на фактори F са ни известни, за да получим стойността на качеството Q прилагаме формулата: Q = F1 * w1 + F2 * w2+..+ Fn * wn, където Fi са факторите, които определят качеството Q. По този начин получаваме числото Q. То остава в интервала [0,1]. Причината за това е, че всички стойности на тегла и на оценъчни елементи са в интервала [0,1|. Ясно е, че програмен продукт с Q = 1 е с максимално високо качество. В реалността стойност 1 не би трябвало да се достига, защото тя има смисъла на идеал за високо качество. Благодарение на нормализацията става възможно сравняването на получените стойност за Q на различни програмни продукти, както и получаването на самостоятелна представа за качеството на отделен програмен продукт.

Оценка на модела и метода

Предимства: 1. **Простота** на модела от структурна гледна точка. Йерархията е лесно обозрима структура,с ясно видими еднопосочни връзки. 2. Структурата на модела съдържа в себе си и директно следващи възможности за **конструктивност**, т.е. за изграждане на процедура за оценяване. 3. Твърде елементарно е да се създаде компютърна програма за пресмятане на оценката, доколкото действията са прости и еднообразни. Нещо повече, без такава **автоматизация**, оценяването става твърде неефективно. 4. Крайният резултат е едно единствено число в зададен интервал и това допринася за избягване на двусмислия и неясноти относно качеството на оценяване програмен продукт.

Недостатъци: 1. Процедурата за оценяване съдържа твърде много елементи на **субективност**: Указанията за това как да се определят стойностите на оценъчните елементи, както се видя по-горе, се изготвят предварително от експерти; При самото определяне на стойностите на част от

оценъчните елементи експертите, оценяващи конкретния програмен продукт, също не могат да не проявят субективност; оказва се, че твърде малка част от тези оценъчни елементи могат да бъдат оценени обективно, повечето получават стойности на основата на експертна оценка; Теглата на всички нива за всеки тип програмни продукти се определят предварително от експерти. 2. Огромна **трудоемкост**, поради посочената вече необходимост от многостранна и разнообразна работа на експертите по предварително определяне на тегла, на указания за оценяване, както и на отделен труд по определяне стойностите на оценъчните елементи за всеки конкретен програмен продукт. 3.Моделът е топологично универсален всеки тип програмни продукти изисква свое **собствено множеСгпво от тегла*.***

Моделът на Boehm COCOMO

Цели и основни идеи: Основополагащата му идея е използването броя редове първичен код.

Същност на модела

За оценяване на трудоемкостта на даден софтуерен проект се прилага формулата: ЧМ = 2.4 * ХРПК^1.05, където **ЧМ** означава брой човекомесеци, **ХРПК** означава хиляди реда първичен код. За оценяване продължителността на разработване на софтуерния про-

ект формулата е: В = 2.5 * ЧМ^0 38, където В е срокът на разработване в месеци. Формулите се прилагат при следните **предположения*:***

а) редовете първичен код: се броят без коментарните редове, принадлежат на крайния продукт, не включват използваните стандартни програми; б) включват се с.амо фазите проектиране, програмиране и оценка, включително усилията по управлението и документирането по време на тези фази; в) не се включват обучението, планирането и инсталирането на софтуера при потребителя; г) включва се трудът на проектантите и програмистите, но не и този на компютърните оператори, висшите ръководители и секретарките; д) счита се, че един човекомесец е от 19 дни или 152 часа; е) предполага се, че никакви сериозни промени не се правят в продукта след одобряването на документа, който съдържа изискванията към него; ж) двете страни - потребителят и разработчикът – се предполага, че са добросъвестни през цялото време.

Усъвършенстване на модела

Първото усъвършенстване на дотук изложения ***базов модел*** е въвеждането на 3 тина софтуерни проекти **разпространен, полупезависимост *и* вграден.** За всеки от тях формулата е различна. Разпространен – разработва се от малка група в познатите условия на софтуерната фирма. Полунезависим – Има междинно положение м/у разпространен и вграден тип. Вграден – Софтуерът работи свързан с апаратурата, друг софтуерн и изчислителни процеси.

Следващото усъвършенстване е свързано с установяването на факта, че редовете първичен код не биха могли да са единственият параметър за установената формула – модели: **междинен и детайлен.** Има таблица на тях и възможните рейтинги. При извършването на оценката експертите следва да се ръководят оТ тази таблица и за конкретния проект трябва да

определят за всеки атрибут съответния рейтинг. Получените стойности се заместват в формулата, която вече е придобила по-сложен вид: ЧМ = к * ХРПК^Р * А1 * А2 *…* А15, к и р с а коефициенти, които са различни за различните типове софтуер. Разликата в междинния и детайлния модел е в начина на получаване на крайната оценка. Основната разлика е в това, че при детайлния вариант се прилагат различни коригиращи коефициенти (рейтинга) за всяка фаза от производствения процес. По-сложен е и процесът, чрез който от оценките на отделните модули се получават оценки за подсистемите и от тях - за цялостния продукт. При междинния вариант нивото модул се игнорира.

Критика на „ редове първичен код": Няма нито стандарт, нито единно виждане за това какво е „ред първичен код"; Много е трудно, дори за експерти с голям опит да предскажат достатъчно точно в ранен етап на разработването броя на редовете първичен код.

Тема 12. Откриване и поправяне на дефекти. Тестване и настройване.

ОТКРИВАНЕ И ПОПРАВЯНЕ НА ДЕФЕКТИ

Качествен е този програмен продукт, който удовлетворява формулираните към него изисквания.

Всяко отклонение от изискванията ще наричаме **дефект.** Дефектите обикновено се дължат на една или няколко грешки.

Под **грешка** ще разбираме неправилност, отклонение или неволно преиначаване на обект или процес. В зависимост от това, в какъв софтуерен продукт се откриват, грешКите могат да бъдат грешки в проекта, в програмата, в документацията, 8 тестовите данни и т.н. Грешките могат да бъдат: **първични** - неправилности в текста на програмите, подлежащи на непосредствено киригиране; **вторични** - изкривявания на получените резултати. Грещите също са: **технологични** грешки - при въвеждане на програмите или при подготовка на входните данни върху технически носители; **алгоритмични** грешки; **програмни** грешки - неправилно използуване на конструкциите от съответните езици за програмиране; **системни** грешки - свързани с функциониране в определена операционна система.

Основни дейности за откриване и отстраняване на грешки

Настройване (debugging) - локализиране и отстраняване на установени грешки.

Тестване - изследване на програмите за установяване на съответствието им с различни по степен на формализираност характеристикики, правила и изисквания.

Дейностите настройване и тестване се различават по основното си предназначение, по използуваните методи и по нивото на сложност на откриваните грешки. Когато настройването завърши, е ясно, че програмата решава някакъв проблем. Предназначението на тестването е да докаже, че точно това е проблемът, който се иска да бъде решен.

Тестването се осъществява в три основни стъпки: планиране, реализация и отчитане. **Планиране** - се определя целта на тестването, какво да се тества, кога, с какви данни, как и кой

да го извършва. **Реализацията** на тестването се описва чрез сценарии за тестване. **Отчитането** се извършва чрез анализ на документираните резултати и прилагане на критериите за изчерпателността и обхвата на тестване. Подхода за тестване: **структурен** и **функционален.** Целта на **структурното тестване** е да се изберат такива тестови данни, че да се осигури преминаване през всички програмни части на системата. При **функционалното тестване** се проверява правилността на реализираните основни функции. В зависимост от избраните тестови данни и очакваните резултати тестването се дели на: **детерминирано тестване -** при зададени входни данни е напълно определено какви трябва да бъдат получените резултати; **стохастично тестване -** тестваните данни са случайни величини с определено разпределение и се знае разпределението на получаваните резултати. В зависимост от **начина на осъществяване**, тестването може да е **низходящо** , **възходящо** или **смесено** - започва от някакво междинно ниво и се провежда в двете посоки. В зависимост от целта: **за доказване на експлоатационната годност** - проверява се дали програмният продукт е работоспособен; **за атестация** - т.н. пускови изпитания, след успешното завършване на които програмният продукт може да се разпространява; **за пълна функционална проверк**а - дали реализираните функции съответстват на формулираните изисквания; **за проверка на специални програмни свойства**; **за проверка на нови свойства или функции** - реализира се след внасяне на изменения в създадения програмен продукт и се нарича регресионно тестване; **за проверка на работоспособността на системата в реални потребителски условия** - изследва се времето за отговор на системата, продължителността на входно-изходните операции, изчислителните ресурси; качеството на потребителския интерфейс и други. В зависимост от това кой извършва тестването: **вътрешно тестване** от самите разработчици (плюс - необходимо допълнитлно разучаване на програмите, недостатък - програмистите имат склонност към надценяване на възможностите си и увереността, че не допускат грешки. **peer review** - проверка, при която програмистите са разделени на двойки и всеки проверява програмите на партньора си); **независимо тестване** - от експерти, които не са участвали в разработването на програмите. **сертифициране** - проверка и издаване на сертификат, удостоверяващ наличието на определени свойства на ПП.

V-модел за разработване и тестване на софтуер

Плюсове- фазите на тестване са представени на ниво, еднакво с това на фазите на разработване, което позволява от мениджърска гледна точка да се прави директна аналогия за планирането и ресурсите; резултатите от всяка от фазите на разработване могат да се проверяват от групата по тестване, за да се осигури тяхната тестируемост достатъчно рано; своевременното планиране и проектиране на тестовете дава възможност за допълнителни проверки и коментари върху междинните продукти на фазите на разработване.

Автоматизиране на дейностите настройване и тестване

Чрез автоматизация на настройването и тестването се постига: систематичност; подобряване на организацията на тестване; повишаване на надежността на програмната система; документиране на извършваните дейности; автоматично измерване на обхвата на тестването.

Инструменталните средства, подпомагащи настройването, се наричат програми- **дебъгери.** Видове***:*** **разглеждане на текста на програмата** на различни нива на декомпозиране (съвременните дебъгери реализират **многопрозоречно визуализиране** на интересуващите ни

програмни части); **трасиране на изпълнението** - показване на операторите в реда на изпълнението им при конкретните тестови данни; възможност за дефиниране на **контролни точки** и определян на действията при достигане на съответната контролна точка. В зависимост от предназначението си, програмните средства за тестване могат да бъдат анализатори на програми, **генератори на тестови данни(**Тези програмни средства подпомагат или реализират съставянето на

тестови данни в съответствие с избрана стратегия на тестване.**)**, **помощни средства и среди за тестване(**Те подпомагат определени стъпки от процеса на тестване**),** **интегриране на системи за тестване(**подпомагат няколко метода на тестване или да реализират изцяло избран метод на тестване, като автоматизират планирането, генерирането на тестови данни, управляемото изпълнение, анализа на получаваните резултати и оценяване на ефективността на тестването**)**.

Анализатори на програми - реализират избран метод на тестване чрез опериране върху тестващата програма по съответстващ на метода начин.

Статичният анализ е изследване на текста на програмите и извличане на определена информация от него. При **елементарния статичен анализ** текстът на програмата се проследява, като се създава таблица на използуваните имена и срещанията им, цъздава се и тъй нареченият статичен профил на програмата, показващ кои оператори са използувани и колко пъти. При **потоковия статичен анализ** се построява управляващ граф на програмата и граф на обръщенията, отразяващ връзките между отделните програмни части. Потоковият анализ дава възможност да се открият няколко вида грешки: структурни - недостижими програмни части, недопустими пресичания в телата на цикли, рекурсивни обръщания към подпрограми, когато това не е разрешено; в дефинирането и използуването на променливи - използуване на променлива преди да е получила начална стойност, дефиниране на променливи, които по-нататък не се използуват, установяване на случаи при цикли, управлявани от логически израз, в които никоя от променливите, участващи в израза, не променя стойността си в тялото на цикъла; несъответствия и грешки при предаване на параметри между различни програмни части и използването на тези програмни части и параметри. Обикновено потоковият анализ се извършва на две нива: **локално** - за всяка програмна единица; **глобално** - за програмната система като цяло. Недостатъка на потоковия анализ: дълбочината на анализа се постига с цената на големи изчислителни ресурси; откриваните аномалии се делят на такива, които непременно ще се случат и такива, които биха могли да се случат. **Тенденции** при статичните анализатори са разработва-

нето им за програми, написани на различни езици за програмиране и включване на статичните анализатори в интегрирани системи за тестване.

При **динамичния анализ** програмата се изпълнява по управляем и систематичен начин и се изследва нейното поведение, за да се потвърди функционалната й коректност или некоректност. При планирането на динамичния анализ се избира стратегия, подготвят се съответстващи на стратегията тестови данни и се определя начинът на тълкуване на получаваните рзултати.Висове: **подход на черната кутия** - изследваната програмна част се

разглежда като елемент с неизвестно съдържание, към нея се подават входни данни и след изпълнението се получават определени резултати; „**бяла кутия“** - програмата се инструментира по такъв начин, че да се натрупва информация за поведението й по време на изпълнение; ,,**метода на пробите**" за всеки изпълним оператор се дава броят на изпълненията му при конкретни входни данни; вграждане в програмата на твърдения - първоначално са като коментари, но могат да се активизират за целите на динамичния анализ.

Формално-функционалните анализатори реализират „символично“ изпълнение на програмата. Програмата се разглежда като **изчислима функция,** която може да се декомпозира на частични функции, изчислявани върху логическите пътища в програмата с подмножество на входните данни. Сложността на тези анализатори е близка до сложността на системите за доказване правилността на програми и затова практическото им използуване засега е ограничено.

Мушационните анализатори генерират мутанти на изследваните програми чрез внасяне на различни по вид и сложност грешки**.** След това се изследва приложимостта на избрана съвкупност от тестови данни и чувствителността на избран метод към типа на внесените грешки.

Осъществяване на тестването- Натрупаният практически опит показва, че резултатите от тестването установяват наличие на грешки и дефекти твърде късно - тогава, когато програмния продукт е почти завършен. Доказано е, че усилията Е за отстраняване на грешки растат експоненциално в зависимост от дължината t на времевия интервал, определен от момента t1 на допускане на грешка до момента t2 на откриването на грешката: E = к.e^t. За това се разработват специални методи и техники, позволяващи раннот0 откриване и отстраняване на грешки.

Метрики за тестването

Обхват на тестването: Изпълнени инструкции(Брой на изпълнените поне веднъж програмни инструкции); Брой на тестовете(Брой проведени тестове); Брой на тестваните пътища(Брой тествани логически пътища)

Ефективност на тестването: Открити дефекти(Брой открити дефекти); Неоткрити дефекти (Пропуснати при тестването, открити след внедряване на ПП); Бизнес-полза(Разходи, спестени от ранното откриване на дефекти); Re-run загуби(Престой на системата поради неоткрити дефекти); Проверени изисквания(Брой на проверените изисквания); Предотвратени загуби(Ресурси, спестени поради откриване на дефекти преди внедряване на ПП)

Ресурси за тестване: Относителна цена на тестване=(Цена на тестване)/(Цена на разработване); Средна цена за открит дефект=(Цена на тестване)/(Брой открити дефекти); Изпълнение на бюджета=(Реална цена на тестване)/(Планирана цена)

Тема 13. Дейности, осигуряващи разработването – съпровождане, документиране.

СЪПРОВОЖДАНЕ - съвкупността от дейноСТИ, свързани с внасяне на промени във внедрен софтуер.

В зависимост от целите на модифицирането: коригиращо - за поправяне на установени грешки; адаптивно за приспособяване към нова операционна или хардуерна среда; усъвършенстващо - за подобряване на съществуващи или добавяне на нови функционални възможности.

Осъществяване на съпровождането

Съпровождането може да се реализира от разработчиците или от други лица или организации въз основа на договор за съпровождане. Преимуществата на съпровождане от разработчиците е, че те познават добре софтуерния продукт като цяло Основните обобщени етапи на съпровождането са: разучава,не на съществуващия софтуер; модифициране на съществуващия софтуер; проверка на правилността на модифицирания софтуер.

Управление на внасянето на изменения

Етап 1. Идентифициране на проблема и възлагане. Попълва се формуляр, в който се описват одобрените изменения с присвоения им приоритет. Оценява се обемът и сложността на работата и се възлага на един или няколко изпълнителя. ***Етап 2.*** Проектиране на измененията. Анализират се възможните начини за осъществяване на измененията и се определя кои програмни части ще бъдат модифицирани. ***Етап 3.*** Програмиране и тестване. През този етап съответните програми се изменят и се тестват локално. ***Етап 4***. Интегриране на програмната система и системно тестване. Извършва се регресионно тестване и пълни приемни изпитания, след успешното приключване на които новата версия се пуска за разпространение. През този етап потребителите се запознават с бюлетина за изменение ако е необходимо, се провежда обучение за работа с тази нова версия***. Етап 5***. Разпространение на новата версия. Всеки от потребителите на ПП може да получи новата версия и съответната документация. Конкретните финансови условия на разпространение зависят от съдържанието на новата версия и от регламентираните отношения между собственика на ПП и потребителите.

Разходи и цена на съпровождането

Белади и Леман предлагат модел на процеса на съпровождане. Съгласно този модел, ако една програмна система се променя непрекъснато, то нейните обем и сложност нарастват. Оценка за разходите на съпровождане: M = p + K^(c-d), където M са общите разходи за съпровождане на програмната система; стойността на p представя продуктивните усилия за анализ, проектиране, програмиране и тестване; c е мярка на сложността, предизвикана от липсата на структурно проектиране и документиране. Тази сложност се редуцира с *d* - степента, до която групата по съпровождане е запозната с софтуера. Константата K е емпирична, като стойността ѝ зависи от средата. Тя се определя чрез регресионен анализ на разходите за съпровождане на реални проекти.

Автоматизирани средства, подпомагащи съпровождането

В зависимост от предназначението си: средства, улесняващи разучаването на програмите. Те реализират преформатиране в стандартен вид, създаване на справки за срещаните имена,

статичен анализ на потока на данните и потока на управление и др.; средства, улесняващи анализа и проектирането на внасяните изменения, например чрез прилагане на метрики върху оригиналната и променената програма; средства за управляемо внасяне на промените, като се следи за правата на достъп, автоматично се документират промените, управлява се последователността на осъществяване, регистрирането и оптималното извършване на модифицирането; средства за провеждане и анализ на резултатите от регресионното тестване; средства за документиране на промените.

ДОКУМЕНТИРАНЕ

Разработването и използването на всяка софтуерна система е свързано със създаването на множество документи, които са с различно предназначение, структура и съдържание. Водове: **документи, свързани с процеса на създаване на софтуер** - тази документация зависи от стила на управление на проектите и се регламентира от действащи в конкретната софтуерна организация вътрешни правила и стандарти, те определят каква е структурата на всеки документ, кой и кога го създава, какви са процедурите за утвърждаване, използване и променяне, къде и колко дълго се съхранява; **документи, описващи създавания програмен продукт** - В зависимост от предназначението си - **съпровождаща** или **експлоатационна**. Предназначението на **съпровождащата** документация е да осигури информацията, необходима за разучаване и за внасяне на изменения в програмите. Тя може да включва: описание на функционалните и не-функционални изисквания към ПП; описание на архитектурата на софтуерната система - ог какви основни компоненти се състои и какви са връзките между тях; описание на спецификациите и на детайлния проект; текст на програмата на изходен език; описание на програмата, което включва предназначение, описание на логиката, структура, компоненти, използвани методи и алгоритми, входна и изходна информация, процедури за извикване и зареждане и др.

Предназначението на **експлоатационната** документация е да осигури информацията, необходима за използването на софтуерната система от крайни потребители или администратори, които отговарят за функционирането на системата. Документацията за администратора се състои от:инструкция за инсталиране,тя съдържа указания как да се инсталира системата в конкретна потребителска среда и какви са техническите изисквания към хардуерната конфигурация; ръководство за администратора - описва връзките с други системи, обяснява извежданите съобщения и възможни действия при получаването им, определя същността и начина на извършване на някои сервизни функции като архивиране, възстановяване на системата след срив, преконфигуриране и др.

Потребителската документация може да се състои от две части - обучаваща и справочна. Обучаващата част е предназначена за начинаещи потребители, а справочната - за потребители с по-голям опит, к0ил'0 се нуждаят от информация за възможностите на системата. Справочната част обикновено представя основните функции, подредени по определен начин за бързо търсене.

Изисквания към потребителската документация: **правилност**. Трябва да има пълно съответствие между функционирането на програмната система и описанието й в

документацията, след всяка нова версия трябва да се обновява и документацията, като се проверяват и описаните примери, така че потребителят да не бъде насочван погрешно; **пълнота и структурираност**.-да има подробно съдържание и дори каква последователност на четене да се следва от потребителите с различна квалификация и опит; **стилътп на изложение** да е ясен, недвусмислен и съобразен с терминологията в съответната област на приложение, да се избягват тривиалните обяснения и примери; ръководството за потребителя да бъде относително **затворена система***,* съдържаща необходимата и достатъчна информация, да се избягват сложните препратки дори с цената на повтаряне на информация, да няма препратки към трудно достъпни източници.

14. Управление на конфигурациите. Автоматизация – CASE средства.

Управление на софтуерните конфигурации

Увеличаващите се разходи за съпровождане изискват ефективно управление на внасянето на промени, така че да не се преустановява (временно или изобщо) реалното използване на софтуерните системи. Обикновено тези системи са внедрени с отчитане на изискванията на конкретния потребител (customization), т.е. те съществуват във версии, които могат да се различават по състава си и реализираните функции. Поддържането на версии изисква управление на промените не само в програмите, но и във всички останали елементи, свързани с използването на даден ПП. От друга страна, управлението на промените се превръща в основна дейност на софтуерното производство поради осъществяването му в динамична среда на променящи се изисквания и очаквания. Удовлетворяването им изисква систематичен и управляем подход през целия ЖЦ. Така се стига до идеята за още една глобална дейност (наред с измерване, осигуряване на качеството и документиране), наречена **Управление на софтуерните конфигурации** (УСК). Следвайки речника по Софтуерни технологии на IEEE, ще дадем следните дефиниции:

Под **конфигурация** се разбира състоянието на компютърна система (или нейни компоненти), така както е определено чрез броя, естеството и взаимосвързаността на съставящите я части. Според тази дефиниция софтуерната конфигурация е съвкупността от всички елементи, необходими за функционирането на даден софтуерен продукт. Това могат да бъдат изходни текстове на програмите, обектен и изпълним код; командни файлове или процедури, необходими за свързване и изпълнение на програмната система; използвани системни файлове, помощни средства, файлове с данни и др. Изключително важни са междинни продукти като спецификации и проекти; спомагателни продукти като тестови планове, сценарии и данни; документация - и съпровождаща, включваща описание на програмните части, и експлоатационна, включваща ръководство за потребителя, ръководство за инсталиране и др.

Управление на софтуерните конфигурации (Software Configuration Management - SCM) е глобална дейност, използваща технически и административни похвати, за да идентифицира и документира функционалните и физически характеристики на всеки елемент на конфигурацията; да контролира промените на тези характеристики; да регистрира и съобщава достигнатия етап на внасяне на изменения, както и да проверява съответствието с определени изисквания. Необходимо е разграничаване на съпровождането от УСК. Съпровождането обхваща само дейности, свързани с внасяне на промени във внедрен софтуер. Управлението на софтуерната конфигурация е фонова(umbrella) дейност през целия ЖЦ, която започва със стартирането на софтуерния проект, продължава в процеса на разработване и завършва с преустановяване на използването на последното копие на програмния продукт [9]. За нея IEEE създава стандарт, който по-късно е приет за индустриален стандарт и от ANSI (American

National Standards Institute). Съгласно този стандарт УСК се осъществява чрез следните четири дейности:

• **Идентифициране на софтуерната конфигурация** - определяне на общата структура на разработвания продукт, избиране на елементите на конфигурацията и регистриране на техните функционални и физически характеристики в техническата документация. Избраните елементи на конфигурацията трябва да съответстват на структурата на продукта, да представляват интерес за, проследяване на развитието им и да имат уникален идентификатор (label). Препоръчва се идентификаторът да се състои от две части - постоянно име (по възможност мнемонично) и номер на версията.

• **Контролиране на софтуерната конфигурация** - оценяване, координиране, одобряване/отхвърляне и реализиране на промени на елементи от конфигурацията след формалното им идентифициране. За всеки ПП съставеното описание на софтуерната му конфигурация задава и връзките между елементите й, така че при всяка промяна в един елемент да може да се проследи кои други елементи са засегнати. Осъществява се управление на заявките за изменение, контрол на версиите и управление на внасянето на измененията. Обикновено се поддържа стандартен формуляр-заявка за изменение, в който се описва исканото изменение и се обосновават причините за него. Специален експертен съвет разглежда постъпилите заявки и всяка от тях определя дали да се отхвърли, дали да се включи в подготвяната нова версия или да се отложи за някаква следваща версия.

Критерии за групиране на измененията на поредна версия могат Да бъдат:

а) подреждане на измененията по технически или процедурни причини;

б) подреждане по неотложност на извършване на промените;

в) оценяване на необходимите ресурси за реализиране на измененията;

г) анализиране на степента на отражение на промените върху останалите части от системата. Би трябвало да се минимизира броят на засегнатите от измененията програмни части, като тези, от които зависи работоспособността на системата да се променят само след доказана необходимост и наличие на ресурси;

д) сходните промени да се групират в една версия.

Задължително е реализирането на приемственост между версиите. Това означава, че всяка нова версия трябва да реализира всички функции на предишната и евентуално да предлага нови. Трябва да се осигури пълно съответствие между ПП и документацията. му, особено потребителската. Обикновено се поддържа Бюлетин за измененията. В него на достъпен за потребителите език се описват промените в новата версия. Ако броят на бюлетините стане много голям, се препоръчва преструктуриране и „преиздаване" на потребителската документация (в хартиен и/или електронен вид). Чрез контролиране на софтуерната конфигурация се предотвратяват:

- многократно съпровождане при всеки потребител (откритите дефекти трябва да се отстраняват във всички копия, и то по един и същ начин);
- едновременно обновяване на един и същ елемент от различни разработчици;
- неправомерно променяне на общи елементи като подпрограми, класове, заглавни файлове, библиотеки и др.;
- неправилно управление на версиите.

• **Следене на състоянието на софтуерната конфигурация** (Status Accounting) - регистриране на заявките и извършените промени протоколи (logs) и архиви и отразяването им в съответни справки (за транзакциите, за промените, за развитието на елемент от конфигурацията, за използваните ресурси и т.н.)

• **Проверяване на софтуерната конфигурация** (Configuration Auditing) - за спазване на регламентираните процедури и за качеството на създаваните версии- междинни (baselines) и потребителски (releases).Проверяваните изисквания съответстват на броя, размера и типа на разработваните елементи на софтуерната конфигурация. Проверките се реализират чрез наблюдения, въпросници и разговори. Те се провеждат регулярно в определени контролни точки на проекта, обикновено от квалифицирани специалисти, неучастващи в разработката.

Изисквания към планирането и осъществяването на дейностите по управлението на софтуерните конфигурации са формулирани в стандартите от серията ISO 9001:2000, CMM, IEEE-SCM 828-1990 и IEEE-SCM 1042-1987 (г:1993).

Автоматизиране

Същността на софтуерния парадокс е, че разработчиците на софтуер, които се занимават с автоматизиране на работата на другите, все още не са направили достатъчно за автоматизиране на собствената си дейност. Статистически данни показват, че разходите за обработка на информацията се удвояват на всеки 5 години, а производителността на програмисткия труд се удвоява на всеки 25 години. Това обуславя актуалността на проблема за автоматизация на софтуерното производство(АСП). Предназначението на автоматизиращите средства е да поддържат избраните методи за разработване на софтуер, да улесняват управлението на проектите и на цялостния процес на създаване на ПП, да извършват трансформации от едно представяне на софтуерните продукти в друго и да проверяват правилността им. В зависимост от прилаганите автоматизиращи средства, могат да се разграничат два основни подхода - използване на индивидуални средства и използване на интегрирани среди. Ще се спрем на основните характеристики и особености на всеки от тези два подхода.

14.4.1 Автоматизация чрез индивидуални средства

Този подход е исторически първият и се състои в използването на отделни (stand-alone) „полезни" програми, улесняващи извършването на някаква дейност при създаването на софтуера. Първоначално средствата са били компилатори, асемблери, свързващи редактори и дебъгери, осигуряващи директна помощ за програмирането. Едва в началото на **70**-те години се появяват средства, подпомагащи и други дейности. Съществуващите индивидуални средства за АСП могат да бъдат класифицирани по различни критерии по функциите им (т.е. кои дейности подпомагат), по използването им в различни фази от жизнения цикъл, по основните им потребители (мениджъри, програмисти, отговорници по осигуряване на качеството и др.), по степента им на сложност на усвояване и прилагане и др. В зависимост от автоматизираните дейности, те се разделят на следните групи:

а) средства за програмиране - езиково-ориентирани редактори, транслатори, свързващи редактори, дебъгери, средства за управляемо изпълнение на програми и др.

б) средства за тестване - анализатори на програми, генератори на тестови данни, средства за управление на тестването;

в) средства, подпомагащи управлението на проекти. Чрез тези средства мениджърът на софтуерната разработка може:

- да оценява предварително трудоемкостта, стойността и продължителността на проекта и броя на хората, необходими за реализацията му. Оценката се прави чрез въвеждане на индиректна мярка за размера на проекта и анализиране на някои общи характеристики - сложност на проблема, опит на разработчиците в тази приложна област, зрялост на процеса на разработване и др.;
- да определя основните задачи и да създава графици за изпълнението им;
- да проследява развитието на проекта и при необходимост да го планира с промяна на ресурсите и сроковете;
- да оценява производителността на труда и качеството на създавания продукт чрез прилагане на подходящи метрики;
- да проследява удовлетворяването на изискванията.

г) средства, подпомагащи документирането

Те осъществяват създаването, оформянето и отпечатването на документи. В тази група са текстообработващите програми с възможности за въвеждане, редактиране, проверка на граматическата правилност и стил на текст. Графичните редактори позволяват илюстрирането на текста с диаграми, схеми и произволни изображения. Издателските системи реализират

форматирането (предпечатната подготовка) на оформените в съответствие с определени стандарти документи.
д) средства, подпомагащи съпровождането
- средства за проследяване на изискванията
- средства за управление на внасянето на изменения
- статични и динамични средства за възстановяване на детайлния проект по изходните текстове на софтуерната система с цел повторно разработване или промяна, за да се подобрят някои характеристики на софтуерната система (reverse engineering и re-engineering).
е) средства за управление на софтуерните конфигурации - идентифициране и контрол на версиите, изграждане (конфигуриране) на софтуерна система;
ж) средства за анализ и проектиране. Те позволяват създаване и оценяване на модел на софтуерната система, която ще се разработва. Могат да поддържат различни методи на проектиране - структурни или обектно-ориентирани.
з) средства за прототипиране и симулиране. Тези средства представят някои функции или характеристики на поведението на софтуерни системи, работещи в реално време.
и) средства за проектиране и разработване на потребителски интерфейс.
й) езикови процесори. Тези средства са за използване на р а з л и ч н и езици - за специфициране, за описание на проекти, за а в т о м а т и ч н о генериране на текста на програмите и т.н.
Изследвания за използване на индивидуални средства за АСП показват, че разпределението им по различните фази и функции от ж и з н е н ия цикъл не е равномерно и че успехът на нови методи в софтуерното проводство зависи в голяма стенен от това, дали тези методи се подпомагат и от съответни автоматизиращи средства.
Основно предимство на подхода за АСП с индивидуални средства е, че те не са толкова скъпи и всяка софтуерна фирма може да си ги позволи. Освен това тези средства улесняват една или няколко дейности и повишават производителността на персонала, участващ в създаването на софтуер. Недостатъците на подхода са няколко. Преди всичко, индивидуалните средства са обикновено хардуерно зависими и са предназначени за точно определена операционна и програмна среда, Използването на няколко независими средства принуждава разработчиците да разучат различни потребителски интерфейса. Липсата на интеграция намалява производителността, защото не е възможно последователно извикване на независими средства, без да има дублиране на общите дейности.

14.4.2 Автоматизация чрез интегрирани среди

Полезността на индивидуалните средства за АСП е безспорна, но интегрирането им за съвместно използване би улеснило:
- предаването на информация между тях;
- ефективното изпълнение на глобални дейности като документиране, осигуряване на качеството и управление на софтуерните конфигурации;
- реализацията на потребителски сценарии за решаване на конкретен проблем.
Естествено развитие на идеята за автоматизация с индивидуални средства е използването на групи от средства, които са проектирани да работят съвместно. Един от приносите на операционната система UNIX към разработването на софтуера е предоставянето на т.н. PWB (Programmer's workbench), при който интеграцията се осъществява чрез общи формати на файловете, възможности за управление на версиите (SCCS) и за изграждане на софтуерна система чрез описание на съставящите я части (MAKE). Идеята за разрастващо се изграждане на средства чрез „навързване" на по-малки средства е изключително полезна. По-нататък се преминава през средите за програмиране на Ада, докато Се достигне до сегашната обща теория за изграждане на интегрирани среди.
В [6] са формулирани следните изисквания към интегрираните среди за АСП:
- Да осигуряват механизъм за общо използване на информацията от сички средства в средата;
- Да допускат директно извикване на всяко средство;

- Да поддържат решаването на всяка конкретна задача при реализацията на софтуерния проект чрез подходящо съчетаване на средства;
- да улесняват комуникациите между всички участници в процеса на създаване на софтуер;
- да натрупват статистическа информация, която да се използва за подобряване качеството на процеса и продукта.

В зависимост от начина на свързване на група от автоматизиращи средства в единна среда, интеграцията може да бъде:

а) еклектична интеграция съществуващи индивидуални средства се обединяват в система чрез създаване на програма-монитор, извикваща всяко от средствата;

б) интеграция чрез данните - използване на общ модел на данните. Този вид интеграция може да бъде с различно ниво на сложност обмен на данни между две средства чрез програма-конвертор, използване на обща съвкупност от прости символни файлове или чрез система за управление на обекти;

в) интеграция чрез потребителския интерфейс. В този случай средствата в системата използват общ стил и съвкупност от общи стандарти за връзка с потребителя;

г) интеграция чрез дейностите, които се поддържат. Този тип интеграция се основава на модел на процеса, който определя основните извършвани дейности, резултатите от тях, потока на данни и потока на управление. Определено е и кои средства в интегрираната среда кои точно основни дейности поддържат.

14.4.2.1. Видове интегрирани среди

Всяка интегрирана среда обединява в едно приложение няколко софтуерни средства, поддържащи специфични дейности в процеса на създаване на софтуер. Чрез интеграцията се постига:

- хомогенен и логически последователен потребителски и н т е р ф е й с ;
- лесно извикване на всяко средство и на верига от средства;
- достъп до общо множество от данни, поддържани по ц е н т р а л и з и р ан начин.

Удобствата за работа, предоставяни от някои среди за ACII, могат да допринесат за въвеждане в софтуерната организация на нови методи и техники.

В зависимост от предназначението си, интегрираните среди могат да бъдат за:

а) планиране и моделиране на бизнесинформационни системи. Този клас включва продукти, подпомагащи идентифицирането и описанието на сложни бизнесдейности. Те се използват за построяване на обобщени модели на предприятие, за да се оценят общите изисквания и информационни потоци и да се определят приоритетите в разработването на информационните системи. Средствата, интегрирани в такива продукти, включват графични редактори (за създаване на диаграми и структурни схеми), генератори на отчети и генератори на справки за срещането на отделни елементи.

б) анализ и проектиране. Съвременните средства автоматизират най-често използваните подходи за анализ и проектиране — структурния, обектно ориентирания и подхода на Джаксън. Те обикновено включват един или повече редактора за създаване и модифициране на спецификации и други средства за анализирането, симулирането и трансформирането им. Например Excelerator има редактори за създаване на диаграми на потока на данните, на блок-схеми и на диаграми същност—връзка. Той включва още редактор и симулатор за създаване и тестване на макети на системните входове и изходи (форми и отчети), както и генератор на код, който създава първичен код на Кобол на основата на блок-схемите.

Възможностите на средствата от този клас зависят от:

—: нивото на формализираност на използваната нотация. Ако тя е неформална (структуриран английски или друго свободно текстово описание), то се осигурява само редактиране и съставяне на документ. Ако нотацията е полуформална (без точна семантика, но да е възможно да се проверява синтаксисът) или с формално определени синтаксис и семантика (крайни автомати или мрежи на Петри);

— от вида на приложението — дали преобладава обработката на данните, както е в банковите и счетоводни системи, или управленските функции;

в) създаване на потребителски интерфейс Смята се, че потребителският интерфейс е определящ за пазарната реализация и използване на всяка софтуерна система. Затова са създадени интегрирани среди, които позволяват на разработчика да създава и да тества лесно компонентите на потребителския интерфейс и да ги свърже с приложната програма. Те включват:

— графични редактори за създаване на прозорци, диалогови кутии, икони и ДР-;

— симулатори за тестване на създадените компоненти преди интегрирането им с приложението;

— генератори на първичен текст;

— библиотеки за поддържане на генерирането на изпълним код.

г) програмиране

Те включват текстов редактор за създаване и променяне на текста на програмите, компилатор, свързващ редактор и дебъгер. Характеризират се с:

— удобен потребителски интерфейс;

— управление на създаваната по време на сесия информация — файлове с първичен текст, междинни, обектни и изпълними файлове. За ускоряване на съставянето и тестването на програмата се съчетават компилатор с интерпретатор и свързване с отчитане на направените промени. Примери за такива среди са Turbo C++, Turbo Pascal, Microsoft C++ Developer Studio.

д) верификация и валидация

Такива среди подпомагат модулното и системно тестване и обикновено включват:

— статичен анализатор за създаване на управляващ граф на програма и граф на извикванията;

— средство за инструментиране на програмата и за проследяване (трасиране на изпълнението) при динамичен анализ;

— генератор на тестови данни;

— управляваща програма — средство за реализация на тестването, което създава, съхранява и поддържа тестови данни, сценарии, резултати и документация.

е) съпровождане

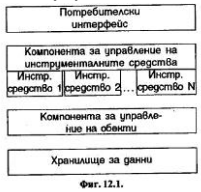
Средата за съпровождане управлява внасянето на промени, създаването и контрола на версии и за управление на софтуерните конфигурации.

ж) управление на проекти

Подпомагат се планирането, съставянето на графици и текущото следене на изпълнението на проекти.

14.4.2.2. Принципи на изграждане и архитектура на интегрираните среди

Предназначението на АСП-средата е да подпомага цялостния процес на с ъздаване на ПП, като поддържа хранилище (repository) на информацията, необходима за осъществяване на софтуерните разработки. Поддържането и използването на информацията от хранилището се осъществява чрез интегрираща архитектура, представена на фиг. 12.1. Основните компоненти са: база от данни, в която да се съхранява информацията, система за управление на обекти, чрез която да се управлява променянето на информацията, механизъм за управление на инструменталните средства (за координиране на използването им) и потребителски интерфейс. Повечето модели представят тези компоненти като слоеве.



Фиг. 12.1.

Ще опишем накратко основните й компоненти.

Потребителският интерфейс осигурява удобна и ефективна работа със системата. Протоколът на представяне е множество от указания, чрез следването на които всички АСП-средства в

системата се използват по подобен начин. Така екраните имат едно и също разпределение на отделни области с еднакво предназначение; има правила за имената и организация на менюта, икони и обекти; стандартизирано е използването на клавиатура и мишка; описан е общ механизъм за достъп до средствата. С използването на протокол на представянето се постига унифицираност.

Слоят на индивидуалните средства включва самите средства и програми за управлението им. Препоръчва се управляващите програми да са проектирани и реализирани така, че да се поддържа динамична съвкупност от индивидуални средства. Това би позволило на потребителя да съставя нужната му конфигурация от средства и да я променя, като добавя, изключва или модифицира някои средства. Ако в системата се поддържа многозадачна работа, управляващите програми осигуряват синхронизация при прилагане на средствата, регулиране на потока на данните, следене на правата за достъп и прилагане на метрики за ефективността на използване на всяко от средствата.

Слоят за управление на обекти осъществява интегрирането на средствата с данните от хранилището и управлението на софтуерните конфигурации.

Той е съвкупност от програми, които за всяка заявка идентифицират, обектите от съответната софтуерна конфигурация и ги представят във вид, подходящ за съответното средство; поддържат различните версии, управляват планирането, внасянето и документирането на промени, регистрация и поддържане на описание на всеки елемент на софтуерна конфигурация.

Слоят за управление на хранилището включва базата данни на системата и функциите за управлението й. Идеята за централизирано съхраняване на данните присъства във всеки модел на интегрирана среда, макар и под различни имена: АСП-база от данни, база на софтуерните разработки, хранилище и др.

Съхраняваната информация може да се раздели на две: обща информация (свързана със софтуерната фирма като цяло и стила на работа в нея) и информация за всеки конкретен софтуерен проект. Общата информация може да включва описание на организационната структура в софтуерната фирма, съвкупността от вътрешни правила и стандарти, които трябва да се спазват, описание на използваната методология на разработване, процедури за извършване на основни дейности и др.

Информацията за конкретен проект може да бъде:

— елементи на софтуерните конфигурации (първичен текст на програми, обектни модули, описания за свързване, изпълними програми, описание на връзките между елементите на дадена софтуерна конфигурация и др.);

— информация за провежданите дейности по осигуряване на качеството (планове, сценарии и резултати от тестване и проверки; резултати от прилагане на метрики, резултати от статистическа обработка на данните);

— информация, свързана с управление на проекта — планове, оценки на трудоемкост и продължителност, графици, отчети от проследяване на проекта в определени контролни точки);

— документация — съпровождаща и потребителска.

Съдържанието и начинът на организация на данните в хранилището се определят при конкретната реализация на концептуалния модел.

Освен обичайните функции на СУБД към компонентата за управление на хранилището са формулирани някои допълнителни изисквания:

— да осигурява интегрираност на данните — да проверява всички елементи така, че да не се допуска дублиране (редундантност) на данните; да осигурява съвместимост между свързаните обекти, автоматично да извършва последователните модификации, когато промяната на един обект изисква промяна на всички свързани с него обекти;

— да осигурява механизъм за използване на информацията от множество разработчици и различни инструментални средства; да управлява многопотребителския достъп до данните и чрез защитни механизми да предотвратява наслагване на промените;

— да осигурява интеграция на данните и индивидуалните средства в системата чрез поддържане на механизми за трансформиране, управляем достъп и защита.

Така се постига съхраняването и обработването на сложни структури от данни, които се използват ефективно от съответните инструментални средства в системата, и то така, че разработването на ПП става в съответствие с избраната методология.

14.4.2.3. Жизнен цикъл на софтуерните среди

Практическото използване на софтуерните среди преминава през следните шест фази:

а) избор на софтуерна среда. През тази фаза трябва да се направи проучване на софтуерния пазар и да се избере софтуерна среда, която е най-подходяща за дадена организация. Основните критерии за избора са:

— методологията на разработване на софтуер в конкретната организация, т. е. съвкупността от прилагани модели, методи, техники и стандарти. В някои случаи чрез използването на определена среда може да се премине от стихийно разработване към систематично и организирано разработване с придържане към поддържаната от софтуерната среда методология;

— наличния хардуер в софтуерната организация. Обикновено използване-то на софтуерните среди изисква значителни изчислителни ресурси, които да не могат да бъдат осигурени от компютрите, с които разполага организацията;

— приложната област, в която ще бъде създаваният софтуер;

— цената на средата. По данни от [1] въпреки обещаваното увеличение на производителността на труда от 40—100% цената за закупуване, инсталиране, настройване и усвояване на работата със средата може да бъде неприемливо висока.

б) настройване на средата към специфични за организацията изисквания

През тази фаза трябва да се създаде версия на средата, която да съответства на използваната в организацията платформа и модел на процеса на разработване. Това изисква определяне на стойностите на параметри, определяне на елементите на системата за управление на обекти, избор на съвкупност от средства и цялостно документиране на получения вариант на средата.

в) инсталиране и експериментално използване на средата

Организира се обучение и пробно използване, за да се оцени полезността на средата. Обикновено трябва да се преодолява съпротивата на разработчиците заради променения стил на работа и съпротивата на мениджърите заради големите инвестиции с неизвестна възвращаемост.

г) използване и еволюция на средата

Препоръчва се натрупване на статистически данни, въз основа на които да се оцени ефектът от използване на средата. Поддържането на обратна връзка с доставчиците би осигурило подобряване на функционирането ѝ и своевременно получаване на нови версии.

д) преустановяване на използването на средата

Възможни са две ситуации — отказ от използване на автоматизиращи среди или замяна на използваната среда с друга. Във втория случай заради съпровождането на разработени вече системи може да се наложи паралелно функциониране на двете среди.

14.4.2.4. Преимущества и недостатъци на автоматизацията чрез интегрирани среди

Потенциалните ползи от прилагането на средства за автоматизация са:

— повишаване на систематичността и управляемостта на софтуерните проекти;

— намаляване на стойността на разработването и особено на стойността на съпровождането;

— подобряване на качеството на софтуера;

— ускоряване на процеса на разработване, т. е. намаляване продължителността на софтуерните проекти;

— повишаване на производителността на труда;

— препоръчвани техники стават реално използваеми поради подпомагането им от софтуерни средства.

Независимо от тези преимущества АСП-средите имат все още ограничено използване. Основни причини за това са:

— няма публикувани изследвания за резултатите от практическото използване на средите;

— няма натрупани данни и подходящи метрики за измерване, как средите влияят на производителността на софтуерните разработчици;

— средите са големи и сложни софтуерни системи. Те изискват значителни инвестиции за закупуване на самите среди, за подготовка и осъществяване на внедряването им. Функционирането им е свързано с големи изчислителни ресурси. Всички тези изисквания не могат да бъдат удовлетворени в неголемите софтуерни организации.

Сложността на съвременните софтуерни системи изисква автоматизация през различните фази на разработването. Независимо от трудностите при приспособяване на автоматизиращите средства към стила на работа в конкретната организация бъдещето на софтуерната индустрия е немислимо без създаването и усъвършенстването на АСП-средствата.

15. Осигуряване на качеството на софтуера.

В глава 6. бяха разгледани проблеми на качеството на софтуерния продукт. Производството на софтуер, както видяхме в глава 2. и специално в 2.3.8., е резултат на процес. Този процес има свои характеристики, които могат да имат различна степен на съвършенство. Следователно може да се говори за качество и на софтуерния процес. Близко до ума е, че при по-добро качество на софтуерния процес по принцип трябва да се очаква и по-добро качество на софтуерния продукт. Например, ако процесът на планиране и създаване на тестовите данни за определен програмен продукт се извърши грижливо и компетентно, т. е. с високо качество, може да се очаква, че продуктът ще има повисока стойност на фактора надеждност. Следователно осигуряването на качеството на софтуера (ОКС) не може да не се свърже с планирането и прилагането на подходящи мерки през целия производствен цикъл. Това е изразено много точно в [1] по следния начин:

"Качеството е едновременно философия и съвкупност от ръководни принципи, които са основата на една непрекъснато подобряваща се организация,интегрираща:
фундаментални техники за управление, постоянни усилия за усъвършенстване,технически средства, всичко това в рамките на един дисциплиниран и целенасочен метод."

Следствие от тази синтезирана мисъл впрочем е, че качеството не може да бъде единствено задача, специално възложена на дадено лице или дори група, а трябва да се разглежда и като приоритетна отговорност на всеки, участващ в
разработването на софтуерния продукт. Следователно методите и конкретните мерки на ОКС трябва да са такива, че да мотивират, ангажират и задължават всеки да работи за качеството на продукта. ОКС е тясно свързано с жизнения цикъл на програмния продукт, по-точно с всяка негова фаза. Планирането му следва да се извърши още в началните
фази и да обхваща всички следващи фази. Всеки междинен продукт, резултат от дадена фаза, заедно със своите характеристики следва да бъде точно дефини-
ран. На тази основа следва да се предвидят начини за обективно установяване доколко удовлетворително е реализиран този междинен продукт (за крайния
продукт казаното се подразбира). За да се осъществи тази дейност по един систематичен и ефективен начин, теорията препоръчва да се изработи *програма за осигуряване на качеството на софтуера (ПОКС).*

Компоненти на програмата за осигуряване на качеството

фактори

Няколко фактора оказват важно влияние върху ПОКС [7]. Без да се впускаме в подробности, ще ги изброим:
изисквания към графика;
разполагаемия бюджет;
технологичната сложност на продукта;
предполагаемия размер на продукта;
относителния опит на разработчиците;
хардуерните и софтуерните ресурси, предвидени за процеса на разработване;
изискванията на договора за възлагане.
В момента на създаването на ПОКС трябва да се анализира и установи доколко всеки от горните фактори (а евентуално и някои други) ще й окаже влияние.

Прегледи

Под преглед (review) се разбира дисциплинирана групова дейност, насочена към изследване на продукт или процес. Ефективното изпълнение на прегледа изисква умело съставяне на групата с оглед комбиниране различните необходими квалификации. Отличават се 3 вида прегледи:

А. **Пробег** (walkthrough). Това е неформален преглед на софтуерния продукт. Обикновено не се подчинява на строги правила. Прилага се най-често върху първичния код на междинните продукти.

Б. **Инспекция** (Inspection). Това е дисциплиниран формален преглед на всякакъв тип продукти.

В. **Проверка на конфигурацията** (Configuration Audit). Много често крайното приемане на софтуерния продукт се основава на редица проверки на конфигурацията. Целта им е да установят доколко крайният продукт удовлетворява изискванията, формулирани първоначално. Тези проверки се делят на 2 категории:

В.1. **Функционални**. Основната цел на функционалните проверки е да преодолеят ефекта от многобройните тествания и съответни корекции. Напълно е възможно при установяване на дадена грешка в процеса на разработване и последващото нейно отстраняване да е била внесена друга грешка. Колкото по-
дълго се разработва даден продукт, толкова повече опасността от такива "вторични", неотстранени грешки нараства. Крайната функционална проверка цели да покаже пълната липса на такива остатъчни грешки.

В.2. **Физически**. Тази категория проверки се съсредоточава върху съответствието на продукта с изискванията на договора по отношение на документацията и сроковете. Те се правят след функционалните.

Оценяване

Оценка се прави обикновено от един специалист. Целта й е да се установи съответствие на всички характеристики на всеки продукт с формулираните изисквания. По същество това е функция по контрол на качеството на продукта. Тя
обаче осигурява информация и*за процеса на разработване. Поради тези причини всички дейности по оценяване трябва да се планират подробно и резултатите им да се използват възможно най-пълноценно. За отделните фази следва да
се предвидят следните примерни дейности по оценяване.

1. **Изисквания към продукта**. Дори и в използвания модел на жизнения цикъл да няма точно такава фаза, не може на един първоначален етап да не се формулират тези изисквания, така че те да отразяват на едно първо, но доста-
тъчно точно приближение възгледите на потребителя за продукта. При плани рането следва да се предвиди преглед и оценка на:

— плана за разработване на продукта;
— софтуерните стандарти;
— плана за управление на софтуерната конфигурация;
— плана за осигуряване на качеството;

— спецификацията на изискванията към продукта;
— спецификацията на изискванията към интерфейса.

2. **Общо проектиране**. Както известно, на този етап изискванията се конкретизират и уточняват, а така също потребителският възглед започва да се третира от гледна точка на реализацията. Съответни на това са и оценките на:
— всички ревизирани производствени планове;
— плана за тестването на софтуера;
— ръководството за оператора;
— ръководството за потребителя;
— ръководството за диагностика,

като последните 3 визират всъщност плановете за тези документи, които очевидно в този ранен стадий не могат да бъдат направени по същество.

3. **Подробно проектиране**. Резултатите от този етап са подробни спецификации, на основата на които може да бъде извършено програмирането (кодирането) на продукта. Тук се предвижда оценяване на:
— текущо ревизираните планове — отново;
— документа — изход от подробното проектиране;
— документа — проект на интерфейса;
— документа — проект на базата данни;
— тестовите примери за проверка на отделните модули;
— тестовите примери за проверка на интегрираните модули;
— описанието на процедурите по тестване;
— ръководството за програмиста;
— останалите ръководства в новото им състояние.

4. **Програмиране и тестване на отделните модули**. Както е известно, на този етап наред с програмирането самите изпълнители тестват своите реализации. През цялото време се извършват оценки на:
— плановете, доколкото търпят промени — както на всеки етап;
— написания първичен код;
— резултатите от тестването на отделните модули;
— всички ръководства.

5. **Интегриране и тестване**. Написаните и проверени отделни модули

започват да се интегрират, което изисква оценяване на:
— текущо ревизираните планове;
— резултатите от тестването на интегрирането;
— актуализирания първичен код;
— описанието на приемните тестове;
— всички ръководства.

6. **Тестване на крайния продукт**. Тук се оценяват:
— текущо ревизираните планове;
— всички ръководства;
— актуализираният първичен код;
— документът за описание и управление на версиите;
— докладът за тестване на крайния продукт (системата).

Типове оценки

Добре е известно, че софтуерът за военни цели изисква особена надеждност и други високи качества. Поради тези причини при създаването на такъв софтуер се полагат особено сериозни мерки за осигуряване на качеството, като специално се набляга на оценяването. Във връзка с това Министерството на отбраната на САЩ има приет стандарт — DOD-STD-2168 Software Quality Program. В него се фиксират типовете оценки, които следва да се правят при производството на всеки програмен продукт. По-важните от тях следват:

— съблюдаване на изисквания формат и стандарти за документацията;
— съответствие с изискванията на договора за разработка;
— вътрешна непротиворечивост;
— добра разбираемост;
— система за лесно намиране на пътя до всеки документ;
— съответствие на продукта с придружаващите го документи;
— коректно извършен анализ на изискванията, проектиране и програмиране (тук следва да се отбележи, че всъщност се прави оценка не на атрибути на
продукта, а на качества на процеса);
— правилно разпределение на ресурсите — по време и памет;
— адекватно проведено тестване за съответствие на продукта с изискванията;
— адекватно съставени тестови примери;
— пълнота на тестването;
— пълнота на регресивното тестване (този проблем беше разгледан по¬ горе в 8.2.2., т. В1 — тук се им предвид, че едни и същи тестове, проведени в начални стадии и довели до отстраняване на грешки, следва да се провеждат отново на по-късни етапи с цел избягване на грешки, резултат от корекции);
— съответствие и непротиворечивост между дефинициите на данните и тяхното използване.

Управление на конфигурацията

Управлението на софтуерната конфигурация обхваща методи и технологии за иницииране, оценяване и управление на измененията в софтуерния продукт след пускането му в експлоатация. Необходимите промени се правят не
само в първичния код, но така също в документацията — вътрешна (съпровождаща) и потребителска, в отчетите за тестване, в отчетите за откритите грешки. Следят се и се документират последователно създаваните версии и движението им сред различните потребители.

Общо взето, при по-малки проекти, реализирани от малки фирми, не се отделят специални усилия управлението на конфигурацията да се върши по един системен, добре формализиран и документиран начин. Това впрочем е
лесно обяснимо. Но ако един екип от 3 души е в състояние да помни историята на развитието на даден продукт наизуст или с малко неформални документи, то от известно място нататък (по отношение на мащаба на продукта и на колектива) такъв подход е невъзможен. Ето основните функции по управления на конфигурацията, които са решаващи за запазване качеството на продукта след влизането му в експлоатация:
— поддържане целостността на продукта;
— напълно контролирано управление на промените;
— управление и контрол на версиите;
— планиране на управлението на конфигурацията.

Отчитане на грешките

Добре известна истина е, че и в най-прецизно тествания софтуер след пускането му в експлоатация се откриват непрекъснато грешки. Все още сме много далече от възможността да се докаже математически строго липсата на грешки в дадена програма. За програми, писани на процедурни езици, има поне теоретическа яснота, как това би могло да стане, за обектно ориентираните — дори и
това не е ясно.

Поради тази причина подържането на ефективна система за регистриране на грешките с оглед последващото им отстраняване е абсолютна необходимост.

Такава система трябва да е действена в следните насоки:

1. **Идентификация на грешките**. Всяка идентифицирана грешка трябва да се опише ясно и точно. Може да се опише както поведението на продукта в дадената ситуация, така и реалният дефект в софтуера. Във всички случаи описанието трябва да бъде разбираемо както за хора, които не са се занимавали с

правенето на продукта, така и с оглед на това, че обикновено е минало вече време от разработването и дори лица, които са участвали пряко в работата, постепенно са загубили подробна представа за направеното. Впрочем така подготвената информация е необходима не само за конкретното отстраняване на грешката, но и за по-системни анализи на типовете грешки.

2. **Анализ на грешките**. Трябва да се документира сериозността на грешката и трудността на нейното отстраняване. Това ще позволи правилно заделяне на необходимия ресурс, определяне на приоритет и график за корекция. Обикновено грешките се откриват сравнително лесно, често те просто се набиват на очи, но тъй като отстраняването им невинаги е лесно, налага се да се вземат управленски решения, свързани с приоритета им. Впрочем отделянето във времето на анализа от корекцията обуславя разглеждането им като отделни и независими операции.

3. **Корекция на грешките**. Коригирането, освен че се извършва по същество, следва и да се документира. Описанието на корекцията трябва да съдържа:

— разказвателно описание на корекцията;
— списък на засегнатите модули от продукта;
— пълна идентификация според създадената система на засегнатите документи;
— евентуални промени в тестовите процедури в резултат на корекцията.

4. **Въвеждане на корекцията в експлоатация**. На практика не е възможно всяка направена корекция да отива незабавно при потребителя. Затова обикновено се прави работен екземпляр, в който постепенно се натрупват направени корекции. В определен момент, резултат на управленско решение, се създава нова модификация на продукта, включваща всички направени корекции (а така също подобрения и нови функционалности, ако е имало такива) и тя се предлага официално на потребителите. Задължително се регистрира в коя модификация (или версия) всяка корекция е включена.

5. **Регресивно тестване**. Вече беше казано, че всяка корекция крие потенциална опасност от създаване на нови грешки. Изследвания показват, че тази опасност е около 20%. Най-разумната възможност за отстраняването им е в момента на откриването им те да бъдат третирани като стандартни грешки и да бъдат отстранявани по вече описаната схема. Във всички случаи обаче описанието на регресивния тест трябва да включва:

— списъка на отново тестваните компоненти;
— върху коя версия/модификация е направено тестването;
— индикация, дали тестването е било успешно или неуспешно.

6. **Категоризация на грешките**. Анализът на грешките би се улеснил, ако те се систематизират с оглед бъдещи анализи. Близко до ума е, че категоризирането ще е най-успешно, ако се направи в момента на отстраняването на всяка грешка. Възможни признаци за категоризация са:

— тип на грешката — от изискванията, от проектирането, от програмирането, от тестването;
— приоритет на грешката — критична, некритична, козметична;
— честота на грешката — повтаряща се, неповтаряща се.

Анализ на тенденциите

Това е пасивна и по-второстепенна дейност, но тя може да насочи към полезни корективи. Състои се в анализиране на определени страни от софтуерния процес и изготвяне на съответни отчети.

1. **Количество на грешките**. Възможно е да се събират данни за количеството грешки както за целия период на разработването, така и за отделни кратки периоди. При това е

препоръчително грешките да се класифицират подходящо, например по функционални групи или по отговорни специалисти. При по-
продължително събиране на подобни данни (в рамките на повече проекти) възможно е след статистическа обработка да се фиксират някакви критични граници, които биха указали например доколко е смислено да се обучават повече,
или да се оставят на постовете им различните отговорници.

2. **Честота на грешките**. Честотата на грешките следва да се свързва с конкретна единица — тестова процедура, програмен модул, дял от спецификация. Също след статистически обработки могат да се правят изводи за латентни
грешки в тези единици.

3. **Сложност на програмните модули**. Известни са немалко метрики, които позволяват обективно измерване на сложността на програмните единици.При установяване на по-голяма от някаква критична стойност, може да се прис-
тъпи към опит за намаляване на сложността на тази единица.

4. **Честота на компилациите**. Въпреки че на пръв поглед тази характеристика изглежда тривиална, Де Марко е показал, че програмни модули, които са били компилирани често при създаването им, след това се компилират често
и при интеграцията и системното тестване. Така че би било оправдано за програмна единица, за която е установено, че при кодирането е била компилирана

Проследимост

Принципът за проследимостта означава, че за всяка единица, създадена по време на разработването на софтуера, трябва да може да се проследи от каква друга единица е получена. Така например за даден алгоритъм, въплътен в програма, следва да може да се види резултат на какво решение е по време на
проектирането или на създаването на изискванията. Благодарение на това се улесняват оценките и преди всичко отстраняването на грешките.

Планиране на ПОКС

Както всяка друга дейност при производството на софтуер програмата за осигуряване на качеството е също обект на грижливо предварително планиране. Практическият съвет в това отношение е да се ползва аналогичен план от предходна разработка. Най-малкото, с което може да бъде полезен подобен план, е, че няма да бъдат пропуснати задължителните елементи.

Социални фактори

Все по-често се обръща внимание на социалните елементи, начина на общуване на ръководителите с подчинените и с клиентите, мотивацията на разработчиците. Доколкото осигуряването на качеството е особено силно свързано
с общуването между членовете на екипа, често при противопоставящи роли е необходимо да се обърне внимание и на този елемент от софтуерния процес.

1. **Точност**. Абсолютната точност на представянето на данните от всяка оценка или друга дейност по осигуряването на качеството е задължителна. Практиката показва, че най-ефективната защита на контролираните срещу искания на контролиращите, например за извършване на промени, е позоваване на неточност на данните. При това обикновено се цитират прецеденти и доколкото почти винаги времето за подробно разглеждане и доказване на представените данни не е достатъчно, такова позоваване се оказва достатъчно. Единственият начин за противопоставяне е просто да не се създава нито един прецедент за неточност.

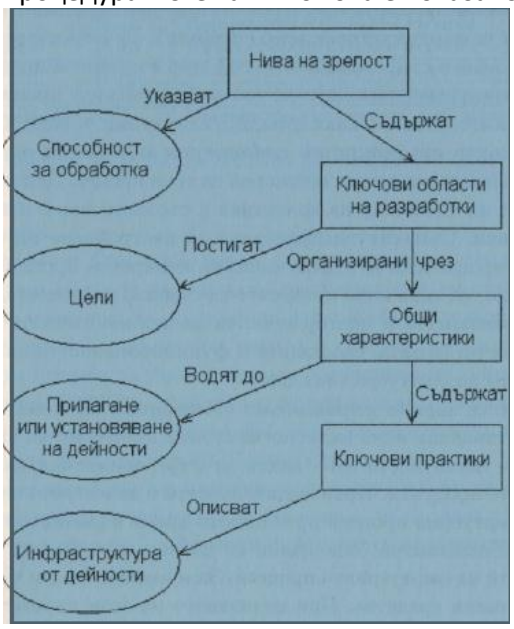
2. **Авторитет**. Обикновено отговорниците по осигуряване на качеството получават значителни пълномощия от ръководството на фирмата. Както и в други дейности обаче, знае се, че истинският авторитет е много по-малко резултат на
административни мерки, отколкото на доказвана непрестанно лична компетентност.

3. **Полза**. По принцип ефективната работа по осигуряване на качеството води до обща полза за всички участници в процеса на разработване. От друга страна обаче, в краткосрочен и индивидуален план много често това не е така,
защото на даден програмист например му се налага да прави корекции, които смята за ненужни и които му отнемат допълнително време и усилия. Още по-характерен и направо типичен пример са исканията към отделните изпълните-
ли, свързани с изготвяне на документация и спазване на стандарти. В този смисъл ръководителят и другите експерти по осигуряване на качеството имат за задача да мотивират засегнатите от мерките изпълнители, като ги убеждават в
общата полезност на такива мерки.
4. **Комуникации**. Както вече се каза, осигуряването на качеството в голяма степен като схема е натрупване и разпространяване на информация. Формите за това са обикновено речеви и писмени в различни модификации. Откъдето следва, че отговорните за осигуряване на качеството трябва да са комуникативни личности, умеещи добре да се изразяват говоримо и писмено.
5. **Постоянство**. Не е възможно да се запази доверието, ако концепциите и произлизащите от тях решения се променят често. В такива случаи скоро идва момент, от който нататък нарежданията просто не се изпълняват в очакване на последваща промяна.
6. **Отмъстителност**. Лице, отговарящо за осигуряване на качеството, нерядко ще се сблъсква с волни или неволни нарушения. Те биха му дали възможност при следващи случаи да се изкуши да се възползва от създалото се
предимство пред нарушителя. Ясно е, че такава злонамереност е вредна за общата атмосфера и за ефективността на работата във фирмата.

16.3Зрялост на софтуерните процеси

Методологията SEI CMM
16.1.1 Същност и предназначение Изграждането на такава организация на процесите, която да осигурява ефективното и навременно създаване на качествен софтуер, се нуждае и от непрекъснато обективно оценяване на достигнатото равнище. В края на 80-те се обявяват изследвания в тази насока, които по-късно се развиват, прилагат в практиката и оказват силно влияние върху теоретиците и практиците в цял свят. Основният продукт на тези изследвания е Моделът за оценяване на зрелостта - ***Capability maturity model - CMM*** [4]. Този модел е ***предназначен за:*** - подобряване на софтуерните процеси; - оценяване на софтуерните процеси, при което специално подготвени експерти определят текущото състояние на софтуерните процеси в организацията;- оценяване от подготвени експерти на способността на потенциални изпълнители на даден софтуерен проект. Оценяването се извършва по точно определена схема на основата на ***150 въпроса,*** отговорите на които са „***да***" или „не". Пример за такъв въпрос е: „Има ли във Вашата фирма формално организирана система за ос игуряване на качеството". ***Подобряването*** се извършва с помощта на точно и подробно определени и структурирани действия в планирането, технологията и управлението на разработването и съпровождането на софтуера. CMM е получил ***широко разпространение*** защото:**-** е основан на ***реалната практика;-***отразява ***най-добрите постижения*** на тази практика;**-** съобразява се с ***нуждите*** на участниците в софтуерния процес и неговото оценяване и подобряване; - *документиран* е добре;- *достъпен* е за широката публика.**16.1.2 Структура** CMM се състои от 5 *нива на зрялост (maturity levels).* Нивото на зрялост е добре дефинирана развиваща се платформа, насочена **към** достигането на зрял софтуерен процес. Всяко ниво на зрялост се състои от *ключови области на обработка (key process areas),* с изключение на ниво 1. Всяка ключова област съдържа група свързани дейности, които, изпълнени съвместно, водят до постигане на целите, считани за съществени за това ниво на зрялост. Например, една от ключовите области на

обработка на ниво 2 е „Планиране на софтуерния проект". Всяка ключова област се състои от 5 секции *общи характеристики (common features):*Ангажираност за изпълнение, Способност за изпълнение. Изпълнявани дейности, Измерване и анализ и Верификация на приложението. Тези общи характеристики показват дали прилагането и формалното установяване на ключовата област на обработка е ефективно, повторяемо и трайно. На следващото ниво са т.н. *ключови практики (key practices),* чрез които се постигат целите на ключовите области. Ключовите практики описват инфраструктурата и *дейностите,* които допринасят за ефективното прилагане и формалното установяване на съответната ключова област. Например, една от ключовите практики на ключовата област „Планиране на софтуерния проект" е „Планът за разработването на софтуерния проект се създава в съответствие с документирана процедура". Схематично това е показано на Фиг. 16.1



16.1.3 Нива на зрялост *Ниво 1* се нарича начално *(initial).* Организация на ниво 1 не осигурява стабилна среда за разработване и съпровождане на софтуер. В такива фирми управлението не е на здрава основа. Дори да има планирани процедури, в момента когато настъпи криза по отношение на срокове или ресурси, плановете се изоставят и се преминава изключително към прграмиране (кодиране) и тестване. Успехът зависи само от опитността и квалификацията на ръководителя и членовете на екипа. Общата способност на организациите на ниво 1 е непредсказуема, каквито са и елементи на софтуерния процес - графици, бюджет, качество на продукта и др. Поради това не може да се предскаже и ефективността на такава организация. Ниво 2 се нарича ***повторяемо (repeatable).*** В организацията е установена политика за управление на софтуерния проект и процедури за **пр**илагане на тази политика. Планирането и управлението на нови продукти е на основава на опита от предишни проекти. Показател за достигането на това ниво е формалното установяване на ефективни процедури за направление. Това позволява на организацията да повтаря успешните дейноости от проект на проект. Може обаче да се случи някои от дейностите (практиките) да не се повторят. По принцип, за да бъде един процес ефективе той трябва да е приложим, документиран, измерим с възможности за усъвършенстване и с обучени за приложението му кадри. Ръководството на всеки проект следи разходите и графиците. Дефинирани са стандарти със стремеж към съблюдаването им. ***Ниво 3*** се нарича ***определено (defined).*** Тук стандартният процес за разработване и съпровождане на софтуер в организацията е документиран, включително технологичните и управленските процеси, като при
това те се интегрирани. Така стандартизираният процес в рамките на CMM се нарича ***стандартен софтуерен процес*** на организацията. Налична е специална група, отговорна за този процес. Действа програма за обучение на членовете на колектива в съответствие с

изпълняваните от тях задачи. Съществува процедура за настройване на стандартния софтуерен процес към нуждите на всеки конкретен проект. Накратко, на това ниво процесите са стандартни и непротиворечиви, поради стабилността и повторяемостта на технологичните и управленските дейности. Цената, графиците и функционалността са под контрол и качеството на софтуера се следи. ***Ниво 4*** се нарича ***управляемо (managed).*** На това ниво организацията установява количествено измерими критерии за качество и се
стреми към постигането им - както за софтуерните процеси, така и за софтуерните продукти. Производителността и качеството се измерват за важните софтуерни процеси през цялото време в рамките на специална програма. Всеобхватна база данни се ползва за събиране и анализ на параметрите на софтуерните процеси. За измерването им са предвидени инструментални средства. При излизането на тези параметри от определени граници може да се установи дали това е случайно явление или са необходими коригиращи действия. При преминаването към нова област на приложение или нов инструментариум се пресмята и контролира рискът. Накратко, организациите на това ниво са предсказуеми, защото работят с измерими процеси и в дефинирани измерими граници. Предсказуеми са както високото качество на процесите и продуктите, така и тенденциите в развитието му. Нарушаването на определените количествени граници подлежи на коригиране. ***Ниво 5*** се нарича ***оптимизиращо (optimizing).*** На това ниво цялата организация е съсредоточена към непрекъснато подобряване на процесите. Тя има средствата за идентифициране силните и слабите страни на процесите, с цел предотвратяване на дефекти. Разполага се с данни за ефективността на софтуерния процес, които се използват за анализ на цената и ползата от въвеждането на нови технологии или изменения в него. Технологични иновации се идентифицират и евнтуално се въвеждат в практиката. Дефектите се анализират до установяване на причините и избягване на повтарянето им, както в текущия, така и на бъдещи проекти.

Ниво според CMM	Честота	Определение
1 = начално	75.0%	Примитивни и случайни процеси
2 = повторяемо	15.0%	Някои стандартизирани методи и контроли
3 = определено	8.0%	Добре структурирани методи, добри резултати
4 = управляемо	1.5%	Висока сложност с повторна използваемост
5 = оптимизиращо	0.5%	Съответства на теорията, нови методи

В таблицата на Фиг. 16.2, наред с кратките определения на петте нива, са дадени и данни за реалното разпределение на фирмите по тези нива на основата на извършени изследвания в САЩ. Както се вижда, фирмите на нива 4 и особено 5 са изключителна рядкост. Като пример за ниво 5 се дава организацията разработила софтуера на космическата совалка. Доста експерти смятат, че достигането на ниво 5 е свързано със значителни инвестиции, които не могат да се възстановят в кракосрочен план. Майкрософт и повечето японски „софтуерни фабрики" са на ниво 3.

16.1.4 Ключови области на обработка

По-горе беше казано какво представляват ключовите области на обработка. Тук ще изброим тези области, така както са разпределени по нива. ***Ниво 2:***Управление на софтуерната конфигурация; Осигуряване качеството на софтуера;Управление на договорите с подизпълнителите; Проследяване на софтуерния проект**;** Планиране на софтуерния проект**;** Управление на изискванията (на потребителя)**.** ***Ниво 3:*** Групови прегледи; Координация между групите;Технология на софтуерния продукт; Интеграция на управлението и технологиите; Програма за обучение; Дефиниране на софтуерния процес; Фокусиране върху организацията на процесите; ***Ниво 4:*** Управление на качеството на софтуера***;*** Управление на количественото оценяване на софтуерните процеси***.*** ***Ниво 5:***Управление промените в процесите***;***Управление промените в технологиите***;***Предотвратяване на дефектите***;*****16.1.5 Цели:***Целите обозначават обхвата, границите и намеренията за всяка ключова област. Те, както се каза, се реализират чрез ключови практики. Когато дадена ключова практика се настройва към конкретен проект, целта е тази, която служи като ориентир, доколко настройката е запазила същността на

практиката в рамките на областта.***Ключови практики*** Всяка ключова практика се изразява с едно изречение, често следвано от по-подробно описание. Характерно за тях е, че те не отговарят на въпроса как да се постигне дадена цел, а описват какво следва да се прави. *Пример* За илюстрация ще изберем *една от 18-те ключови области* и ще дадем в схематичен, съкратен, но достатъчно пълен вид нейното описание.*Ниво 3: Програма за обучение Предназначението* на Програмата за обучение е да развие умения и знания у членовете на колектива, така че те да изпълняват задачите си ефективно.Програмата за обучение включва преди всичко идентификация на нуждите от обучение и след това осигуряване на такова обучение, някои случаи обучението се извършва неформално (в течение на изпълняването на задачите, чрез ръководство от страна на по-опитни членове на екипа), в други то се организира формално във вид на курсове или ръководено самообучение.*Целите* на Програмата за обучение са:1. Да се планират дейностите по обучението. 2. Да се осигури обучение за уменията и знанията, необходими *за* изпълнение на управленските и технологичните роли. 3.Всяко лице, за което това е необходимо, да получи нужното му обучение *Ключовите практики* за осъществяване на Програмата за обучение са:1. Всеки софтуерен проект разработва и поддържа план за обучение, *в* който са указани нуждите от обучение. 2. Планът за обучение се разработва и актуализира в съответствие с документирана процедура. 3. Обучението се извършва в съответствие с плана за обучение. 4. Курсовете за обучение, подготвени в организацията, се разработват в съответствие с вътрешни стандарти.5. Съществува процедура, която се ползва и с чиято помощ се установява доколко дадено лице притежава или е придобило необходимите за неговата работа умения и знания. 6. Подържа се архив за протичането на обучението.

16.3 Стандарти за качеството на софтуера

В областта на качеството на софтуера има най-разнообразни стандарти, отличаващи се по степен на общност, предназначение, степен на готовност за приложение, разпространение и др.Някой от тях не визират точно качеството на софтуера а по-общи или други проблеми на неговото създаване.

17. Организационно управление

Основна техника за преодоляване на стихийността в създаването на ПП е мениджърският подход. В една компания извършваните дейности могат да се разделят в 2 групи: ритунни и еднократни (за създаването на нещо ново). Вторият вид се реализират чрез **проекти**. Всеки проект има цел, която трябва да постигне с крайни и ограничени ресурси. **Целта** на управлението на проекти е те да се осъществяват систематично и контролирано чрез постъпкови процедури. **Ефективни проекти** са тези, които водят до повишаване на производителността и осигуряване на качествени крайни продукти услуги. Резултатите от тях са постигнате в определения срок и с оптимално използване на ресурсите. Предпоставки за ефективни проекти са: създаване на благоприятна среда за работа; формулиране на ясни изисквания за работата; управляване на проектите чрез планиране и контрол; прилагане на ефективни процедури.

Управление на **човешките ресурси** са дейностите по подновяване и подбор на кадрите, повишаване на квалификацията, професионално развитие, обучение за работа в екип и др. Управлението на **продукта** е определяне на предназначението, основните функции и изисквания към новия продукт още вначалото на проекта. Управлението на **процеса на разработване** включва планиране и ефективно осъществяване на всички дейности. В зависимост от предназначението, обхвата и начина на организирането им, дейностите могат да бъдат основни (програмиране) или допълнителни (осигуряване на качеството). Избраният в софтуерната организация **модел** на ЖЦ определя осъществяването на основните дейности (общи характеристики). Глобалните дейности са едни и същи за всички проекти, т.е. могат да се

прилагат общоприети стандартни методики. Основната цел на управлението на **проекта** е да се държат нещата под контрол. Всеки проект има определена начална и крайна дата, цел и обхват, добре дефиниран краен резултат, ясни и точни критерии за завършване. Управлението на проекти е в съответствие с избрана методология -съвкупност от принципи, методи, процедури и правила изясняващи как се осъществява определен проект и какви са взаимоотношенията му с други обекти. Методологията е формална, ако е представена в писмен вид. Преимуществата на прилагане на формална методология са, че тя: подпомага избирането на проекти; позволява разпределение на ресурсите; контролирането на риска; подпомага счетоводно-финансовите дейности; подобрява качеството на създавания ПП продукт чрез задължително спазване на определени стандарти; дава възможност за сравнителен анализ на различни проекти; осигурява систематичност и управляемост на проектите.

Традиционната **управленческа структура** е йерархична, като нивата на йерархията, видът и броят на ръководителите на всяко ниво зависят от големината на софтуерната фирма. В общия случай се разграничават две нива: ръководители на фирмата (**senior managers**) и ръководители на проекти (**project managers**). **Ръководителите на фирмата** отговарят за: това кои проекти ще се осъществяват и при какви условия; сключване на договори за разработване; назначаване на ръководители на проекти; ръководство на основни фирмени дейности, подбор и развитие на кадрите и др. **Ръководителите на проекти** отговарят за: подготовката и съставянето на офертата за разработката; оценка на стойността и продължителността на проекта и необходимите за осъществяването му ресурси; сформиране на екип(и) за проекта и оценяване на работещите; съставяне на планове и графици за проекта; текущо управление на проекта; представяне на проекта пред външни или горестоящи инстанции. Разработени са специални процедури за подбор на ръководителите на проекти. Обикновено се предпочитат лица с информатично или бизнес образование с допълнителна специализация в областта на софтуерните технологии. Практическият опит в разработването на софтуер е съществено предимство. Професионалните изисквания се допълват от изисквания за лидерски качества, комуникационни умения и индивидуални характеристики, като способност за оценяване на нови идеи.

Планирането обхваща дейностите по разработване на планове и процедури за реализирането им, възлагане на задачи и проверка на изпълнението им. Планирането е основна управленческа функция. Ръководството на фирмата определя дали и какво да се планира. Основни възражения срещу планирането са, че отнема много време, че е скъпо, че се основава на предположения, които може и да не са верни, че не може да се използва рационално опитът от предишни проекти. В подкрепа на планирането се посочва, че то: подпомага осъществяването на целенасочени действия и резултати; служи за координация и контрол на всички дейности; увеличава ефективността, като предотвратява повторно извършване на определени действия; осигурява „прозрачност" на управлението на проектите. В зависимост от целите и предназначението си плановете се делят на **общи и конкретни**. **Общите планове** се отнасят до софтуерната организация. Делят се на целева програма, стратегически и тактически план. **Целевата програма** (The Mission) отговаря на въпроса защо съществува софтуерната организация. Най-често целта е разработване на софтуер, който да се реализира печалба. Но съществуват и софтуерни организации с идеална цел, които разработват софтуер за пуляризиране на нови научни постижения, създаване на freeware или shareware. Специални са целите на разработване на софтуер за специални отрасли с национална значимост.

Стратегическият план отговаря на въпроса „какво" да създава софтуерната организация. Този план се създава за относително по-дълъг период от време (около 5г.) и определя в какви приложни области ще бъдат разработваните ПП, дали и как ще се разширява дейността на организацията, какви ще са нейните основни и странични дейности. **Тактическият план** отговаря на въпроса „как" ще се осъществяват планираните софтуерни проекти (около 1г.).

Конкретните планове се отнасят до отделен софтуерен проект. Освен плана за проекта, се съставят бюджет, календарни планове и графици за групите, реализиращи проекта, както и

индивидуални планове за всеки участник. Съществен конкретен план е **бюджетът**. Съставянето на бюджета се подчинява на следните правила: съставя се от ръководителя на проекта със съдействието на финансист; представлява писмен документ, който се утвърждава от ръководството на фирмата и спазването му е задължително. Препоръчва се регламентирането на процедура за внасяне на изменения; бюджетът описва необходимите средства и начина на осигуряването им. (Разходите са описани по видове, наречени **„пера"**. Основни пера са: за трудови възнаграждения, за закупуване или наемане на техника, за поддържане на средата за разработване, за консумативи, за командировъчни и др. Обикновено не се допуска безконтролно прехвърляне от едно перо в друго); препоръчва се да се предвидят контролни точки от проекта, в които да се анализира изпълнението на бюджета и ако е необходимо, да се актуализира.

Под **риск** се разбира потенциален проблем, който може да възникне и да застраши успешното завършване на проекта. Всеки риск се характеризира с **неопределеност** и **вредност** - винаги има нежелани последици или загуби. В зависимост от това, какво засягат, рисковете се делят на **рискове на проекта**, **технически** или **бизнес-рискове**. **Очаквани** са рисковете, които могат да се идентифицират чрез анализ на проекта; **предвидими** са рисковете, които могат да се опишат въз основа на опита от предишни проекти и **непредвидими** са рисковете, които просто се случват. Основната идея на управлението на риска е да се анализират някои от най-вероятните рискови ситуации и да се направят опити за избягването им или за намаляване на последиците от тях. Управлението на риска се осъществява в три стъпки: идентифициране, оценяване и ограничаване на възможните последици. За всеки риск тези стъпки са последователни, но дейностите за управление на различните рискове се преплитат.

Идентифицирането на риска е систематичен опит да се установи какво може да застраши успешната реализация на проекта. Рисковите фактори могат да се разделят на две групи: общи за всички софтуерни проекти и специфични за даден проект. Основна техника за идентифициране на риска е съставянето на въпросници, свързани с характеристики на основните застрашени обекти (за обекта проект - обхват, размер и продължителност; за продукт - спецификации, сложност, уникалност; за персонал - организация, състав и мотивираност на работната група и др). За всеки риск се определя и степента му на влияние върху проекта. **Оценяване на риска** - данните за риска се систематизират и се преобразуват в информация за вземане на решения. За всеки риск се определя вероятността и последиците от риска, въз основа на които се определя и приоритетът му. **Предотвратяване на последиците от риска** изисква планиране на управлението на риска - определянето на рисковете с най-висок приоритет и свързаните с тях действия за предотвратяването им. По време на разработването се следят идентифицираните рискови фактори и се натрупва информация за всеки от тях, за да се установи дали и как се променя вероятността за настъпване на рисковата ситуация. Политиката на следене на риска поддържа увереността на ръководството, че всичко е под контрол и се увеличават шансовете за успешно завършване на проекта.

След изясняване на основните параметри на проекта работата по проекта трябва да се организира така, че той да се вмести в определения срок и ресурси. За тази цел се съставя и се следи изпълнението на календарен план-график, представящ разработването на ПП във времето. **Съставянето на графици** преминава през следните етапи: 1 . Декомпозиране на всички необходими дейности на множество от задачи. 2. Определяне на зависимостите между задачите. Някои задачи могат да се изпълняват само последователно, други - едновременно. 3. Определяне на трудоемкостта на всяка задача в избрана мерва единица и дефиниране на начална и крайна дата за извършването й. 4. Разпределяне на работата между членовете на групата така, че изпълняваните задачи да не надвишават отпуснатите за проекта ресурси. 5. Определяне на персонални отговорници за всяка задача. 6. Определяне на получаваните изходи от всяка задача. 7. Дефиниране на контролни точки, при достигането на които ще се анализират и оценяват получените междинни продукти.

Освен съставянето на графици, организирането на проекта включва и проверка дали те се спазват. **Проследяването** на проекти става чрез регламентиране на система за контрол.

Контролират се основните параметри на проекта. Текущото състояние се сравнява с планираното и при несъответствия се планират коригиращи действия. Основен принцип на контрола на проекта е, че се контролира работата. Контролът се основава на проверки за свършената работа по всяка задача и осъществяване на обратна връзка. Реализира се чрез последователност от проверки, чрез които се натрупват отчети, справки и др. Най-често използваните техники за събиране на данни са провеждане на интервюта или преглеждане на материали, свързани с реализацията на проекта. Данните се проверяват за правилност и непротиворечивост. Съществено е, че контролът трябва да обхваща стандартните дейности и ситуации, а не изключенията. **Нивата на контрол** са обикновено четири: организация, отдел, проект, работна група. За всяко ниво на контрол се определят съответни процедури. **Развитието** на проекта може да се следи чрез т.н. **„earned value analysis"**, който се извършва в определени контролни точки. Той се базира на следните мерки: **Бюджетна трудоемкост на задача** (Budgeted Cost of Work - BCW): Прогнозната трудоемкост на всяка задача; **Бюджетна трудоемкост на завършените до момента задачи** (Budgeted Cost of Work Scheduled - BCWS): Сумата от прогнозната трудоемкост на тези задачи, които според графика трябва да са завършени до момента; **Бюджетна трудоемкост на проекта** (Budgeted Cost at Completion-BAC): Сумата от всички BCWS и следователно, оценка за трудоемкостта на проекта като цяло; **Планирана стойност** (Planned Value - PV): Частта на всяка задача PV = BCW/BAC; **Бюджетна трудоемкост на извършената работа** (Budgeted Cost ofWork Performed - BCWP): Сумата на реалните трудоемкости на задачите, завършени до момента; **Реална трудоемкост на извършената работа** (Actual Cost of Work Performed - ACWP): Сумата на реалните трудоемкости на завършените задачи.
Могат да се пресмятат следните **индикатори за развитието на проекта**: Рентабилност (Earned Value - *EV = BCWP/BAC); Индекс за изпълнение на графика* (Schedule Performance Index *SPI = BCWP/BCWS); Вариране на графика* (Schedule Variance - *SV= BCWP-BCWS); Индекс за изпълнение (Cost Performance Index CPI = BCWP/ACWP); Вариране на трудоемкостта (Cost Variance CV = BCWP – ACWP).*
За подобряване на управлението на проекти въз основа на натрупания опит се прилага и т.н. „**постфактум**" **подход**. Същността му е да се изберат представители на разработчиците и потребителите, които да изразяват писмено мнението си по проблеми на управлението на проекта, процеса на разработване, чрез попълване на „заключителен отчет".
Важно е да се идентифицират практики, които са доказали полезността си за успешното управление на проекти. Ето и няколко **съвета към мениджър**: Управлявай риска; Оценявай разходите и графиците; Използвай метрики за целите на управлението; Помни, че хората са най-важният ресурс; Управлявай софтуерните конфигурации през целия ЖЦ на ПП; Проследявай изискванията; Проектирай два пъти, програмирай веднъж; Оценявай риска и ползите от прилагането на re-use подхода;Проверявай изискванията и проектите; Осъществявай тестването като непрекъснат процес; Компилирай и тествай често.
Правила за софтуерни проекти, спазването на които би повишило ефективността на реализираните проекти: комуникации, определяне на крайния срок, възлагане на проект, осигуряване на необходимата документация, задължения на изпълнителя, конфиденциалност, независимост (изпълнението на проекта да не се обвързва с трета страна), работна група, проблеми, отчети, плащания, прекратяване на договора, особени ситуации (обстоятелства, възпрепятстващи нормалното завършване на проекта).

Човешкият фактор

Ефективното управление на софтуерните проекти се основава на управлението на хората, продукта, процеса и проекта (т.н. четири P's - people, product, process and project). Наредбата не е случайна. Мениджърите в софтуерното производство са осъзнали, че подценяването на човешкия фактор може да застраши успешната реализация на всеки проект. Беше разработено и разширение на модела CMM -PM-CMM (People Management Capability Maturity Model),

предназначението на което е да подобри готовността на софтуерните организации, да осъществяват приложения с нарастваща сложност чрез привличане на талантите, необходими за подобряване на възможностите за разработване на софтуер.
Всяка софтуерна организация трябва да има кадрова политика. **Набирането на кадри** (recruitment) обхваща всички дейности, насочени към намиране и назначаване на най-подходящия кандидат за определена длъжност. Разглежданите кандидатури могат да бъдат вътрешни (internal recruitment) или външни (external recruitment). При ***вътрешното набиране*** след определяне на длъжностните характеристики за вакантните позиции, се разглеждат възможностите за преназначаване. Използваните техники могат да бъдат:- поддържане на информация за всички служители (worker pool repository);- професионални знания и умения, квалификация, опит от предишни софтуерни проекти и т.н.;- вътрешен конкурс. Свободните места се обявяват и всички служители, които се интересуват, могат да подават съответните документи;- назначаване на основа на препоръки (employee referalls). При ***външното набиране*** целта е да се потърсят и назначат нови за софтуерната фирма специалисти. Основни техники са:- използване на връзки със съответните университети и привлизаче на способни завършващи студенти;- използване на услугите на специализирани фирми за подбор на кадри; - примамване на висококвалифицирани специалисти (head-hunting), които не само не са безработни в момента, а обикновено имат успешна кариера в друга конкурентна организация. За избор на подходящи служители се предлага използването на т.н.„**психологически профили"**. Психологическият профил е съвкупност от индивидуални личностни качества, за които се счита, че са важни за съответната професия. Създадени са модели на профил за програмисти.При назначаване чрез тестове се изследва психологическият профил на всеки кандидат и се сравнява с този на най-добрите професионалисти във фирмата. Реалното използване на този подход изисква разработване на адекватни тестове, съобразени с националните, културни и професионални различия. Не се препоръчва изборът на софтуерни специалисти да се основава изцяло на психологическия профил. Друг подход е предложен от **Уейнбърг**. Той предлага изследването само на две характеристики, които счита за задължителни при този тип работа: адаптивност и устойчивост към стрес. Индивидуалната устойчивост към стрес, съчетана с целенасочените мениджърски действия за контролиране на стресовите фактори, би осигурила висока производителност на. създателите на софтуер.
Ще разгледаме някои основни принципи за сформирането на работните групи, като вземем предвид индивидуални, личностни и поведенчески характеристики. Идентифицирани са четири типа комуникативно поведение при изпълнение на групови дейности: **обособяващ се, водим, лидер и сътрудничащ**. Ефективна работа може да се постигне, като на всеки член се възлагат задачи, които са съобразени със стила на неговото поведение.
Разработването на софтуер се извършва в групи, чиито членове са обединени от обща социална дейност и се намират в непосредствено лично общуване, което е основа за възникване на емоционални отношения. От психологическа гледна точка са определени две нива на взаимоотношенията - **делови функционални контакти** и **междуличностни взаимоотношения**. Последните се основават на предпочитанията и личната привързаност между хората, субективните установки, чувството за симпатия и антипатия, доверие и подозрителност. Психическата съвместимост е необходимо условие за успешна работа. За цялостното функциониране на групата отговаря избиран или назначаван ръководител (**мениджър**). При определянето му се вземат предвид две групи условия - обективни (устойчиво централно положение в групата, лидерски тип комуникативна ориентация) и субективни – добре изразени и стабилни личностни качества. Могат да се разграничат четири вида управленчески стила:***- насочващ;- съветващ;- подкрепящ;- делегиращ***. Добрият мениджър може да използва различни стилове в отношенищта си с различни хора и в зависимост от конкретната ситуация. За да поддържа високо ниво на мотивираност, мениджърът трябва да се съобразява с фактори, определящи мотивацията на подчинените си (като работна среда, хората с които се работи, възнаграждения), и да се опитва да ги контролира

Предложено е така нареченото „**egoless programming**". Това е стил на разработване, при който всички софтуерни продукти (спецификации, проекти, програми, документация и др.) се разглеждат като собственост на групата, независимо от кой конкретен неин член са създавани. Те могат да се изследват обективно и да се променят, без да се засяга нечие самочувствие. Този подход преодолява „**когнитивния дисонанс**", който се изразява в склонността на всеки човек да приписва заслугите за успехите на себе си, а причините за провалите да търси у другите. Колективната отговорност за разработването и резултатите от него подобряват значително параметрите на проекта.

Комуникациите между членовете на групата трябва да се регламентират така, че да се осигури ефективността им. Формите и интензивността на контактите зависят от:- числеността на групата (за група от *N* члена възможните контакти са *N(N*~ 1)/2);- структурата на групата;- състава на групата. Регламентираните формални контакти могат да бъдат централизирани (чрез координатор) или свободни (на всеки с всеки).

Сбирките са форма на комуникация, която поддържа колективния стил. Целта им може да бъде анализиране на текущото състояние на проекта, информиране на участниците за настъпили изменения и т.н. Няма правила за броя, типа и продължителността на провежданите сбирки, но са формулирани някои препоръки:а) ***планиране на сбирката*** Преди всичко трябва да се, определи необходимо ли е провеждането на сбирката и тя да се свиква само ако проблемите не могат да се решат чрез индивидуални срещи;**б) провеждане на *сбирката*** За ефективното протичане се определя водещ, който:- открива и ръководи сбирката; в) ***закриване на сбирката*** Сбирката се прекратява, когато свърши запланираното време, когато се изчерпи дневният ред или когато групата няма ресурси да продължи. Задължително е резюмиране на постигнатите резултати и взети решения.

Съгласно CMMI-модела (integrated product and process development) систематичното и управляемо разработване на качествен софтуер трябва да се осъществява от екипи (integrated teams), включващи специалисти с допълващи се знания, умения и опит, които си сътрудничат през целия ЖЦ на ПП за по-добро удовлетворяване на потребителските нужди. Следващата стъпка е **team-building** - създаване на „екипност" чрез целенасочени усилия за подобряване на ефективността на работа. **Обучението** на екипите трябва да се провежда от професионалисти, които да изготвят специална за конкретната организация програма и да я осъществят чрез подходящи техники(outdoor adventure training, group dynamic training).

Основната цел на **ергономиката** е да изследва и да предложи подходяща работна среда, така че човешките ресурси да се използват по най-ефективен начин. Изследванията са показали, че организацията на работното място влияе в най-голяма степен на производителността. Работните места могат да бъдат в общо помещение или в индивидуални стаи за всеки разработчик. Двата вида организация на работното пространство са с различни възможности за, контрол на работата и за комуникации между участниците в работните групи. Освен характеристиките на физическата среда, от значение са и наличните комуникационни, хардуерни и софтуерни средства, предоставени на всеки участник в софтуерната разработка.

Споменатият вече **PM-CMM** модел описва прагматичен подход, целта на който е подобряване на цялостното функциониране на софтуерна организация чрез повишаване на производителността на най-важния управляем ресурс - хората.

Липсата на **правилно етично поведение** има огромно влияние върху развитието на организацията. Повтарящите се случаи на неволно или умишлено некоректно поведение, в резултат на което са се оскъпили или провалили проекти, създават лош имидж на софтуерната фирма и застрашават нормалното й функциониране и дори съществуване. Едно възможно решение е в софтуерната организация да се обсъждат проблемите от морално-етично естество и да се приемат правила за

етично поведение. Организации като ACM, IEEE, IFIP и British Computer Society имат собствени етични кодекси.