14. Софтуерно инженерство и неговото място като дял от знанието. Софтуерен процес и модели на софтуерни процеси. Концепция за многократна употреба. (УСИ)

Анотация: Изложението по въпроса трябва да включва следните по-съществени елементи:
1. Софтуерно инженерство – какво е софтуер, видове софтуер, същност и обхват на софтуерното инженерство.
2. Софтуерен процес – фази и основни дейности, видове модели и езици за моделиране, шаблони за описание.
3. Сравнителен анализ на описателни модели на софтуерен процес – модел на водопада, прототипен модел, модел на бързата разработка, спираловиден модел и др.
4. Концепцията за многократна употреба.

1. Софтуерното инженерство – какво е софтуер, видове софтуер, същност и обхват на софтуерното инженерство.

Какво е софтуер
Софтуерът представлява компютърни програми (инструкции), които при изпълнение осигуряват желан резултат, който е свързан с удовлетворяване на характеристики, функционалности, производителност, сигурност и други. В софтуера се включват и структури от данни, които дават възможност на програмите адекватно и ефективно да съхраняват и обработват информацията, с която работят. Документите, които описват работата и използването на програмите също са част от софтуера.

Софтуерът е логическа единица, която се разработва, а не се произвежда. Той не се износва и не се амортизира в резултат на външни явления. Въпреки това може да подлежи на остаряване, тъй като иновациите в тази сфера са често срещано явление в наши дни. Тази тема със „срока на годност“ на софтуера е философска. Някой алгоритми, на които се базира даден софтуер е доказано, че не е възможно да се подобрят от гледна точка на броя операции, които извършват, но в същото време, технологиите, които ги изпълняват може да се подобрят.

Софтуерът, който е предназначен да се продава на клиент или потребител, се нарича продукт. Различават се два типа софтуерни продукти:
- общи (generic) – самостоятелни продукти/системи, които се продават на пазара и са предназначени за всеки потребител, който иска да си ги закупи;
- поръчани (customized) – продукти/системи, които са възложени от отделен клиент и се разработват специално за неговите нужди.

Изискванията към първия тип продукти се определят от самата организация, която ги разработва и в повечето случай са породени от или създават някаква нужда на пазара, а във втория – от клиента поръчител.

Видове софтуер
а) **Системен** софтуер – съвкупност от програми, предназначени да обслужват други програми. Характеризират се с интензивна комуникация с хардуера и разрешават синхронизационни проблеми при работа с множество процеси. Пример за такъв софтуер са операционните системи, драйверите за устройства, компилатори и други.
б) **Приложен** софтуер – програми, които решават специфична нужда на бизнеса или потребителите. Това са например, текстови редактори или редактори на изображения, медиини плейъри и други.
в) **Научен** софтуер – облсужва нуждите на отделните дялове на науката. Използва се за симулации на системи, за емпирично тестване, анализи, автоматизирано проектиране и други.
г) **Вграден (embedded)** софтуер – вгражда се в различни от класическия компютър хардуерни устройства – телефони, прахосмукачки, перални, микровълнови фурни, автомобили, климатици и много други. Може да е насочен както към потребителски, така и към бизнес нужди.
д) **Продуктова линия** – съвкупност от софтуерни системи, които притежават общи черти и задоволяват определени нужди. Ключовото тук е използването на общи основни компоненти, което намалява цената на разработката.

е) **Уеб приложения** – в най-простата си форма са множество свързани файлове хипертекст. Съвременните уеб приложения обикновено са многослойни и могат да се използват от множество потребители едновременно.
ж) **Изкуствен интелект** – използва нечислови алгоритми за решаване на сложни проблеми, които не подлежат на директен анализ. Това са например приложения за разпознаване на образи, звук или подчерк. Невронни мрежи, линейна регресия и други.
з) **Наследен (legacy)** софтуер – софтуер, който е разработен преди няколко десетилетия и продължава да се използва като непрекъснато е бил модифициран, за да задоволи промени в бизнес изисквания, регулации или платформи (хардуерни и софтуерни). Ключовите характеристики на наследения софтуер са дълъг живот и критична важност за бизнеса.

Същност и обхват на софтуерното инженерство
Дефиниция (IEEE). Софтуерното инженерство е прилагането на систематични, дисциплинирани и количествено измерими подходи към разработването, функционирането и съпровождането (поддръжката) на софтуера. Изследване на тези подходи.

Софтуерното инженерство е дисциплина, която се занимава с всички аспекти на проектирането и разработването на висококачествен софтуер. Софтуерните инженери трябва да прилагат систематичен и организиран подход в работата си и да използват подходящи инструменти и техники, зависещи от решавания проблем, ограниченията върху разработката и наличните ресурси.

Софтуерното инженерство се занимава с разбирането и анализирането на даден проблем. Проблемът или задачата се декомпозира на подпроблеми, на всеки от които се търси решение поотделно. След това се конструира решение на цялостния проблем от отделните подрешения.

Докато компютърните науки се занимават с теория и фундаментални знания, софтуерното инженерство се занимава с практическите въпроси за разработване и доставяне на полезен софтуер. Теориите на КН не са достатъчни, за да служат като изчерпателна основа за СИ, необходими са платформи и методологии, който да позволят лесното им прилагане.

Софтуерното инженерство е дисциплина, която обхваща интегрирането на процеси, методи и средства за разработване на софтуер.

Софтуерното инжинерство включва най-общо 5 аспекта:
- Исторически – как се появява дисциплината;
- Икономически – цели се разработката на най-икономически ефективното решение;
- Поддръжка на готовите решения;
- Изисквания, анализ, дизайн – подходи за намаляване на грешките и препядствията при разработка;
- Екипна разработка – техники за управление и организиране на екипите.

2. Софтуерен процес – фази и основни дейности, видове модели и езици за моделиране, шаблони за описание.

Софтуерният процес е:
- Последователност от предвидими стъпки – план, който следваме, за да създадем продукт или система в срок и с високо качество;
- Рамка на задачите, които е необходимо да се изпълнят, за да се изгради висококачествен софтуер;
- Процесът е повече от процедура – процесът е съвкупност от процедури, организирани така, че да се изграждат продукти, задоволяващи определени цели и стандарти.

Етапи при разработването на софтуер:
а) Анализиране и дефиниране на изискванията;
б) Проектиране на системата;

в) Проектиране на програмата;
г) Писане на програмата;
д) Тестване на единиците (Unit Testing);
е) Интеграционно тестване (Integration Testing);
ж) Тестване на системата;
з) Доставяне на системата;
и) Поддръжка.

Основни дейности – малък брой, които са приложими за всички софтуерни проекти:
- **Комуникация** – интензивно взаимодействие и сътрудничество с клиента. Събиране и разбиране на изисквания за функционалността и на ограниченията върху работата и разработката на продукта;
- **Планиране** – създава се план за бъдещата работа по разработка на софтуера. Описват се рисковете, необходимите ресурси, работните продукти, които ще се произведат и времевия график;
- **Моделиране** – включва две действия:
  - **Анализ**, в резултат на което се създават като работни продукти модел на анализа и спецификация на изискванията;
  - **Проектиране/дизайн** – определя се дизайна на данните, архитектурата, интерфейса и на ниво компоненти. Работните продукти са модел на дизайна и спецификация на дизайна.
- **Конструиране** – включва генериране на код (ръчно или автоматично) и последователното тестване – на самостоятелни компоненти, на модулите, на подсистемите и на цялата система, а също и потребителско тестване (бета-тестване);
- **Внедряване** – софтуерът се предоставя на клиента и той го оценява.

Модел на софтуерен процес и видове модели
Моделът е опростено описание на начина на разработване на софтуера, представено от определена гледна точка. Той е абстракция на действителния процес.

Видове модели спрямо гледната точка:
а) **(Workflow model) Модел на потока от дейности** – описва последователност от дейности, които се извършват в процеса на разработка. Отделните дейноти се описват със своите входни и изходни параметри;
б) **(Dataflow model) Модел на потока от данни** – проследява преобразуването на данните и при какви дейности става това;
в) **(Role/action model) Модел на роля/действие** – описва ролите в екипа и дейностите, за които всяка роля отговаря.

Софтуерните модели на процеси могат да са **описателни** – описват как се разработва софтуерна система или **предписателни** – предписват как би трябвало да се разработи нова софтуерна система.

Езици за моделиране – видове
Езиците за моделиране се използват за детайлно и изчерпателно представяне на:
- дейностите, които трябва да се извършат;
- ролите на хората;
- структурата и същността на артефактите, които се създават и/или поддържат;
- средствата, които се използват.

В зависимост от парадигмата се разделят на дескриптивни, мрежово базирани, императивни и хибридни, които комбинират елементи от други видове.

Спрямо организацията си, езиците се разделят на:
- entity-relation – същности и връзки между тях;
- role-interaction – роли и техните взаимоотношения;
- object-oriented – основната единица е обект, който има данни и функционалност.

В зависимост от формалността си, езиците се разделят на:

– Формални – представя се чрез формален синтаксис и семантика, има ясно дефинирани правила за употребата им и списък на конструкциите им;
– Полуформални – обикновено имат графично представяне с формален синтаксис, но не и формална семантика;
– Неформални – няма строго дефинирани правила.

Шаблони
Шаблон – описание на общо решение на общ проблем или въпрос, на базата на което може да се извлече детайлно решение на специфичен проблем.

Шаблон на процес – шаблон, който описва доказан, успешен подход или последователност от действия за разработване на софтуер. Описва какво трябва да се направи, а не точни детайли как трябва да се направи. Шаблоните могат да бъдат дефинирани на различни нива на абстракция.

Според нивото си на абстракция са:
– Task Process Patterns – описва отделни стъпки за извършване на действие или задача, която е част от процеса;
– Stage Process Patterns – дефинира стъпките, които се извършват итеративно в рамките на една базова дейност за процеса;
– Phase Process Patterns – дефинира последователността и взаимодействието между базови дейности, които се случват в рамките на процеса.

Всеки шаблон трябва да съдържа:
– Начален контекст;
– Проблем;
– Решение;
– Краен контекст;
– Свързани шаблони;
– Начини на употреба.

3. Сравнителен анализ на описателни модели на софтуерен процес – модел на водопада, прототипен модел, модел на бързата разработка, спираловиден модел и други.

Описателни модели на софтуерен процес
Водопад – най-старият метод на структурирано разработване на софтуер. Предлага систематизиран и последователен подход. Моделът налага по-скоро структура на управление на проекта, отколкото дава насоки как да се извършат отделните дейности.
Всяка дейност трябва да бъде напълно завършена (след одобряване на множество документи) преди да премине към следващата и след това не трябва да се връщаме към тази дейност отново.

Модел на бързата разработка (RAD – Rapid App Development) – високоскоростна адаптация на модела на водопада. Целта му е кратък цикъл на разработка (60-90 дни). Следват се основните дейности както при модела на водопада с разликата, че при „планиране“ се разделя проектът на модули, така че различни екипи да моделират и конструират различните модули паралелно. При конструирането се разчита на предварително съществуващи софтуерни компоненти и автоматично генериране на код. Създадените отделни модули се интегрират заедно при „внедряване“.

Еволюционни модели:
– **Инкрементален/постъпков** – не доставя системата като едно цяло, а на стъпки като всяка стъпка доставя само част от цялата функционалност. Следва основните дейности на водопада, но ги прилага за всяка стъпка отделно;
– **Итеративен** – още в началото доставя цялостната система, макар и част от функционалността да е в примитивна форма. При всяка следваща итерация не се добавя нова функционалност, а само се усъвършенства съществуваща.

Първите стъпки от разработката може да служат като прототип, за да помогнат за извличане и изясняване на изискванията.

Прототипен модел
Еволюционен прототип – представя базов вариант на системата, който после се доразвива до крайния вариант на системата.
Изхвърлен прототип – създава се прототип, предлагащ ограничено подмножество от функциите на крайната система, който се представя на клиента с цел определяне и валидиране на изискванията.

Спираловиден модел – еволюционен модел на софтуерен процес, който съчетава прототипния модел и модела на водопада, като се поставя акцент на анализ на риска. Моделът е скъп за прилагане.

Проблеми, особености и рискове
Водопад – трудно е за потребителя да формулира всички изисквания в началото. Клиентът трябва да е търпелив, тъй като не получава обратна връзка как се разработва системата, преди тя да е завършена.

RAD – опасно е да се прилага когато се разчита на все още нови и неусвоени технологии, защото това ще затрудни преизползването на код.

Еволюционни технологии – уменията за комуникация и координация са от особено голямо значение при разработката. Необходимостта от активно участие на клиентите може да доведе до закъснения.

Прототипен – може да използва значителни ресурси, а да не успее да удовлетвори очакванията. Прототипът може да доведе до лошо проектирана система, ако самият той стане част от крайния продукт. Моделът не е подходящ за разработване на софтуерни системи, където проблемът е добре разбран и интерфейсът е ясен и прост.

Спираловиден – може да се окаже трудно да се убедят клиентите, че процесът на разработка е контролируем, а не е безкраен цикъл. Изисква се участието на разработчици с компетентност за оценка на рисковете.

За какви проекти са подходящи?
Водопад – когато изискванията са осъзнати и ясно формулирани още в началото и когато са ясно дефинирани ролите на всеки. Може да се приложи и за по-големи проекти, но при които времето и бюджета не са критични.

RAD – проекти, при които могат ясно да се обособят отделни модули. Обикновено се прилага за малки проекти. При големи проекти и множество модули е необходимо голямо количество човешки ресурс, за да се завърши проекта на време.

Еволюционни модели – за проекти, при които трябва да се достави бързо системата и клиентът да започне да използва част от функционалностите ѝ още преди тя да е завършена. Прилагат се когато няма достатъчно човешки ресурс за цялостната реализация в определен срок или когато с итерациите може да се управляват технологичните рискове.

Прототипен – проекти, при които не са достатъчно ясни потребителските изисквания и дизайнът на софтуерната система.

Спираловиден – прилага се за големи проекти. Благодарение на итерациите е гъвкав и промени в изискванията не са толкова голям проблем както при модела на водопада например.

Моделите могат и често да се прилагат в комбинация с други модели.

4. Концепция за многократна употреба

Опитните софтуерни инженери проектират системите така, че да могат да се преизползват изцяло или частично вече съществуващи системи. Няма нужда да се преоткриват вече съществуващи решения. Сред ползите от многократна употреба са:
- спестяване на време и ресурси при разработка;
- по-ефективно използване на човешкия труд;
- надеждност;
- намаляване на риска от провал на проекта;
- съобразяване със стандарти;
- предвидимост по отношение на качеството на софтуера и процеса на разработка.

Сред недостатъците са:
- увеличена цена за поддръжка;
- споделена отговорност;
- трудност при намирането на подходящи компоненти.

Съществува баланс между многократната употреба (reusability) и приложимостта (usability) – колкото по-обобщен е даден интерфейс, толкова по-вероятно да се използва многократно, но и е толкова по-сложен и труден за употреба.

Съществуват два концептуални подхода за многократна употреба на софтуер:
- Базиран на съществуващи програмни единици:
  - COTS продукти;
  - компонентно базирано разработване;
  - аспектно-ориентирано разработване;
  - архитектура, ориентирана към услуги.
- Базиран на създаване на програмни единици:
  - шаблони на проектирането;
  - програмна генерация.

COTS – Commercial Off-The-Shelf
COTS системите обиконвени са цели приложения, които предлагат API (Application Programming Interface – приложно програмен интерфейс). Приложими са в широк кръг от области.

Ключовото тук е да се намери този COTS продукт, който предлага най-подходящата функционалност и да се реши въпроса с комуникацията между модулите – всеки модул има собствен формат и структура на данните. Също така, обикновено чрез COTS продуктите се получава повече функционалност от реално необходимата и понякога трябва да се добави обвивен wrapper модул, който да предотврати достъпа до ненужната функционалност.

Проблеми:
- понякога интеграцията на COTS продукти е доста труден процес;
- липса на контрол върху равитието и появата на нови версии на продуктите;
- недостатъчна поддръжка от производителите на COTS продуктите.

Компонентно-базирана разработка
Компонентите са по-общо понятие от обектите – един компонент може да е изграден от един или няколко обекта. Чрез компонентите се постига по-високо ниво на абстракция.

Софтуерният компонент е градивна единица на софтуерните системи, вътрешната структура и процесите, на която не са видими за другите компоненти. Компонентите могат да осъществяват връзка помежду си единствено чрез входно-изходни точки, наречени интерфейси.

Различаваме:
- Входен (requires) интерфейс – дефинира какво трябва да се предостави на компонента, така че той да се изпълни според спецификацията;

– Изходен (provides) интерфейс – дефинира услугата (и резултата от нея), която компонентът предоставя, в следствие на изпълнението му при условие, че са удовлетворени изискванията на входния интерфейс.

Компонентите са независими самостоятелно изпълними единици. Те предоставят една и съща услуга, без значение използваната платформа и програмен език.

COTS продуктите и компонентите са две много близки понятия. Може да се каже, че COTS е частен случай на компонентно-базираното софтуерно инженерство.

Процес на разработка при компонентно-базирано софтуерно инженерство (КБСИ):

Тук има някои разлики, които се дължат на необходимостта от търсене на компоненти за интегриране в системата – т.нар. Evaluation phase. Компонентите, между които се избира са 3 вида – комерсиални, с отворен код (COTS) и налични от предишни проекти. Препоръките са да не се променя кода на компонента, дори да има достъп до него.

Аспектно-ориентирано разработване
Засяга и цели да постигне т.нар. разделяне на функциите (separation of concerns). Припокриващата се функционалност се разработва като аспекти, които динамично се вмъкват в програмата.

Шаблони на проектирането (design patterns)
Описва се дадена задача и най-съществените моменти от решението ѝ. Шаблоните са абстрактни и целят да могат да се прилагат в множество различни ситуации, без да зависят от конкретен програмен език. Най-често стъпват на свойствата на класовете наследяване и полиморфизъм.

В зависимост от предназначението си, различаваме следните видове шаблони:
– Създаващи (creational) – използват се за създаване на обекти;
– Структурни (structural) – използват се за улесняване на интеграцията на класове/обекти;
– Поведенчески (behavioural) – предоставят методи за комуникация между обектите.

Програмна генерация
Програмните генератори използват вече дефинирани шаблони и алгоритми. Те са вградени в генератора и може да се параметризират. Програмната генерация е възможна, когато може да се дефинира съответствие между понятията от приложната област и програмния код. Използва се специфичен (за проблемната област) език. Програмната генерация е ефективен метод за разработка, но приложим в ограничен набор от приложни области.