01. Софтуерни технологии- основни понятия

0. Въведение

Предмет на разглеждане е това, което на английски език се нарича Software Engineering вече над четири десетилетия.

Преводът на български не може да бъде буквален, защото "инженерство" и Engineering смислово не се покриват.

Немското название - Software Entwicklung или пък френското - Genie Logiciel ни насочват, но не ни дават крайното решение.

Най-различни съображения през 1985 г. ни доведоха до българския термин Софтуерни технологии.

Първият у нас курс по софтуерни технологии беше подготвен и четен от Аврам Ескенази и Владимир Занев в продължение на 3 години, започвайки от 1984/85 учебна година, за студентите от IV и V курс на Факултета по математика и информатика на Софийския Университет.

От 1987/88 уч. година след известна преработка той стана задължителен за студентите там и в четенето на лекциите и упражненията се включиха Нели Манева и Валя Петрова.

Междувременно, най-напред в Икономическия университет - Варна (с лектор Георги Зеленков), по-късно в Пловдивския университет, International University, Нов български университет и на други места започнаха и продължават лекции по този предмет, макар и не винаги под същото име, а понякога и в рамките на повече от един курс.

1. Програми и програмни продукти

1.1. Хронология

Ще навлезем в тематиката и в проблема програми-програмни продукти (прилики и разлики) хронологично.

Заедно с построяването на първите компютри от средата на 40-те години се появяват и първите програми за тях.

Предназначението им е било за военни цели.

Доколкото в разработката и на хардуера, и на софтуера са участвали учени, естествено е било да се направят опити за използване на компютрите за подпомагане на научните изследвания, преди всичко за изчисления.

Още преди края на 50-те години, банките установяват, че дейност им би могла да бъде направена по-точна и по-бърза, ако бъдат приложени компютри.

Така и става и до днес банките си остават едни от най-сериозните консуматори на компютърни ресурси.

За първа внедрена софтуерна реализация се счита тази, разработена през 1951 година за нуждите на известната английска фирма за чай, сладоледи и сладкиши J. Lyons.

Любопитно е, че за целта фирмата си разработва и нужния хардуер (LEO) на основата на известната тогава електронна машина EDSAC, създадена в Кембриджския университет.

Този софтуер е имал практически всички възможности на аналогичния днес – транзакции, склад, отчети.

Нещо повече – учудващо за тогавашното ниво, имало е дори възможности за анализ на резултатите, за съставяне на варианти на бюджет и за изготвяне на прогнози.

Постепенно компютрите навлизат в търговската дейност, в медицината, в дейността на авиолиниите и в много други области.

Всяка от тях налага създаването на съответните програми.

Компютрите обаче си остават "големи" (main frame) или се произвеждат с известни вариации нагоре (свръхкомпютри) или надолу (мини).

Във всички случаи потребителите са професионалисти, или добре обучени специалисти в дадена област на приложение.

В края на 60-те години е извършена една решаваща за по-нататъшното развитие на софтуерното производство стъпка.

Известната фирма IBM, по това време най-големият производител на хардуер и софтуер в света, в края на 1968 въвежда т.н. политика на unbundling.

Това означава, че софтуерът, който дотогава се е давал като безплатна добавка към хардуера, започва постепенно да се продава.

В началото това се е отнасяло само до част от доставяния софтуер, постепенно обаче кръгът му се разширява.

Какви са били стратегическите помисли на IBM с този техен ход не е докрай ясно, но едно е безспорно - той всъщност дава тласък за развитие на софтуерната промишленост.

1.2. Проблеми

От така скицираното развитие на програмите за компютри за последните около 60 години, се виждат няколко най-общи техни характеристики:

- всяка програма би трябвало да може да бъде поправяна, разширявана, подобрявана от своя автор или от друго квалифицирано лице; това е станало очевидно веднага при експлоатацията на първата компютърна програма;
- първоначално не е било много ясно дали други хора, освен автора, ще ползват написаната програма; днес разбира се, този въпрос е немислим - може по-често да се случи обратното - след написването авторът никога да не ползва своето творение;

- последното обаче веднага води до необходимостта от преносимост на програмата от един компютър на друг;
- след като отдавна програмите са станали обект на производствена и търговска дейност, ясно е, че трябва да има начин те да бъдат оценявани, преди да бъдат продадени.

От тези характеристики непосредствено следва, че програмата трябва да може да бъде записвана на някакъв технически носител и да бъде придружавана от определени документи, които я описват в различни аспекти.

Не трябва да се мисли, че днес, при възможностите за предаване на информация чрез Интернет, съхраняването на технически носител е излишно.

1.3. Определения

След тези подготвителни бележки сме готови да дадем определение за програма и за програмен продукт.

***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 продукт** е програма или съвкупност от взаимодействащи програми, записани върху технически носител и придружени от съответна документация.*

Възниква въпросът за често използваната дума софтуер.

Ще считаме термините софтуерен продукт и програмен продукт за еквивалентни по смисъл.

Думата *софтуер* се употребява за обозначаване на *съвкупност от взаимодействащи програми* и в този смисъл се доближава донякъде до понятието програмен продукт.

Друг термин, който се доближава още повече до понятието програмен продукт, е формулиран през 1983 от професионалната американска асоциация на електронните инженери IEEE и гласи, че *софтуерът - това са компютърни програми, процедури, правила и евентуално придружаваща документация, както и данни, отнасящи се до функционирането на компютърната система*.

Днес, най-голяма гражданственост е придобило разбирането за *софтуера* като *общо понятие за програми и/или програмни продукти*.

2. Характеристики на софтуера

Ще разгледаме няколко най-общи характерни особености на софтуера.

Редът на разглеждане не е свързан с тяхната значимост.

2.1. Необходими ресурси

Софтуерните продукти, в сравнение с много други, се отличават значително по отношение на влаганите в тях ресурси, от където следва и цената им.

Знае се, че един нов автомобил може да струва примерно от 5 000 до 200 000 US$, а ако се разгледат 95% от продаваните автомобили, този интервал ще се свие още много силно.

За сравнение ще посочим софтуерни продукти около двата края на скалата.

По отношение на използвания ресурс от време и хора (а оттам и цена) на горния край се намират софтуерните проекти, свързани с американската космическа програма - Аполо, Скайлаб, Совалката.

Те са изисквали труда на 700 души в продължение на 7 години.

Може да се оцени, че това е струвало над милиард долара.

Още по-скъп е проект на данъчната администрация на САЩ – 4 милиарда US$ през 1997.

За сравнение можем да посочим проста (но вършеща работа) информационна система за персонален компютър, която би могла да се разработи от един опитен програмист за един ден и следователно да струва примерно 100 долара.

2.2. 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.

Разбира се, написаните команди на първичния текст на една програма могат да бъдат видени, но логическата същност на програмата, особено ако е по-голяма, няма как да бъде "видяна".

Още по-малко - цялостната структура на комплекс от програми.

Това е смисълът на твърдението, че софтуерът е на много високо ниво на абстрактност.

2.3. Уникалност на софтуерното производство

Софтуерното производство наистина има черти, които го правят уникално.

Да се върнем към сравнението с автомобилното производство:

- **основните усилия при производството на програмен продукт са преди появата на първото работещо копие; обратно, производството на всеки екземпляр автомобил (без да забравяме значителните ресурси, необходими за проектирането на дадения модел и настройката на производствените линии) изисква влагането на немалко труд, енергия, материали;**

- ако има **грешка в даден програмен продукт** (а безгрешен софтуер, както е известно, няма), то тя **се отнася до всички негови копия**; при автомобилите този случай е твърде рядък или поне не е типичен - там дефектите могат да бъде разнообразни, но общо взето са **индивидуални - за всеки конкретен автомобил**;
- когато дойде време програмният продукт да **излезе от употреба**, това става относително едновременно и поради една и съща причина - обикновено идва нова версия, с някои възможности повече и съобразена с променения хардуер; при **автомобилите и причините за "умирането" на даден екземпляр са индивидуални** (катастрофа, повреди, възможности на собственика и пр.) и времето на живот се отличава от екземпляр към екземпляр значително.

2.4. Мултидисциплинарност на разработването на софтуер

Отделните програмни продукти са изключително **разнообразни по предназначение**.

Изключваме **специфичните инструментални средства** (компилатори, свързващи редактори, средства за тестване на програми), **операционните системи** и още малък брой типове софтуер.

Те се разработват **изключително от софтуеристи**.

Във всички останали случаи **не е възможно** програмният продукт да бъде създаден **само от програмисти**.

Когато се прави **счетоводен софтуер**, трябва задължително да **участва експерт по счетоводство**.

Ако се разработва **статистически пакет**, не може да се мине без **математици-статистици**.

При повечето програмни продукти трябва да се впрегнат заедно усилията на много повече групи специалисти.

При такова сътрудничество възникват проблеми на общуването.

Те са породени не толкова от личните особености на участниците, колкото от различния им тип на мислене и различните професионални езици, на които те говорят.

В последните години в комплекса курсове по софтуерни технологии често има курс по проблемите на общуването в рамките на колектива, разработващ програмен продукт.

При това специфични проблеми възникват, както пред ръководителите на различни нива на разработващия екип, така и между редовите изпълнители.

2.5. Специфични проблеми на надеждността

Известен на всички факт е, че абсолютно безпогрешен софтуер няма.

По изследвания на Американския национален институт по стандартите (NIST) от 2002 година загубите на американската икономика от грешки в софтуер са от порядъка на 60 милиарда US$ годишно.

Knight Capital Group Inc. е отчела загуба от $389.9M за 3. тримесечие на 2012.

Причината е софтуерна грешка на 01.08.2012, предизвикала погрешни нареждания за закупуване на акции в рамките на 30 минути.

Това е довело в крайна сметка до загуба от $457.6M.

През 2005 година от затвор в Мичиган били предсрочно освободени 23 затворници. Грешката била между 39 и 161 дни и се дължала на дефект в софтуера.

Едновременно с това необявен брой затворници били задържани след срока им за освобождаване.

Малка програма с линеен характер и еднообразни входни данни, вероятно би могла да бъде проверена докрай.

Типичният програмен продукт обаче никога не може да бъде абсолютно гарантиран срещу грешки.

Сред най-съществените причини са:

- прекалено много възможни пътища през програмата,
- големи допустими (и недопустими) съвкупности от данни,
- непредвидими действия от страна на потребителя.

Следователно, дори да разполага с много време, пари и други ресурси, разработчикът не е в състояние да докаже абсолютна липса на грешки в своя програмен продукт.

Съществуват чисто теоретически методи, гарантиращи пълна безпогрешност на програмата, но те са засега приложими само към прекалено тривиални програми.

Има разработени и технологии за тестване и отстраняване на грешки.

Те обаче поне от теоретическа гледна точка не могат да гарантират пълна липса на грешки.

А за иначе много добрата в други отношения обектно-ориентирана технология, все още няма и теоретически възможности за доказване на безпогрешност на програмите.

Отдавна са станали анекдотични примерите за програмата, която работила безпогрешно х години и изведнъж сгрешила (което е напълно възможно).

Или за онази 0 вместо 1 в програмата, управляваща космическа ракета, довела до преждевременно прекратяване на полета.

Ежедневно сме свидетели на програмни продукти, разработени и продавани в милиони екземпляри от световни фирми, в които изскача някоя и друга грешка.

От друга страна обаче, ако за дадена програма се докаже по един или друг начин, че не прави грешки, поне при определени условия, ясно е, че нито едно копие на тази програма никога няма да направи грешка при тези условия.

2.6. Рискове

Производството на софтуер в много повече случаи, отколкото това става в по-стари и утвърдени производства, може да доведе до по-малка или по-голяма загуба.

Не са малко случаите, когато потребители съдят производителя на поръчания от тях софтуер и успяват да получат огромни суми за неизпълнени изисквания от доставения програмен продукт.

Едромащабен пример е цитираната по-горе свръх система на данъчната администрация на САЩ.

Друг обичаен рисков фактор е срокът на доставка.

Все още не е възможно да се планира напълно сигурно крайният срок на доставка на разработван програмен продукт.

Причина са - най-общо казано - **огромният брой определящи фактори**.

За част от тях **няма обективни измерители**, а се разчита на **експертното мнение** на опитни специалисти и ръководители.

Не случайно **опитът** тук е от решаващо значение.

Доста отдавна е добре изследван и описан т.н. **"90% синдром"** - неопитните софтуеристи (ръководители, проектанти, програмисти) планират и работят (несъзнателно) така, че обикновено **90%** от програмния продукт **се завършва за определен** (обикновено точно определен) **период от време**, а за **останалите 10%** се оказва, че е необходимо в най-добрия случай **още толкова време**.

2.7. Софтуерът: средство, а не цел

Софтуерът е функция от трети ред.

Това означава, че **софтуерът** е средство, което **задейства** (управлява) **някаква система**, а **тя** самата **довежда до искания резултат**.

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**.

Смисълът на тази малко абстрактна конструкция е в това софтуеристите да помнят винаги, че това, което създават, е всъщност само средство за постигане на някаква цел.

От доста време вече има трудове, посветени на психологията на т.н. "софтуерни фанатици".

Техните приоритети често противоречат на здравия разум.

Задача на софтуерните ръководители е да внушават на изпълнителите правилните приоритети по подходящ начин.

Модерните софтуерни технологии обръщат специално внимание на този проблем.

2.8. Мащаби на производителността

Ирландия 2004:

- **24 000** в софтуерната индустрия - в международни и местни фирми (в последните - 11500),
- **общ износ – 19** млрд US$,
- следователно 1 лице "произвежда" годишно софтуер за износ за около **800 000** US$.

Индия 2004:

- **570 000** в софтуерната индустрия (най-голямата фирма – Infosys - има 35000),
- **общ износ – 12** млрд US$,
- следователно 1 лице "произвежда" годишно софтуер за около **21 000** US$.

3. Софтуерни технологии

3.1. Терминология

Терминът “Software engineering” се е появил за първи път на конференция на НАТО през 1969 година (Наур, Рандел – редактори на сборника, Бауер – председател на ПК).

Както вече беше казано, за най-подходящ еквивалент на български считаме “Софтуерни технологии”.

Споменатата конференция е била проведена с цел обсъждане на проблема със “софтуерната криза”.

Няколко години преди това са се появили компютрите от т.н. трето поколение.

Тяхната мощност, превъзхождаща на порядък тази на компютрите от второ поколение, е изисквала програмни продукти, непознати дотогава по своя мащаб и сложност.

Оказало се е, че в този период не е било ясно как да се произведе такъв софтуер.

Така възниква естествената необходимост от специална дисциплина, която да се опита да помогне за преодоляването на софтуерната криза.

3.2. Определения

Известни са немалък брой определения на “Софтуерни технологии”. Ето някои от тях:

- **Наур, 1969: *установяването и използването на здрави принципи за разработване с цел по икономичен начин да се произведе софтуер, който е надежден и функционира ефективно върху реална машина.***

- **IEEE (асоциацията на американските електронни инженери):** ***софтуерните технологии са систематичен подход към разработването, експлоатирането, съпровождането и изваждането от експлоатация на софтуера***
- **Феърли, 1984:** ***технологична и мениджърска дисциплина, занимаваща се систематично с производството и съпровождането на софтуерните продукти, които се разработват на време и на основата на точно определени разходи.***

Може да се смята, че второто от тези определения най-много се приближава до съвременното разбиране за същността на софтуерните технологии.

Едно необходимо допълнение е прилагателното "качествен" към "софтуер".

Второто е във връзка с огромното разпространение на търговска основа на програмни продукти и е свързано с техния маркетинг.

От друга страна изваждането от експлоатация по важност и по съдържание далече отстъпва на другите елементи на дефиницията.

Следователно, определението, което ще ползваме оттук нататък, гласи:

Софтуерните технологии са систематичен дисциплиниран и измерим подход към разработването на качествен софтуер, както и към неговото предлагане на пазара, експлоатация и съпровождане.

3.3. Цели

При поставеното условие за качество на разработвания софтуер, следва да идентифицираме основните му атрибути.

- **Всеки програмен продукт (ПП) следва да може да бъде лесно съпровождан.**

Това означава, че 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 ПП да бъде добре документиран.

- **Софтуерът трябва да бъде надежден.**

Това означава, че грешките в него трябва да сведени до възможния минимум, а продуктът да изпълнява очакваните от потребителя функции.

- **Програмният продукт трябва да бъде ефективен.**

Това не означава използване на възможностите на хардуера до границите на допустимото - най-малкото защото такъв софтуер обикновено се съпровожда много трудно.

Все пак потребителят очаква оптимално в известен смисъл експлоатиране на ресурсите на компютъра, преди всичко по отношение на икономичното използване на оперативната памет и достигането на голяма бързина на изпълнение.

- **Всеки софтуерен продукт трябва да предоставя удобен и лесен за усвояване потребителски интерфейс.**

Това става все по-актуално, поради драстичното нарастване на броя на потребителите и разширяването на спектъра им.

Не са малко случаите, когато поради недобре проектиран или реализиран интерфейс, много от възможностите на продукта остават неизползвани от повечето потребители.

Това от своя страна директно води до намаляване на броя на продажбите му.

- Особено място сред тези атрибути на качеството заема цената му.

Тук става въпрос за минимизиране разходите по разработването, но също и особено на тези по 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 това, че цената на подобренията в много случаи не расте линейно, т.е. малки подобрения в определен атрибут могат да изискват значително нарастване на вложените ресурси, следователно и на цената.

Всъщност, погледнато най-общо, **задачата на софтуерните технологии** 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.

Продължава да е разпространено обаче мнението, че **софтуерната криза все още не е преодоляна**.

Налице са значителни постижения, подобряващи методите и технологиите на разработване на софтуер или довели до създаването на мощни инструментални средства.

Но изглежда че **нуждите от софтуер нарастват по-бързо, отколкото се подобрява производителността на разработчиците на софтуер**.

3.4. Наука и практика

Както в много други области, и в полето на софтуерните технологии **практиците не винаги са склонни да следват незабавно теоретиците**.

Това е **обяснимо**, а и "**здравословно**".

Не всяка теория се оказва толкова **полезна**, колкото е изглеждала на пръв поглед, а понякога се оказва дори, че не е вярна.

От друга страна **пренастройването** към един нов метод, език или технология **изисква** от крайния му потребител (ръководител, проектант, програмист) **усилия и време за усвояването** му.

Много поучително в това отношение е едно **изследване**, извършено преди време в САЩ.

Анкетираните професионалисти-практици са отговорили на въпроси, свързани с приложението на нови методи, предлагани им от науката за софтуерните технологии.

Формулирали са отговора си съгласно следната петстепенна скала:

- използвам активно
- изучавам задълбочено
- следя литературата
- очаквам без да следя
- не проявявам никакъв интерес.

Тук са особено интересни установените от проучването най-малко прилагани технологии и методи.

За повечето от тях преобладаващият отговор е най-ниската степен - "не проявявам никакъв интерес".

Най-отблъскваната методика се оказва прилагането на метрики и различни техники на измерване.

Обяснението е, че сред теоретиците не съществува единно становище относно това кои метрики кога да се прилагат.

Също така, че липсват за повечето теоретично разработени метрики съответни програмни средства, които ги реализират.

Но най-вече – оценяването с обективни средства е потенциално опасно.

Това, разбира се, не пречи големи фирми (Hewlett-Packard, IBM) да ги прилагат масово и задължително.

Особено учудващо е, че практиците не са имали голям интерес и към обектно-ориентираните (ОО) методи.

Тук **обяснението** се търси в това, че:

- за проектантите и програмистите от **по-възрастното поколение** тази парадигма е необичайна и **трудна** за усвояване,
- немалка част от тях се занимават със **съпровождане на софтуер**, **създаван преди доста време** с тогавашните средства и трудно подлежащ на преработване с ОО методи,
- независимо от неоспоримите им предимства, по отношение на **валидирането на програмите**, ОО методите засега изостават от класическите структурни методи.

Свързан с този е и въпросът **докъде** всъщност **е науката в създаването на софтуерни продукти**.

Ясно е, че ако трябва например да се натоварят повечко **пакети в багажника на кола**, **никой не решава въпроса математически** (въпреки елегантните методи, които математиката предлага за целта).

Всеки прави няколко **проби** и решава проблема.

В много случаи софтуеристите не могат без формални методи или пък 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.

4. Източници

1. Fox J. M., Software and its Development. Prentice-Hall, Inc., Englewood Cliffs, 1982.
2. Ескенази А., Н. Манева, Софтуерни технологии. КЛМН, София, 2006.
3. Sommerville I., Software Engineering. Addison Wesley Publ. Company, 9. edition, 2011.
4. Glass R.L., Formal Methods vs. Heuristics: Clarifying a Controversy, The Journal of Systems and Software, 15(1991), No 2, p. 103-105.
5. Glass R.L., "Who Cares?" Technologies in Practice, The Journal of Systems and Software, 41(1998), No 1, p. 1-2.
6. Cusumano M.A., Software in Ireland, Comm. of ACM, Oct. 2005/Vol.48, No.10, p. 25-27.