Нов български университет,
деп. Информационни технологии,
/OOOK025/ Информатика.

Реферат на тема

Сравнителен анализ върху моделите на софтуерни процеси за разработване на софтуер.

Преподавател: Доц. Д-р Петя Асенова **Изготвил: Александър Пилафов /F91104/**

Съдържание

Успехът в разработката на даден софтуер зависи от използваните модели на софтуерни процеси. Те са ключови фактори в цялостният процес на работа върху един софтуер. Състоят се от няколко различни етапа: събиране на изисквания, проектиране, разработка, тестване и внедряване. Приносът на този реферат е изчерпателно изследване на широко използвани модели на процеси за разработване на софтуер, като се разглежда и подробен анализ на всички тези пет модела.

Увод

Качеството на разработката на софтуер води до ефективен модел на софтуерен процес, известен също като „жизнен цикъл“ на софтуерната разработка. Има различни модели за различни видове софтуерни процеси. Всеки модел описва разнообразието от действия, които са извършени по време на процеса. „Моделите на софтуерни процеси“ се използват за разработване на проекти в зависимост от естеството на проекта. Разработваните модели на софтуерни процеси през десетилетията се използват за подобряване на качеството на разработване на софтуер. Моделът на процеса на разработка на софтуер е скелет предложен за изработване на софтуерен продукт. Софтуерните процеси на разработка се състоят от различни дейности, които биват следните.

- Спецификация и анализ на изискванията
- Избиране на софтуерна архитектура
- Проектиране на софтуера
- Разработване на софтуера
- Тестване
- Имплементация
- Документация и обучение към софтуера
- Софтуерна поддръжка

Екипът за разработка на софтуер взема предвид целите и задачите на проекта при избирането на модела на софтуерния процес по, който да върви целият проект. Има различни видове модели и организации на процеса на разработка на софтуер приема се за най-подходящ модел, този който опростява „процеса на разработване” и увеличава продуктивността на разработчиците. Това изследване обсъжда сравнението между пет „модели на процеси“.

- Каскаден модел
- Еволюционно разработване
- Итеративно разработване
- Компонентно-базиран софтуерен инженеринг
- Гъвкави методи

„Жизнен цикъл“ на софтуерната разработка

Фиг.1 „жизнен цикъл“ на софтуерната разработка

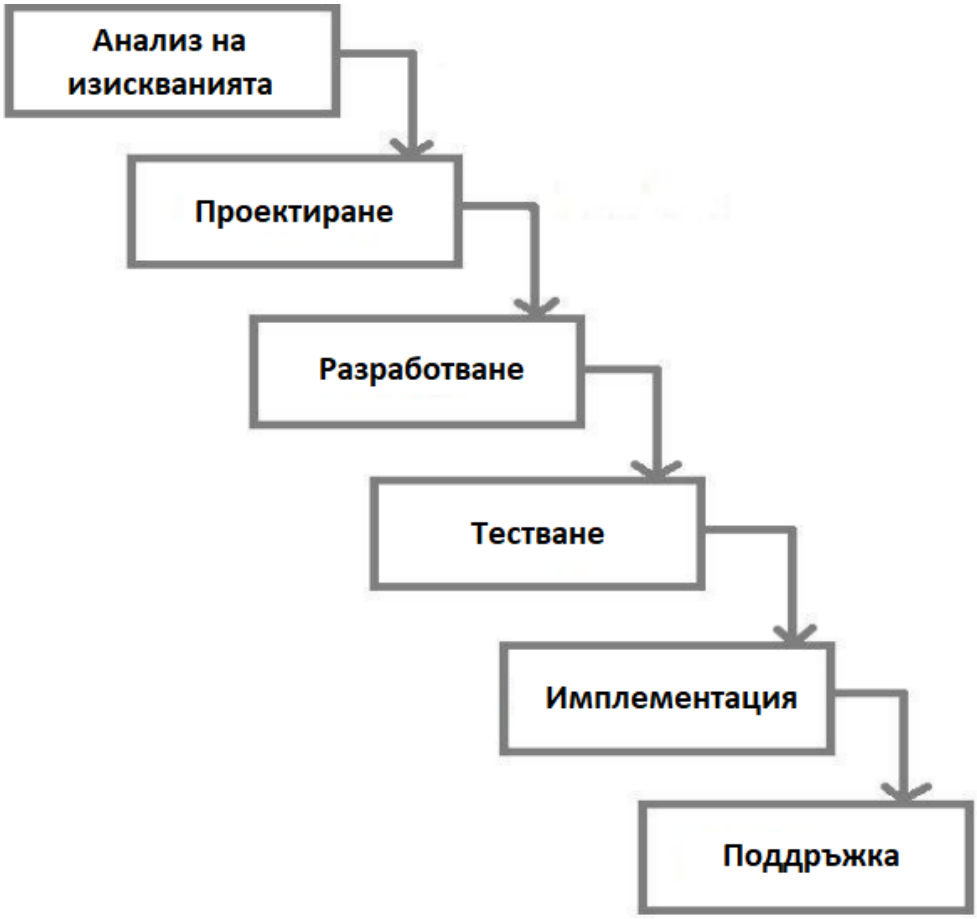
Фази на жизненият цикъл:

- Спецификация и анализ на изискванията
- Дизайн и проектиране на системата
- Разработване (кодене)
- Тестване
- Имплементация (реализация)
- Поддръжка на софтуера

„Модел на софтуерен процес“ е абстрактно представяне на процес, който се използва за разработване на софтуера и следва фазите на жизненият цикъл на софтуерната разработка, който включва проектиране, внедряване, тестване, и поддръжка.

Каскаден модел

Каскадният модел е традиционен модел, известен още като първоначален или класически модел за разработка на софтуер. Този модел описва методологията на разработка, като е линейни и последователни фази на проектиране. Поради последвателното протичане на фазите в каскадният модел, той не може да се върне към предишния си етап. След завършвайки фаза на разработка, процесът преминава към следваща стъпка и не може да се върне към предишния си етап. Каскадният модел има различни цели за всяка фаза от неговият модел. Той се използва често за правителствени проекти и е най-вече приложим в случай на малки проекти, където няма шанс за промяна в изискванията. Планирането и спецификацията при каскадният модел са много важни. Те помагат за правилното реализиране на целият проект и гарантират минимизирането на недостатъците в проектирането, преди да започнете да разработвате софтуера. Моделът започва с анализ и спецификация на изискванията на системата и продължава към избора на архитектурен дизайн, след което кодиране, тестване, внедряване и поддръжка.



Фиг.2 Каскаден модел

Основни принципи при използването на Каскаден модел на разработване

- Проектът се разделя на последователни фази
- Изключителна важност има фазата на спецификация и анализ на изискванията. Тя е основополагаща при използването на този модел на разработка.
- Дефинирани са стриктни времеви диапазони и крайни срокове за приключване на отделните части от проекта.
- Изисква се одобрение на потребителя и ръководството при завършване на всяка фаза, преди да се започне нова.

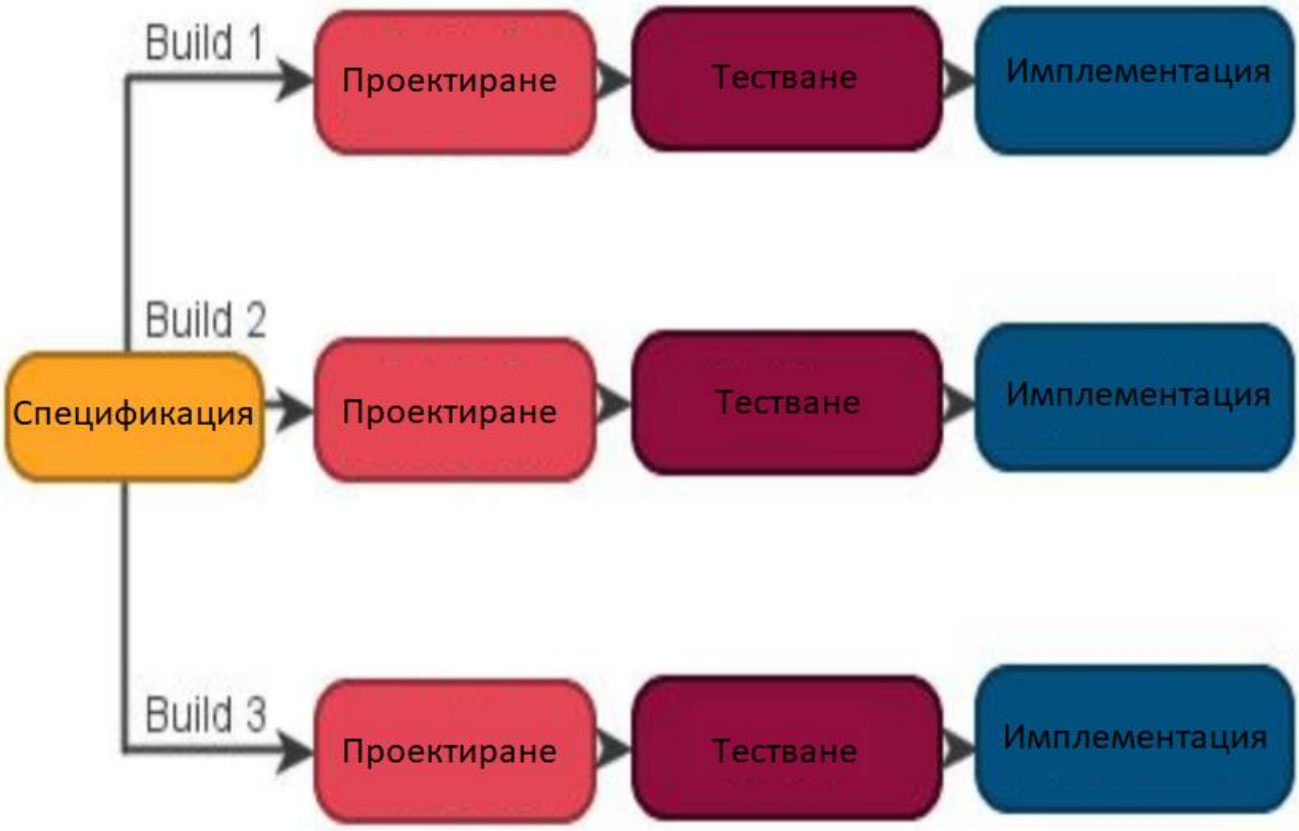
Предимства на Каскадният модел

- Лесен за имплементиране и разбиране
- Всяка стъпка има добре дефинирани цени
- Работи добре при вече започнати проекти и не изисква силни програмисти
- Действа като шаблон, в който методи за анализ могат да се направят за фазите на проектиране, кодиране и поддръжка

Недостатъци на Каскадният модел

- Изискванията не се променят след като фазата на проектиране е завършена
- Има голям риск при разработване на проект с заложени грешки в спецификацията
- Ако клиентът не е точен и ясен във фазата на спецификация, е трудно да се приеме каквато и да е промяна на всеки следващ етап
- Друг съществен недостатък на каскадния модел е, че на клиентът не може да се предостави прототип на продукта докато крайният етап на разработване не е завършен.

Итеративен модел (Модел на постоянно подобряване)
Итеративният дизайн е метод в жизнения цикъл на разработката на софтуер, който се фокусира върху разбиването на разработката на софтуер на малки парченца. В итеративния дизайн характеристиките са разработени с повтарящи се цикли. Процесът започва с итерации и преминава с просто изпълнение на малки части от „софтуерните изисквания” и постепенно се създават версиите на софтуера до окончателната версия. За разлика от каскадният модел, в началото, не е необходимо изискванията да са пълни. Разработването започва с внедряване на малка част от софтуера и продължава до края или до окончателното завършване на софтуера. Докато работим с итеративния модел, създаваме груб продукт(прототип). Тогава продуктът започва да се проверява и обновява в следващата итерация и същият процес продължава до завършването на продукта. Целта на итеративната работа е да позволи по-голяма гъвкавост за промени. Докато работи итеративно, екипът преминава през итеративен цикъл, в който във всяка итерация се тества и оценява версията, като се правят необходимите промени за доставяне на продукт, който следва спецификацията на клиента.



Фиг.3 Итеративен модел

Основни принципи

- Продуктът е разделен на малки итерации
- Изискванията на потребителите се управляват вместо задачите.
- Изискванията се основават на use cases и нефункционални изисквания
- Управляване на бизнес целите в зависимост от бюджета и сроковете за изпълнение
- Започва се с просто изпълнение на подраздел на изисквания, който да демонстрира ключовият аспект на системата. [прототип]

Предимства на итеративният модел

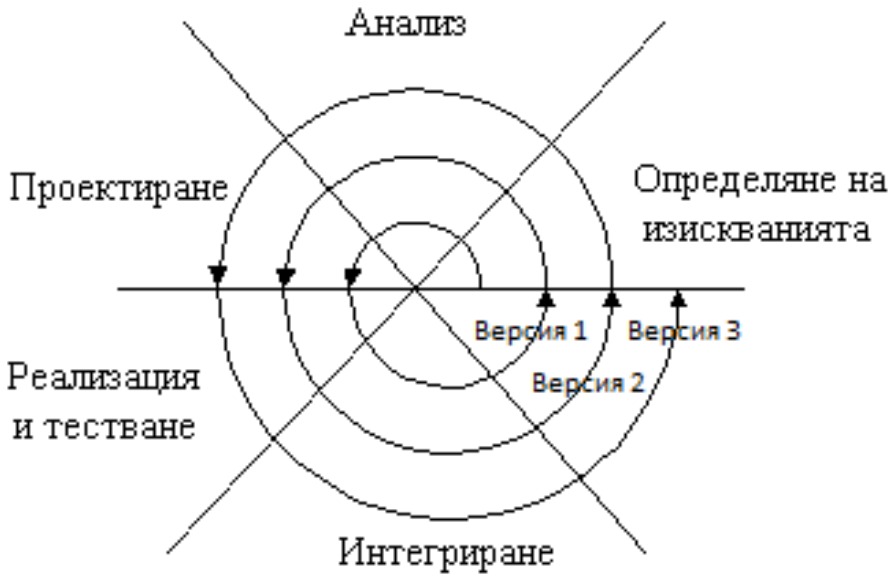
- Напредъкът може да бъде измерен
- Някои работни функции могат да бъдат разработени бързо
- Промените са лесни за приемане и са по-евтини
- Разработването на различни итерации може да бъде паралелно
- Тестването и отстраняването на грешки е по-лесно и се прави на всяка итерация
- Позволява обратната връзка върху различните итерации [версии]

Недостатъци на итеративния модел

- Няма ясна цел за итерациите
- Може да са необходими повече ресурси
- Комплексно управление в екипа за управление на итерации
- Проблемът със системата и дизайна може да се увеличи поради непълноценност при спецификацията и анализа на изискванията

Спираловиден модел

Спираловидният модел действа като итеративния модел, но при него има по-голямо значение рискът при разработването на софтуера. Този модел е по-подходящ за мащабни проекти. Има специфични дейности/фази в една итерация (спирала), където е резултатът е компонент от цялостният софтуер. В дейностите на спираловидният модел са „Планиране, анализиране, Инженеринг и оценка". Често софтуерен проект преминава през тези фази в итерации, наречени „Спирали" през цялата продължителност на разработката на софтуера.



Фиг.4 Спираловиден модел

Основни принципи на спираловидният модел

- Фокусът е върху оценката на риска и минимизирането на такъв в проекта.
- Проектът е разделен на по-малки части като същевременно дава по-голяма гъвкавост за промяна по време на процеса на разработване.
- Позволяващ оценка на риска и създаване на рисков анализ на проекта, който продължава през целия му жизнен цикъл на развитие.
- Всяко завъртане около спиралата преминава през дейностите на спираловидния модел.

Предимства на спираловидния модел

- Анализ на риска.
- Оценка на бюджета, разходите и планирането.
- Осигурява се работещ продукт на ранен етап на софтуерна разработка.
- Изискванията подлежат на промяна във всеки етап.

Недостатъци на спиралния модел

- Има нужда от експерти, които да анализират рисковите фактори.
- Този модел може да бъде скъп.
- Съществува риск от неспазване на графика или бюджета.
- Успехът на проекта зависи от рисковия фактор.

Дейност	Извършвани дейности	Резултат
Планиране	Събиране на изискванията	Лист с финализирани изисквания
Анализ на риска	Изискванията се преразглеждат и група от специалисти правят пълен анализ на риска и потенциалните рискови точки	Създават се документи, които описват всички потенциални рискови точки и процедури за предотвратяването им
Инжинеринг	През тази фаза се извършва проектиране и тестване на софтуера.	- Проектира се кода - Създават се тестови случаи - Създава се резултат от тестовете и репорт върху тях
Оценка на итерацията	Клиентът разглежда софтуера и преценява дали той изпълва изискванията. След което се преценява дали е приет или отхвърлен.	Характеристиките, които са изпълнени биват документирани.

Фиг.5 Дейности при Спираловиден модел

Модел на прототипизиране

Моделът на прототипизиране се отнася до разработването на прототип на софтуерно приложение, което показва характеристиките на софтуера на ниско ниво. В прототипният модел изискванията не са замразени, преди да започването на етапът на проектиране, разработване и тестване на софтуера. Разработката на софтуера започва дефиниране на условия. Чрез този модел клиентът може да добие представа за системата, която ще бъде доставена в края. Помага при получаване на ценна обратна връзка от клиента, която помага на екип за разработка на софтуер, за да изясни дали системата е точно по изискванията дадени от клиента. Прототипът е работещ модел на софтуер с някои ограничени функционалности. Прототипът не винаги запазва точната логика, която ще бъде използвана в крайния софтуер. Също така помага да се разберат изискванията, които са уникални за потребителя и не могат да бъдат изпълнени от разработчика по време на дизайна на продукта. Прототипът се дава на клиента и клиентът го използва. Освен това те предоставят обратна връзка за прототипа на разработчиците: „Какво е правилно, какво трябва да се промени, какво е липсва, какво не е необходимо и т.н. Въз основа на обратната връзка, прототипите се променят, за да се подобрят или имплементират някои от предложените промени. Следователно потребителите и клиентите могат отново да използват и еклплоатират системата. Този цикъл се повтаря.

Оценяване - Въз основа на обратната връзка, основните изисквания се променят, за да се получи окончателна спецификация, която след това се използва разработване на системата за качество.

Фиг.6 Модел на прототипиране

Основни принципи на модела на прототипизиране

- Рискът на проекта се намалява чрез разбиването на проекта по-малки секции и осигуряване на гъвкавост за промяна по време на процеса на разработка.
- Разработват се прототипи и различни версии при всяка итерация.
- Потребителят участва в цялостното разработване на софтуера, като дава обратна връзка след прототипизиране за удобряване на характеристиките и всяка друга промяна в проекта.
- Докато повечето прототипи биват създадени като изпълват само на някои от изискванията, други могат да бъдат реализирани до краен продукт, стига да изпълват изискванията на проекта.

Предимства на модела на прототипизиране

- Прототипът дава представа на потребителя как ще изглежда системата.
- Увеличава скоростта на разработка
- Потребителите се участват активно в тестването с обратна връзка
- Грешките могат да бъдат открити на по-ранен етап
- Липсващата функционалност може да се определи лесно в по-ранен етап
- Намалява се рисковият фактор от грешки в системата

Недостатъци на модела на прототипизиране

- Този модел отнема време и е по-скъп в случай в, които потребителят не харесва прототипите.
- Прототипът може да доведе до фалшиви очаквания и нови изисквания
- Основната цел на прототипите е бързата разработка. По този начин дизайнът на системата може да изглежда така както е разработен серийно, без да се вземат предвид интеграция на всички останали компоненти.
- Възможността за добавяне на нови компоненти и модули.
- Не е подходящ за големи проекти.

Гъвкави методи

Гъвкавите методи на разработка имат постепенен характер. Този метод се фокусира върху гъвкавостта на процесите и удовлетвореността на клиентите чрез бърза доставка на работещ „софтуерен продукт“. В Agile, задачите са разделени на малки времеви диапазони, които са лесни за изпълнение. Предоставят се работещи функции на края на всяка итерация. Итерацията обикновено е от две до четири седмици и включва функционален екип за работа по проекта, едновременно. Във всяка итерация се извършват всички дейности като планиране, проектиране, разработка и тестване. При гъвкавите методи, комуникацията с клента е за предпочитане пред документацията. Потребителските изисквания и желаните характеристики на системата може да бъдат превърнати в успешни резултати. Има различни видове гъвкави методи за разработване на софтуер. Някой от тях биват: Like, Scrum, Feature Driven Development (FDD[1]), Екстремно програмиране (XP[2]) и Метод за динамично развитие на системата (DSDM[3]).

Фиг.7 Процеси при гъвкавите методи

Основни принципи на гъвкавите методи

- Най-висок приоритет на гъвкавия модел е да задоволи клиента чрез ранна доставка на работещ софтуер.
- Работният продукт е основната мярка за напредък върху проекта
- Най-ефикасният и ефективен начин за предаването на информация е комуникация лице в лице
- Много екипи работят заедно за постигането на целта

Предимства на гъвкавите методи

- Насърчаване на работата в екип и кръстосване на функционален екип и екипи в обучение.
- Функционалността може да се развива бързо.
- Променящите се изисквания не са проблем, дори късно при по-напреднало развитие.
- Приоритеризират се хората и взаимодействията пред процесите и инструментите. Разработчици, клиенти и тестери редовно взаимодействат помежду си и работят заедно.

Недостатъци на гъвкавите модели

- Не се фокусират върху необходимото проектиране и документация.
- Оценяването на големите проекти в началото е трудно.
- Не е подходящ за обработка на сложни зависимости.
- Промяната в технологията може да бъде доста трудна и предизвикателна задача поради липсата на документация.

Заключение

Този реферат е фокусиран върху сравнителен анализ на различни модели за софтуерни процеси. Освен това разглежда пет различни модела на процеси, отчитащи различни фактори. Като заключение можем да кажем, че каскадният модел осигурява основата за растежът на модела и екипът за разработка, ако са запознати по отношение на пълните изисквания и среда. Итеративният дизайн е подобреният модел на каскадтият дизайн. Той дава обратна връзка за предходната версия на софтуера. Спираловият модел се занимава с риска от общата стойност на проекта за развитие и се използва за разработка на сложни и скъпи проекти. Прототипният модел се използва, когато желаната система трябва да има повече взаимодействие с крайният потребител. Гъвкавите методи приветства промените във всяка фаза. При тях е по-лесно да се правят промени често.

Този сравнителен анализ включва предимствата и недостатъците на различните видове софтуерни процеси и модели, които помагат при избора на конкретен начин за удовлетворяване на нуждите на клиента. Като също така, се имат предвид и други фактори като цена, време и ефективност.

Литература и допълнителна информация

Статии:

[1] What are the Software Development Models?
http://tryqa.com/what-are-the-software-development-models/

[2] 8 Software Development Models: Sliced, Diced and Organized in Charts
https://www.scnsoft.com/blog/software-development-models

[3] SDLC (жизнен цикъл на разработка на софтуер) Фази, методологии, процес и модели
https://bg.myservername.com/sdlc-phases

Други източници:

[1] Wikipedia.
https://en.wikipedia.org/wiki/Software_development

[2] Google Images
https://www.google.com/imghp?hl=en

Речник

[1] Feature Driven Development (FDD) - организира разработката на софтуер около функционалноста на проекта и обновяването и.

[2] Екстремно програмиране - тип програмиране и проектна методология за създаване на софтуер, една от няколкото гъвкави методологии

[3] Метод за динамично развитие на системата (DSDM) – това е рамка за предоставяне на бизнес решения, които разчитат в голяма степен на прототипирането като основна техника и самият той е одобрен по ISO 9001

Нов български университет,
деп. Информационни технологии,
/OOOK025/ Информатика

Изготвил: Александър Емилов Пилафов /F91104/

Преподавател: Доц. Д-р Петя Асенова