3.Agile методологии-Резюме

Бързата разработка и доставка често са най-важното изискване за **софтуерни системи**(С): Бизнесът оперира в бързо променяща се среда, което прави невъзможно създаването на стабилен набор от софтуерни изисквания(И). Софтуерът трябва да се развива бързо, за да отразява променящите се бизнес нужди.

Бърза разработка на софтуер

▪ Спецификацията, дизайна и имплементацията са взаимосвързани.
▪ С се разработва като **поредица от версии**, в които заинтересованите страни участват в оценката на версиите.

▪ Потребителските интерфейси често се разработват с помощта на интегрирана среда за разработка (IDE) и графични инструменти.

Гъвкави(Agile) методи(ГМ)

Недоволството от големите разходи, свързани със софт. проектиране през 80-те и 90-те години доведе до създаването на ГМ:

▪ Фокусират се върху кода, а не върху дизайна.

▪ Базирани са на итеративен подход към софтуерната разработка.

▪ Бързо доставяне на работещ софтуер и неговото бързо развитие, за да отговори на променящите се И.

Целта на ГМ е да намалят overheads in the software process (by limiting documentation) и да позволят бързо адаптиране към променящите се И без прекомерна преработка.

Манифест на Agile: Ние откриваме по-добри начини за разработка на софтуер, като го правим и помагаме на другите да го правят. Чрез тази работа осъзнахме ценността на:

- Индивидите и взаимодействията преди процесите и инструментите
- Работещия софтуер преди изчерпателната документация
- Сътрудничеството с клиента преди договарянето
- Реагирането на промени преди следването на план

Принцип	Описание
Включване на клиента	Клиентите трябва да бъдат тясно ангажирани през целия процес на разработка. Те предоставят и приоритизират новите И към С и оценяват итерациите на С.
Incremental delivery	Софтуерът се разработва на отделни increments, като клиентът определя И, които да бъдат включени във всяки increment.
Хората преди процеса	Уменията на екипа трябва да бъдат разпознати и използвани. Членовете на екипа трябва да имат свободата да развиват свои собствени начини на работа без налагане на строги процеси.
Приемане на промяната	Очаква се И към С да се променят, затова С трябва да бъде проектирана така, че да може да се адаптира към тези промени.
Поддържане на простота	Фокус върху простотата както в разработвания софтуер, така и в самия процес на разработка. Да се работи за елиминиране на сложността на С.

Приложимост на ГМ

-Разработка на продукти, когато софтуерна компания разработва **малък или средноголям продукт** за продажба.

-Разработка на **персонализирани системи** в рамките на организация, където има ясен **ангажимент от страна на клиента да участва** в процеса на разработка и **няма много външни правила и** регулации, които влияят върху софтуера.

-Поради фокуса си върху малки, тясно интегрирани екипи, ГМ имат затруднения при големи С.

Проблеми при ГМ

-Може да бъде трудно да се поддържа интересът на клиентите, участващи в процеса.

-Членовете на екипа може да не са подходящи за интензивното ангажиране, което характеризира ГМ.

-Приоритизирането на промените може да бъде сложно при наличие на множество заинтересовани страни.

-Поддържането на простота изисква допълнителна работа.

-Договорите могат да са проблем, както при др. итеративни подходи.

ГМ и поддръжка на софтуер - Повечето организации харчат повече за поддръжка на същ. софтуер, отколкото за разработка на нов. ГМ трябва да поддържат както поддръжката, така и първонач. разработка

Два ключови въпроса:

-Могат ли С, разработени с ГМ, да бъдат поддържани, тъй като процеса на разработка минимизира формалната документация?

-Могат ли ГМ да се използват ефективно при промени в С при искане от клиента?

Могат да възникнат проблеми, ако оригиналният екип за разработка не може да бъде запазен.

Планово-управляема и гъвкава разработка

Планово-управляема(ПУ) разработка:

- отделни етапи на разработка, като резултатите от всеки етап се планират предварително.
- Не е задължително да следва водопадния модел – **ПУ**, итеративна increments- разработка е възможна.
- Итерациите се случват в рамките на отделни дейности.

Гъвкава разработка-Спецификацията, дизайнът, имплементацията и тестването са взаимно свързани, като резултатите от процеса на разработка се определят чрез процес на договаряне.

Технически, човешки и организационни въпроси: Повечето проекти включват елементи както от **ПУ**, така и от гъвкавите процеси. Балансът зависи от следните фактори:

-Необходимост от детайлна спецификация и дизайн преди имплементация – ако това е критично, е по-добре да се използва **ПУ**

-Итеративна/ incremental/ доставка – ако е възможно да се доставя софтуер на клиентите и да се получава бърза обратна връзка - ГМ

-Размер на С – ГМ работят най-добре за малки, ко-локирани екипи, докато големите системи може да изискват **ПУ** подход.

-Тип на системата – сложни системи с И за анализ преди имплементация (например реалновремеви системи с времеви ограничения) често изискват **ПУ** подход.

-Очакван жизнен цикъл на С – дългоживеещите С може да изискват повече документация за предаване към бъдещи екипи.

-Налични технологии – ГМ разчитат на добри инструменти за проследяване на дизайна.

-Организация на екипа – ако екипът е разпределен или част от разработката е възложена на външни изпълнители, може да е необходимо създаване на документация.

-Културни и организационни фактори – традиционните инженерни организации често следват **ПУ** методи.

-Ниво на умения в екипа –ГМ изискват по-високо ниво на умения в сравнение с **ПУ** подходи.

-Регулаторни И – ако софтуерът подлежи на външна регулация (например сертификация от FAA за авиационни системи), ще бъде необходимо създаване на детайлна документация.

Extreme Programming (XP) -Най-известният и широко използван гъвкав метод. XP следва "екстремен" подход към итеративната разработка:

-Нови версии могат да се изграждат няколко пъти на ден.

-increments се доставят на клиентите на всеки 2 седмици.

-Всички тестове трябва да бъдат изпълнени за всяка нова версия, и изграждането се приема само ако тестовете преминат успешно.

XP и принципи на Agile

- **Итеративна/incremental/ разработка** чрез малки и чести издания на С.
- **Ангажираност на клиента** - постоянно участие в екипа.
- **Хората преди процесите** - pair програмиране, колективна собственост върху кода и избягване на продълж. работно време.
- **Поддръжка на промени** чрез редовни издания на С.
- **Поддържане на простота** - постоянно рефакториране на кода.

Принцип или практика	Описание
Итеративно /incremental/ планиране	И се записват на потребителски истории (**story cards**), а историите, които ще бъдат включени в дадено издание, се определят според наличното време и относителния им приоритет. Разработчиците разбиват тези истории на задачи.
Малки издания	Най-минималният полезен набор от функционалности, който осигурява бизнес стойност, се разработва първо. Изданията на С са чести и постепенно добавят нова функционалност към първоначалн. издание.
Опростен дизайн	Извършва се само толкова проектиране, колкото е необходимо за изпълнение на тек. И – нито повече, нито по-малко.
Разработка, ориентирана към тестове (Test-first development)	Използва се автоматизирана среда за модулно тестване (unit test framework), за да се напишат тестове за нова функционалност, преди да бъде имплементирана.
Рефакториране	Разработчиците рефакторират кода непрекъснато веднага щом се открият възможности за подобрение. Това помага кодът да остане прост и лесен за поддръжка.

Практики на Extreme Programming (XP)

Практика	Описание
Партньорско програмиране (Pair programming)	Разработчиците работят по двойки, проверявайки работата си взаимно и осигурявайки подкрепа, за да постигат винаги високо качество.
Колективна собственост (Collective ownership)	Двойките разработчици работят върху всички части на системата, така че да не се формират „острови на експертиза“.Всички разработчици носят отговорност за целия код, а всеки може да прави промени навсякъде.
Непрекъсната интеграция (Continuous integration)	Веднага след като работата по дадена задача е завършена, тя се интегрира в цялостната С. След всяка такава интеграция всички модулни тестове в С трябва да преминат успешно.
Устойчиво темпо (Sustainable pace)	Големи количества извънреден труд не се считат за приемливи, тъй като в дългосрочен план водят до намаляване на качеството на кода и продуктивността.
Клиент на място (On-site customer)	Представител на Клиентът трябва да бъде на разположение на пълен работен ден за ХР екипа. Клиентът е част от екипа за разработка и е отговорен за предоставянето на И към С.

Сценарии за И

-В ХР клиентът е част от ХР екипа и е отговорен за вземането на решения относно И.

-Потребителските И се изразяват чрез сценарии или потреб. истории

-Те се записват на карти, а екипът по разработка ги разбива на задачи за изпълнение- основата за графици и разчети на разходите.

-Клиентът избира историите, които да бъдат включени в следващото издание, на базата на техните приоритети и оценките за графика.

XP и промяната

-В традиционното софтуерно инженерство се смята, че трябва да се планира за бъдещи промени, тъй като това намалява разходите в по-късен етап.

-XP обаче твърди, че предварителното прогнозиране на промените не е ефективно.

-Вместо това XP предлага постоянно рефакориране кода за да се улесни адаптирането, когато се налага промяна.

Рефакториране:

-Екипът за разработка търси възможности за подобрение на софтуера и ги прилага дори когато няма непосредствена нужда.

-Подобрява четимостта на кода и намалява нуждата от документац.

-Промените стават по-лесни- кодът е добре структурир. и разбираем.

-Някои промени изискват рефакториране на архитектурата, което е значително по-скъпо.

Примери за рефакториране

-Реорганизация на йерархията на класовете за премахване на дублиращ се код.

-Подреждане и преименуване на атрибути и методи за по-добра четимост.

-Замяна на вграден код с извиквания към методи от библиотеката.

Основни моменти

-ГМ са итеративни методи за разработка, които се фокусират върху бързото развитие, честите издания, намаляване на бюрокрацията и осигуряване на висококачествен код.

-Решението дали да се използва гъвкав или традиционен метод зависи от вида на софтуера, възможностите на екипа и културата на компанията.

-XP е добре познат гъвкав метод, който включва добри практики като чести издания, постоянно подобрение на кода и активно участие на клиента в разработката.

Тестване в XP - централен аспект на XP и се извършва след всяка промяна в кода. Основни характеристики на XP тестването:

-Разработка на тестове преди кода.

-Инкрементално развитие на тестовете от сценарии.

-Участие на потребителите в разработването и валидирането на тестовете.

-Автоматизирани тестови рамки за изпълнение на всички тестове при всяко ново издание.

Разработка на тестове преди кода

- Писането на тестове преди кода уточнява И, които трябва да бъдат изпълнени.
- Тестовете се пишат като програми, а не като данни, така че да могат да бъдат изпълнявани автоматично.
- Често се използват тестови рамки като JUnit.

- Всички стари и нови тестове се изпълняват автоматично при добавяне на нова функционалност.

Участие на клиента в тестването

-Клиентът помага при разработването на приемателни тестове за историите, които ще бъдат реализирани в следващото издание.

-Клиентът, като част от екипа, пише тестове по време на разработка

-Клиентът често няма достатъчно време да работи на пълен работен ден с екипа и може да е неохотен да участва в процеса на тестване.

Автоматизация на тестовете

-Тестовете се пишат като изпълними компоненти преди реализирането на задачата. Тези компоненти трябва да бъдат самостоятелни, да симулират входните данни и да проверяват резултатите.

-Автоматизирани тестови рамки като JUnit улесняват писането и изпълнението на тестовете. Автоматизацията на тестовете позволява бързо и лесно тестване на цялата система при всяка промяна.

Трудности при тестването в XP

- Програмистите предпочитат да пишат код, вместо тестове, и понякога правят **компромиси** при тестването.
- Някои тестове са трудни за писане, например тези **за сложни потребителски интерфейси**.
- Трудно е да се оцени доколко един набор от тестове е пълен.

Pair програмиране

- В XP програмистите работят по двойки, седейки заедно на едно работно място.

- Това помага за колективната собственост върху кода и разпространението на знания в екипа.
- Pair програмирането действа като неформален преглед, тъй като кодът се проверява от двама души.
- Подпомага процеса на рефакториране, защото цялото развитие на кода е колективно.
- Производителността при разработка с **Pair** програмисти е сходна с тази на двама души, работещи самостоятелно.
- Двойките се създават динамично, така че всички членове на екипа да работят заедно по време на разработката.
- Споделянето на знания, което се случва при **Pair** програмиране, е много важно, тъй като намалява общите рискове за проекта, когато членове на екипа напуснат.
- **Pair** програмиране не е непременно неефективно и има доказателства, че двойка, работеща заедно, е по-ефективна от двама програмисти, работещи поотделно.

Предимства на pair програмиране

-Поддържа идеята за колективна собственост и отговорност върху С.

-Индивидуалните разработчици не носят отговорност за проблемите в кода. Екипът има колективна отговорност за тяхното решаване.

-Действа като неформален процес на преглед, тъй като всеки ред код се проверява от поне двама души.

-Подпомага рефакториране, тъй като всички членове на екипа веднага виждат подобренията.

Гъвкаво управление на проекти

- Основната роля на Project-мениджъра е да гарантира, че софтуерът ще бъде доставен навреме и в рамките на бюджета.

- Стандартният подход за управление на проекти е планов.
- Гъвкавото управление на проекти изисква разл. подход, който се адаптира към итеративното разв. и силните страни на ГМ.

Scrum - гъвкав метод, който се фокусира върху управлението на итеративното развитие отколкото гъвкави практики. 3 основни фази:

-Първоначално планиране и дефиниране на архитектурата.

-Поредица от спринтове- всеки спринт създава инкремент в С

-Финална фаза, в която се завършва документацията / help frames, user manuals/ и се прави анализ на научените уроци.

Спринт цикъл: Спринтовете са с фиксирана продължителност 2–4 седмици. Те съответстват на разработването на версия/ a release/ на С в XP. Началната точка за планиране е **product backlog -списък със задачите**, които трябва да бъдат изпълнени в проекта. Фазата на подбор включва целия проектен екип и клиента, за да избере функциите и features, които ще се разработват в спринта. След като тези параметри бъдат съгласувани, екипът сам организира работата си по разработката на софтуера. През този етап екипът е изолиран от клиента и организацията, като всички комуникации преминават през **Scrum master**- защитава екипа за разработка от външни разсейвания. В края на спринта свършената работа се преглежда и представя на заинтересованите страни. След това започва следващия спринт.

Работа в екип при Scrum:

Scrum master е координатор, който организира ежедневни срещи, следи беклога със задачи, документира решенията, измерва напредъка и комуникира с клиенти и мениджмънта.

Целият екип участва в кратки ежедневни срещи- daily meetings, където всички членове споделят информация, описват напредъка си

от последното дейли, обсъждат възникнали проблеми и планират работата за следващия ден. Това гарантира, че всички в екипа знаят какво се случва и, ако възникнат проблеми, могат бързо да преразпределят задачите, за да се справят с тях.

Scrum ползи

- Продуктът е разделен на малки, управляеми части.
- Променливите И не спират напредъка.
- Всички членове на екипа имат видимост върху развитието на проекта и се подобрява комуникацията в екипа.
- Клиентите виждат навременни издания- on-time delivery of increments и могат да дават обратна връзка.
- Създава се култура на доверие и увереност в успеха на проекта.

Скалиране на ГМ –ГМ успешни за малки и средни проекти, разработвани от екип на едно място. За големи системи е необходимо предварително проектиране и документация. Междуекипната комуникация трябва да бъде организирана с редовни срещи и видео конференции - успехът на тези методи се дължи на подобрената комуникация, която е възможна, когато всички работят заедно

Разширяването на гъвкавите методи включва тяхното адаптиране, за да се справят с по-големи и по-продължителни проекти, при които има множество екипи за разработка, работещи на различни места

-XP подчертава важността на **автоматизираните тестове и рефакториране**. **Scrum** осигурява рамка за управление на **гъвкавите проекти чрез спринтове**. **Скалирането на ГМ** за големи проекти изисква баланс между **гъвкавост и предварително планиране**.