# Гъвкави методологии за разработване на софтуерни системи

**1. Същност на гъвкавите (Agile) методологии.**

**Гъвкави (Agile) методологи** за разработка на софтуер е неформален сбор от методологии и техники за управление на проекти за разработка на софтуер. Във фокуса на гъвкавите методологии е идеята, че разработката на софтуер е динамичен процес, в който дългосрочното планиране има ограничена ефективност.

Гъвкавите методологии намират широко приложение в разработката на продукти, където чрез създаване на прототипи, производителите имат възможност да получат обратна връзка от клиентите и да адаптират разработката според новопостъпилите от това изисквания.

Гъвкавата методология за разработка на софтуер се състои от група методи при създаването на софтуер, базирани на повтарящо се и постепенно разработване. Софтуерните спецификации и решения се развиват поетапно чрез сътрудничество между самоорганизиращи се екипи, съставени от хора с различни функции. Гъвкавата методология насърчава адаптивно планиране, еволюираща разработка и доставяне на софтуера, времево-разпределен итеративен подход. Тя поощрява бързото и гъвкаво реагиране на промени. Това е концептуална рамка, която насърчава предвидени взаимодействия през цялото времетраене на разработката на софтуерни проекти.

Един софтуерен процес може да се разглежда като „гъвкав", ако позволява разработката на софтуер да е инкрементална, в сътрудничество с клиента (добра комуникация между разработчик и клиент), адаптивна (да допуска промени) и лесна.

Терминът гъвкаво разработване на софтуер (agile software development) е въведен през 2001 година в Манифестът на гъвкавите методологии (The Agile Manifesto) [1] от разработчици на софтуер. Част от авторите на манифеста създават The Agile Alliance, неправителствена организация, която насърчава разработката на софтуер в съответствие с принципите на манифеста.

Манифестът на гъвкаво разработване на софтуер гласи следното:

„Откриваме по-добри методи за разработване на софтуер, като ги практикуване и помагаме на другите да го правят. В процеса на работа стигнахме до следните важни изводи:

- ✓ Фокусът трябва да бъде върху индивидите и взаимодействията, а не върху процесите и инструментите;
- ✓ Работещ софтуер е по-важен от изчерпателната документация;
- ✓ Сътрудничеството с клиентите е по-важно от преговорите по договора;
- ✓ Процесът трябва да реагира на промените, а не да следва план."

The Agile Manifesto се основава на следните дванадесет принципа:

1. Удовлетворение на клиентите чрез бърза доставка на полезен софтуер.
2. Промяна в спецификациите е възможна, дори и в късните фази на проекта.
3. Често предоставяне на работещ софтуер (в периоди от седмици, а не месеци).
4. Работещият софтуер е основната мярка за напредък.
5. Устойчиво развитие, което успява да поддържа постоянно темпо.
6. Тясно, ежедневно сътрудничество между бизнес служители и разработчици.

7. Разговорите лице в лице са най-добрата форма на комуникация (съвместна локация).
8. Проектите се изграждат около мотивирани хора, на които се има доверие.
9. Непрекъснато внимание към техническо съвършенство и добър дизайн.
10. Простота – изкуството максимално количество работа да бъде пропусната, е от съществено значение.
11. Самоорганизиращи се екипи.
12. Редовна адаптация към променящи се обстоятелства.

Гъвкавите методи разбиват задачите на малки стъпки с минимално планиране, без да засягат дългосрочното планиране на проекта. Итерациите (етапите) стават на кратки срокове (timeboxes), които обикновено траят от една до четири седмици. През всяка итерация екипът, съставен от хора с различни функции, работи по всяка една от функциите: планиране, анализ на изискванията, проектиране, разработка, тестване и проверка при приемане. В края на итерацията работещият софтуер се представя пред заинтересованите страни. Така се намалява цялостния риск и проектът може да бъде адаптиран бързо към промени. Една итерация може да не добавя достатъчно функционалност, за да обоснове пускане на пазара, но целта е да съществува работещо решение (с минимални грешки) в края на всяка итерация. За да се завърши даден софтуер или функционалност, могат да бъдат нужни множество итерации.

Независимо кои дисциплини за програмиране се изискват, всеки екип съдържа представител на клиента. Той се назначава от заинтересованите страни и действа от тяхно име като има личен ангажимент да бъде на разположение на разработчиците за отговори на въпроси, възникнали по време на дадена итерация. В края на всяка итерация, заинтересованите страни и представителят на клиента преглеждат заедно напредъка на проекта и оценяват приоритетите с оглед оптимизиране на възвръщаемостта на инвестициите (ROI) и съобразяване с нуждите на клиента и целите на компанията.

Обща характеристика на гъвкавите методи за разработка са ежедневните срещи относно напредъка на проекта или т.н. стенд-ъп срещи (stand-ups). По време на кратката среща, членовете на екипа докладват пред всички какво са направили предишния ден, какво възнамеряват да правят днес и има ли нещо, което ги възпрепятства да свършат задачите си.

Специфични инструменти и техники, като например, непрекъсната интеграция, автоматизирани xUnit тестове, програмиране по двойки, разработка чрез тестове (test-driven development), шаблони за дизайн (design patterns), дизайн според сферата (domain-driven design), преработка на код (code refactoring) и други техники, често се използват за подобряване на качеството и повишаване на гъвкавостта на проекта.

В гъвкавите методологии за разработка на софтуер се използват информационни радиатори (напр. дъска с листчета) – те онагледяват физически прогреса на проекта на видно място в офиса, където минувачите могат да го видят. Представляват актуално обобщение на състоянието на един софтуерен проект или друг продукт. Името е въведено от Алистър Кокбърн и е описано в неговата книга от 2002 година, „Гъвкава разработка на софтуер“. Могат да бъдат използвани и светлинни индикатори, информиращи екипа за текущото състояние на проекта.

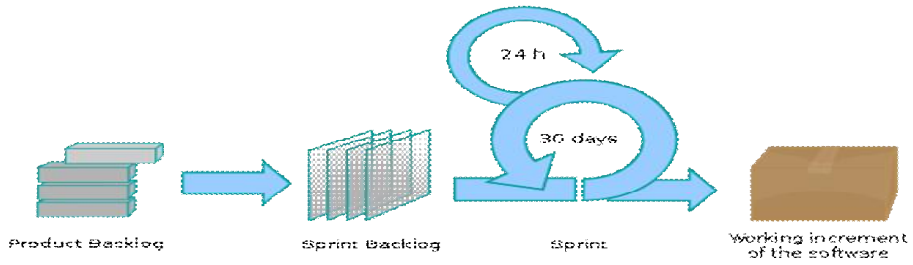
Някои от основните компоненти на гъвкавата методология за разработване на софтуер включват:

## 2. Скръм (Scrum)

Scrum е процес, използван при изготвяне и управление на големи проекти. Той е разработен с цел дългосрочното планиране на изработването на даден продукт да бъде улеснено значимо. За разлика от типичния мениджмънт чрез контрол и командване, при Scrum процесите се набляга на обратната връзка и се дава повече власт на хората, вършещи „черната работа“.

Scrum е итеративна, инкрементална управленска рамка с много широки възможности за контрол и управление на проекти.

Scrum процесът се състои от отделни итерации, наречени спринтове. Спринтовете могат да имат продължителност от една седмица до четири седмици. В края на всеки спринт екипът разполага с работеща версия на продукта, която включва всички готови задачи от backlog-а. (фиг.1).




*Фиг. 1. Scrum процес [1]*

Спринтът е най-малката единица време за разработване. Спринтовете са с постоянна дължина от 1 седмица до 1 месец. Всеки спринт е опит за подобрение вкаран във фиксирани времеви рамки.

Преди всеки спринт има среща за планиране на спринта. На нея се поставят измеримите цели за спринта и се идентифицират задачите, които ще бъдат свършени в неговите рамки. По време на всеки спринт екипът създава завършени парчета от даден продукт. Дейностите за всеки спринт се описват и взимат от „продуктовата опора“ или на английски „product backlog” (фиг. 3). Често тези дейности са описани като характеристики, които продукта трябва да има и да бъдат постигнати за спринта. Product backlog е приоритизиран списък на изискванията. Какво от списъка да влезе в даден спринт се решава на планиращата среща преди спринта. Продуктовият собственик уведомява екипа кои части от списъка с изисквания иска да бъдат свършени на предстоящия спринт. Развойният екип преглежда, обсъжда, решава и записва в Sprint Backlog кои от тези изисквания и цели ще успее да изпълни на предстоящия спринт. Sprint Backlog е собственост на развойния екип. Целите, вписани в този документ, не трябва да бъдат променяни по време на спринта. За разработването се определя фиксирана продължителност, такава, че спринтът да свърши навреме. Изискванията, които не бъдат удовлетворени за спринта се изключват и връщат към product backlog. След като спринтът е изпълнен, екипът демонстрира как се използва софтуерът.

SCRUM позволява организирането на самоорганизиращи се екипи.

Ключов принцип на SCRUM е това, че се приема още в самото начало на проекта, че изискванията няма как да са пълни и напълно разбрани. Т.е. очакват се нововъведения от клиента – промяна на желанията му. SCRUM се фокусира върху способността на екипа да доставя бързо и да е готов да отговори бързо на неочакваните промени.

Както в другите гъвкави методологии за управление на проекти, SCRUM може да се имплементира чрез различни видове инструменти. Най-много се използват спредшийтове (екселски таблици или еквивалента им в Google Docs). В тях се правят така наречените SCRUM „артефакти“ като sprint backlog-а. Други организации прилагат SCRUM с помощта на бели дъски, лепящи се листчета и хартиени документи.

По време на спринта, всеки ден се провеждат т.нар. правостоящи срещи (stand-up meetings). Тези срещи продължават от 5 до 15 минути и се провеждат ежедневно в определен час. На срещата всеки от екипа абсолютно неформално разказва за три неща:

- какво е работил предишния ден;
- какво планира за предстоящия ден;
- какви проблеми е срещнал, които му пречат да работи.

Главни атрибути на срещата са:

- ✓ Срещата започва точно навреме.
- ✓ Мястото и часът са едни и същи всеки ден.
- ✓ Срещата трябва да се вмести точно в 15 минути.
- ✓ Може всички, но главно говорят основните роли в екипа.

На срещата се обновява и backlog-ът като се отбелязва свършената работа.

Ако се идентифицират някакви проблеми, те се решават колективно. Важно е да се отбележи, че това не са срещи за отчет пред ръководството, а за синхронизация (самоорганизация) на екипа и разкриване на потенциални пречки в работата.

Специфични практики на на SCRUM са:

- ➢ Ежедневни правостоящи срещи (Stand-up meetings);
- ➢ Backlog: списък със задачите за текущия спринт и тяхното състояние;
- ➢ Самоорганизиращ се екип: екипът не следва предварително раздадени задачи, а всеки негов член се стреми да допринася за постигне целите на спринта – всеки ден всеки си взима задачи, за които отговаря;
- ➢ Работни срещи и обсъждания с клиента и с екипа след всеки спринт.

### 3. Методът Канбан (Kanban)

Организациите използват метода Канбан, за да управляват създаването на проекта, като същевременно акцентират върху непрекъснатата доставка и не претоварват екипа за разработка. Подобно на Scrum, Kanban процесите са предназначени да помагат на екипите да работят по-ефективно заедно.

**Канбан** е метод за управление на интелектуални дейности, който набляга на доставката точно на време, без същевременно членовете на екипа да са пренатоварени. Името на този метод произлиза от японски език и преведено буквално означава „сигнална карта“.

Канбан може да бъде разделен на две части:

- ✓ Канбан – Система за визуално управление на процеси, която указва какво да се произведе, кога да се произведе и какво количество да бъде произведено.
- ✓ Методът Канбан – Подход за постепенно, еволюционно подобряване на процесите в една организация.

Канбан системата се базира на следните принципи:

- Визуализирайте това, което правите: вижте всички елементи в контекста един на друг - по-информативни.

- Ограничете количеството на текущата работа (WIP): балансирайте подхода, базиран на потока, така че екипите не се ангажират да правят твърде много работа наведнъж.
- Веднага след като една задача е завършена, започнете от следващата.

Канбан методологията е свързана с четири базови принципа:

- Започни с това, което правиш сега;

Канбан не определя специфичен набор от роли и стъпки. Методът започва с ролите и процесите, които са на разположение и стимулира продължителни, нарастващи и еволюционни промени в системата.

- Съгласявай се да преследваш постепенни еволюционни промени;

Организацията (или екипът) трябва да се съгласи, че продължителните, нарастващи и еволюционни промени са начинът да се правят подобрения в системата.

- Уважавай текущите процеси, роли, отговорности и титли;
- Лидерство на всички нива.

Методът Канбан има следните ключови практики:

✓ Визуализирай

Работният процес при интелектуалната дейност е присъщо невидим. Визуализирането на потока на работа е ключов за разбирането на това как върви работата. Познат метод за визуализиране на работния процес е използването на стена с колони и карти. Колоните представляват различните състояния или стъпки на работния процес.

✓ Ограничавай задачите, които са „в процес на работа“

Ограничаването на задачите, които са „в процес на работа“ гласи, че трябва да се имплементира система за поемане („теглене“) на задачи, касаещи части от работния процес. Критичните елементи са, че текущата работа във всеки един етап от работния процес е лимитирана и нови задачи се „изтеглят“, когато има наличен капацитет в ресурса до достигане на лимита за все още незавършена работа.

✓ Управлявай потока на работа

Процесът на работа през всеки етап трябва да бъде следен, измерван и докладван. Чрез активно управление на процеса може да бъде оценено дали продължителни, постепенни промени в системата ще имат положителни или отрицателни ефекти.

✓ Уточнявай изискванията

✓ Използвай обратна връзка

✓ Усъвършенствай съвместно, еволюирай експериментално (използвайки модели и научни методи)

Методът Канбан окуражава малки, продължителни и еволюционни промени, които остават. Когато екипите споделят еднакво разбиране на работните теории, работния процес и рисковете, е по-вероятно за тях да изградят споделени разбирания за даден проблем и да предложат действия за подобрение, с които да се съгласят. Методът предлага използването на научен подход за въвеждането на продължителни, постепенни и еволюционни промени. Методът не определя специфичен научен подход, който да се използва.

Методът Канбан насърчава непрекъснатото сътрудничество на клиента и екипа. Той насърчава продължаващото обучение и подобрения, за да осигури възможно най-добрия работен процес за екипа.

**4. Екстремно програмиране (Extreme Programming – XP)**

XP е метод за непрекъснато доставяне на висококачествен софтуер по-бързо. Клиентът е активно ангажиран с екипа за тясно сътрудничество, за да извършва непрекъснато планиране, тестване и бърза обратна връзка, за да доставя работещ софтуер често. Софтуерът трябва да се доставя на интервали от всеки до три седмици.

Основната цел на XP методологията е да редуцира цената на проект, ако се наложи дадена промяна. Тя е подходяща да се използва при проекти, който имат често променящи се изисквания и тогава по-стандартни методологии са оптимални за постигане на голяма продуктивност; подходяща е при проекти, които включват нови технологии или непредвидими проблеми, свързани с имплементацията; използва се също така при по-малки и по-лесни за реализация проекти с неофициални методи; добра технология за проекти, изискващи изследване.

Дейности, който XP описва и който се използват при самия процес на софтуера разработка:

- ✓ **Кодиране** –важно в разработката на софтуер е писането на код.
- ✓ **Тестването** – без тестване не можем да бъдем сигурни дали нашия продукт наистина отговаря на изискванията на спецификацията; не можем да сме сигурни дали това, което сме написали е това, което сме искали да напишем – за целта използваме Unit Test.
- ✓ **Слушане** – За програмистите по принцип не е необходимо да са наясно с бизнес страната на самата система. От друга страна те трябва да „слушат“, за да знаят от какво се нуждае клиента.
- ✓ **Правене на дизайн** – едно от най-важните неща е създаването на добър дизайн в началото.

XP се основава на простота, общуване, обратна връзка и смелост. Тази методология има 12 поддържащи практики**:**

1. Планиране на игра
2. Малки издания
3. Тестове за приемане от клиента
4. Прост дизайн
5. Програмиране на двойки
6. Разработено от тест
7. Рефакториране
8. Непрекъсната интеграция
9. Колективна собственост върху кода
10. Стандарти за кодиране
11. Метафора
12. Устойчиво темпо

**5. Кристал (Crystal)**

Кристалната методология е един от най-леките и адаптивни подходи в разработването на софтуер. Тя се състои от няколко гъвкави процеси, включително ясни, кристално жълти, кристално оранжеви и други уникално охарактеризирани. Има няколко фактора, които стимулират тези процеси, включително: размера на екипа, критичността на системата и приоритетите на проекта.

Crystal се фокусира върху осъзнаването, че всеки проект има уникални характеристики, следователно, политиките и практиките трябва да бъдат персонализирани, за да отговарят на тези характеристики.

Кристалният метод има няколко основни принципа:

- Съвместна дейност
- Общуване
- Простота
- Размисъл
- Чести корекции
- Подобряване на процесите

Този гъвкав процес, подобно на други методологии, насърчава ранно и често работещо доставяне на софтуер. Насърчава активното участие на потребителите, приспособимостта и елиминирането на разсейването и бюрокрацията.

### 6. Lean методология

Стегнатата разработка на софтуер (**Lean software development (LSD)**) представлява адаптиране на принципите и похватите на стегнатото производство и стегнатите ИТ (lean IT) за приложение в сферата на софтуерното инженерство.

Стегната разработка на софтуер е итерационна методология, първоначално разработена от Mary и Tom Poppendieck. Фокусира се върху това да даде на клиента ефективен механизъм "Стойност-поток", който доставя стойността на проекта.

Основните принципи на тази методология са:

- Премахване на отпадъците *(отърви се от излишното);*
- Увеличете ученето;
- Вземайте решения възможно най-късно;
- Поставете резултати възможно най-бързо;
- Дайте възможност на екипа;
- Изградете целостта;
- Представете си целия проект.

Всичко, което не допринася полза за клиента, се счита за излишно, за отпадък. Това включва:

- ✓ ненужен код и функционалности;
- ✓ забавяния в процеса на софтуерната разработка;
- ✓ неясни изисквания;
- ✓ недостатъчно тестване, водещо до повторения на определени процеси, които могат да бъдат предварително избегнати;
- ✓ бюрокрация;
- ✓ бавна вътрешна комуникация.

За да бъдат елиминирани излишъците, те трябва да бъдат идентифицирани. Ако някоя от дейностите може да бъде избегната или резултатът може да бъде постигнат и без нея, тя е излишна. Код, написан донякъде и изоставен в хода на процеса на разработка на софтуера, е излишък. Допълнителни процеси и функционалности, които не се използват често от потребителите, е излишък. Изчакване на други дейности, екипи или процеси е излишък. Дефектите и ниското качество са излишък. Мениджмънт нива, които не добавят

реална стойност, са излишък. За да се отличат и разпознаят излишъците, се използва техниката на value stream mapping. Втората стъпка е да се идентифицират източниците на излишъци и да се елиминират. Тези стъпки би следвало да се повтарят итеративно дотогава, докато дори процеси и процедури, които привидно изглеждат съществени, бъдат идентифицирани и премахнати.

Разработката на софтуер е продължителен процес на учене с допълнителните предизвикателства, поставяни от размера на екипите по разработка и на крайния продукт. Най-добрият подход за подобряване на средата за разработка на софтуер е да се подобри ученето. Натрупването на дефекти трябва да бъде избягвано чрез провеждане на тестове веднага след като кодът бъде написан. Вместо добавяне на допълнителна документация или детайлно планиране, различни идеи могат да бъдат изпробвани чрез писане на код и асемблиране. Процесът на събиране на потребителските изисквания може да бъде опростен чрез представяне на примерни екрани на крайните потребители и обратна връзка от тях.

Процесът на учене допълнително се ускорява от кратки итерационни цикли – всеки последван от рефакторизация и интеграционно тестване. Интензификацията на обратната връзка чрез кратки сесии с потребителите помага за определянето на актуалната фаза на разработка и калибриране на усилията за бъдещи подобрения. По време на тези кратки сесии както представителите на потребителите, така и екипите от разработчици научават повече за конкретния проблем и идентифицират възможни решения за бъдещо развитие. По този начин самите потребители разбират по-добре своите нужди, базирано на резултатите от усилията на разработчиците до момента, а разработчиците научават как по-добре да удовлетворят нуждите на потребителите.

Тъй като разработката на софтуер винаги е свързана с известна доза неизвестност, по-добри резултати биха се получили отлагайки решението колкото е възможно повече, така че то да може да бъде базирано на факти, а не на несигурни предположения и очаквания.

Гъвкавата методология за разработка на софтуер предвижда предоставянето на възможностите на потребителите по-рано в процеса на разработка, като по този начин критичните решения се отлагат за по-късен етап, след като потребителите са осъзнали своите нужди в по-пълна степен. Това позволява и по-късна адаптация към промени и предотвратява скъпите по-ранни решения, зависими единствено от технологиите.

В началото, потребителят предоставя необходимите входящи изисквания. Те могат да бъдат оформени и съвсем просто, като малки карти или истории, като разработчиците оценяват времето, необходимо за имплементацията на всяка карта. Така организацията на работата се превръща в саморазвиваща се система – всяка сутрин по време на работна среща всеки член на екипа прави равносметка на свършеното през вчерашния ден, какво предстои да бъде направено през днешния и утрешния и получава необходима информация от своите колеги или от потребителя. Това изисква прозрачност на процеса, което е от полза и за комуникацията в екипа.

Стегнатият подход подкрепя афоризма "намерете добри хора и ги оставете да си вършат сами работата", окуражава напредъка, улавя грешките и премахва пречките, но не управлява на микро ниво.

Клиентът трябва да има цялостен опит със системата – това е така наречения възприеман интегритет: как системата се рекламира, доставя, инсталира, достъпва, колко интуитивна е употребата ѝ, каква е цената ѝ и доколко добре решава проблеми.

Концептуалната цялост означава, че отделните компоненти на системата работят добре заедно, като едно цяло, с баланс между гъвкавост, възможност за поддръжка, ефективност и отзивчивост. Необходимата информация се получава на малки части – не в един огромен документ, за предпочитане е комуникацията лице в лице, а не писмената документация. Потокът на информация трябва да бъде постоянен и в двете посоки – от клиент към разработчиците и обратно, избягвайки по този начин голямото стресиращо количество информация след дълга разработка в изолация.

Софтуерните системи в днешно време не са просто сбор от части, а също така и продукт от техните взаимодействия. В софтуера се натрупват дефекти при разработка – чрез разбиване на големите задачи на по-малки и чрез стандартизиране на различните етапи на разработка. Колкото по-голяма е системата, толкова повече организации участват в разработката ѝ и толкова повече нейни части са разработени от различни екипи, тогава толкова по-важно е да има добре дефинирани връзки между различните доставчици, за да се получи система с добро взаимодействие на компонентите.

Принципите на Lean медодологиите се базират на лозунга "Мисли мащабно, действай на дребно, проваляй се бързо, учи бързо".