МОДЕЛИ ЗА РАЗРАБОТКА НА СОФТУЕР

Съдържание

- Модели за разработка на софтуер
  - Модел тип “Waterfall” (Водопад)
  - V-модел (Последователен модел за разработка)
  - Итеративни и инкрементални модели (Iterative-incremental Development Models)
    - Модел тип “Agile” (Гъвкав модел)
      - Модел ръководен от тестовия процес (Test driven development)
      - Модел екстремно програмиране (Extreme Programming model)
      - Модел за разработка Lean
      - Модел за разработка Scrum

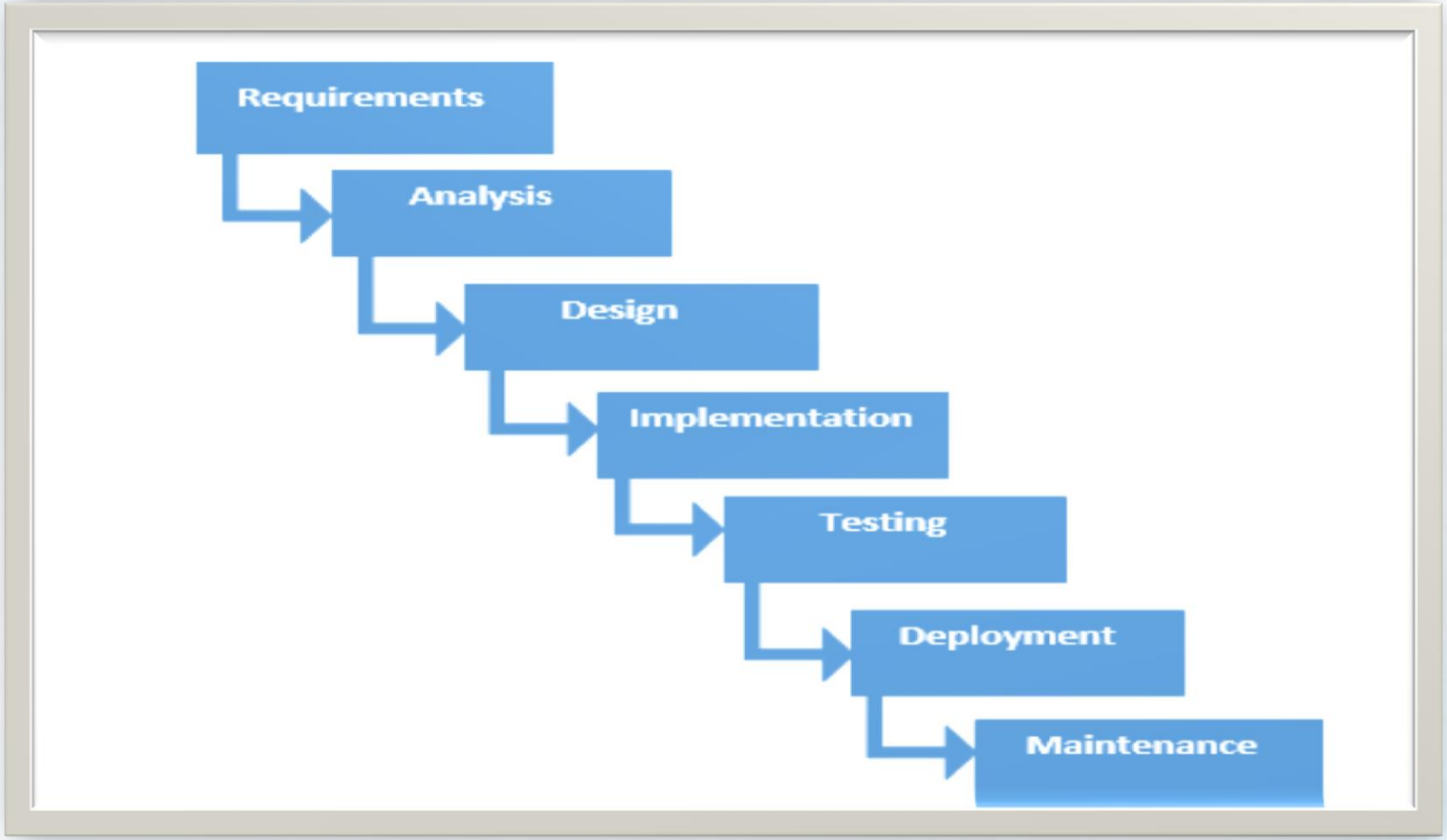
Какво представлява модел за разработка на софтуер?

- Моделът за разработка на софтуер е съвкупност от практики и принципи за организиране на процеса на разработване.
  - Определени правила който програмистите трябва да спазват
  - Определени правила (конвенции) които организацията трябва да спазва
  - Систематичен подход за организиране и поддържане на софтуерните проекти

Етапи в развитието на софтуера

- Документиране на изискванията на клиента (Business Requirements)
- Дизайн на софтуерния продукт(Software product design)
- Създаване/разработка на софтуерния продукт (Implementing the software product)
- Тестване на софтуерното решение (Testing the software solution)
- Представяне на продукта на клиента (Deployment of the solution)
- Поддръжка на продукта (Maintenance of the product)

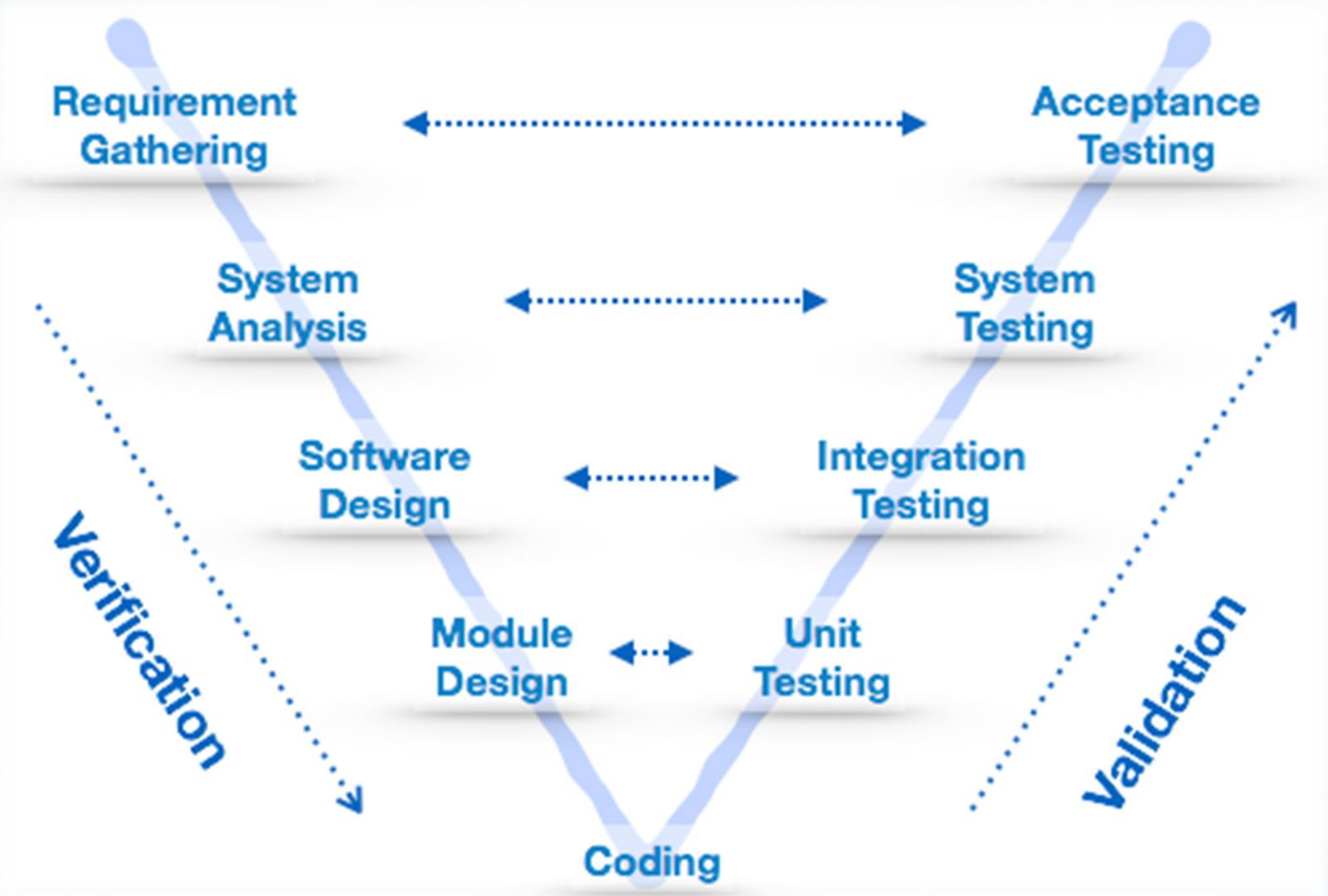
Модел за разработка на софтуер тип “Waterfall”



Характени черти на модела Waterfall

- Първия основен модел за разработка на софтуер
- Много лесен начин за разработка и добре известен (датира от средата на 70-те години на 20 век)
- Едва когато един етап приключи се започва със следващ
- Има възможност да се правят ревизии и проверки на близки нива (revision and feedback loops on adjacent levels)
- Основна роля играя документацията и стриктното и спазване
- В случая тестването се приема че е еднократно действие в края на разработката на продукта (Testing is a "one time" action at the end of the development)

V – модел за разработка



Основни принципи на V-модела за разработка

- Модела има две разклонения (branches)
  - Задачите на програмиране (Development tasks)
    - Това е процеса на дизайн и създаване на продуктът(Design and Implementation)
  - Задачи на тестване (Testing tasks)
    - Верификацията и интеграция между подсистемите към по-голяма система (Verification and integration into bigger subsystems)
- И двете дейности са еднакво важни за модела

Задачи на програмиране (Development tasks)

1. Спецификация на изиксванията (Requirements specification)
   - Идентицифиране на изисванията на клиента
   - Ясно дефиниране на целта и характеристиките на системата
2. Функционален дизайн на системата (Functional system design)
   - Свързването на функциите и диалоговите прозорци в новата система (Mapping functions and dialogues of the new system)

Задачи на програмиране (Development tasks)

3. Технически дизайн на системата (Technical system design)
   - Дизайн на това как ще се имплементира системата (Designing the implementation of the system)
   - Дефиниране на интерфейсите които системата ще ползва (Defining interfaces to the system environment)
   - Разделяне на системата на по-малки и разбираеми подсистеми (Decomposing the system into smaller understandable subsystems)
     - Системна архитектура (System architecture)

Задачи на програмиране (Development tasks) (3)

4. Специфициране на компонентите (Component specification)
   - Дефиниране на всяка подсистема (Defining each subsystem)
     - Какво трябва да прави всяка задачам вътрешната структура и интерфейси (Task, behavior, inner structure, interfaces)
5. Самото програмиране (Programming)
   - Имплементация на компонентите на съотвения програмен език (Implementing each specified component in a programming language)
     - Модули, юнити и класове (Modules, units, classes)

Задачи на тестването (1)

1. Компонентно или Юнит тестване (Component (unit) test)
   - Верифицира всеки софтуерен компонент (Verifies each software component)
   - Отговаря на въпроса дали се държи адекватно спрямо изяснените спецификации
2. Интеграционно тестване (Integration test)
   - Проверка на няколко компенента заедно в групи (Checks groups of components)
   - Отговаря на въпроса дали работят правилно един с друг според техническия дизайн

Задачи на тестването (2)

3. Системно тестване (System test)
   - Верифицира системата като едно цяло (Verifies the system as a whole)
   - Отговаря на въпроса дали системата отговаря на специфицираните системни изисквания
4. Крайно тестване преди предаване на клиента (Acceptance test)
   - Отговаря на върпоса дали системата отговаря на клиентските изисквания

Нива на абстракция (Levels of Abstraction)

- Тестването следва нивата на абстракция в които се прави програмирането (Testing follows the levels of abstraction of development)
- Най-лесния начин за намиране на дефекти
  - Като се търсят на нивото на абстракция в който са създадени (Unit, Integration, System)
  - Тестването се извършва заедно с разработката на софтуер

Валидация и Верификация (Validation and Verification)

- Създаваме ли правилната система (Are we building the right system?)
  - Дали продуктът(или част от него) решава неговата предназначена задача (Does the product (or a part of it) solve its task?)
  - Този продукт подходящ ли е за предназначената му нужда (Is this product suitable for its intended use?)
- Създаваме ли системата правилно (Are we building the system right?)
  - Посреща ли продукта спецификациите
  - Дали са изпълнени напълно и коректно

Итеративно-инкрементални модели за разработка на софтуер (Iterative-incremental Development Models)

Итеративно-инкрементални модели за разработка на софтуер (Iterative-incremental Development Models)

- Основната идея на този вид методи (The basic idea behind these methods):
  - Създаване на система чрез повтаряне на итерации (цикли) (Develop a system through repeated cycles (iterative))
  - Като системата се разделя на малки порции във времевия период (инкрементални) (Smaller portions at a time/cycle (incremental))
  - Изучаване на системата и учене от грешките си от предишните версии на системата (Learn during development of previous parts or versions of the system)
  - Цикъл на работа тип Планирай – Създавай – Провери – Действай (Plan-do-check-act cycle)

Преимущества

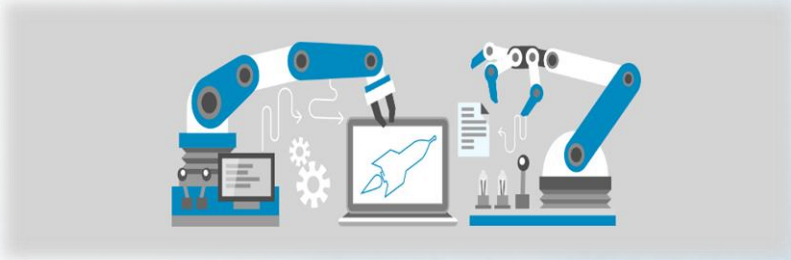
- Системата продобива ценност за бизнеса още в началото на развитието на софтуера (Produces business value early in the development life cycle)
- Верификация и Валифация (Verification and validation)
  - Могат да се извършват в всеки един цикъл
  - Възможност за подобраване на процеса благодарение на придобитите знания (Take advantage of what was learnt)
- Актуалнизация на продукта при всяка итерация (Product update at each iteration)

Недостатъци

- Подтиква да се старира писането на код без да се има ясна предстаава какво трябва да се направи (Tempts you to start coding before you have a clear idea of what you want to do)
- Изисква се повече участие от страна на клиента (Requires more customer involvement than the linear approaches)
- Разделянето на функциите и характеристиките на продукта може да станат трудна задача (Partitioning the functions and features might be problematic)

Първата стъпка (Initialization Step)

- Създаването на основна версия на приложението (base version of the system)
  - Трябва да се създаде продукт, който потребителя да може да ползва
  - Отразяване на основните проблеми които могат да възникнат
  - Да се направи лесно решение което да е достатъчно лесно за разбиране и имплементиране



Итерацията (Iteration Step)

- Итерационна стъпка включва:
  - Дизайн и имплементация на задача от контолния списък на проекта (Design and implementation of a task from the project control list)
  - Анализ на състоянието на последната актуална версия на системата (Analysis of the current version of the system)
  - Анализ на структурата,разделението в модули,използваемотстта, ефективността, надеждността на системата и покриването на изискванията(Analysis of the structure, modularity, usability, reliability, efficiency & achievement of goals)
- Този анализ е на база обратна връзка от клиента (user feedback) или друг отдел за анализ ако има такъв

Контролен списък за проекта (Project Control List)

- Контролен списък за проекта (Project control list)
  - Основна му цел е да бъде използван като референция за итеративния процес
  - Съдържа списък със задачи, които трябва да се изпълнят:
    - Нови функционалности (features) които трябва да се имплементират
    - Области , в системата които имат нужда от поправка в дизайна
    - Постоянно се променя и допълва на база фазата на анализиране от клиента

Фазите за итеративно разработване на софтуер (Iterative Development Phases) (1)

- Процеса на разработка се разделя на парчета (slices) според тяхната функционалност (Incremental development slices the system functionality into increments (portions))
- Всеки инкремент има фази:
  - Начална (Inception)
    - Определея какъв ще е обхватът ,рисковете, изискванията (функционални и нефункционални) на проекта на високо ниво (Identifies project scope, risks, and requirements (functional and non-functional) at a high level)
  - Разработване (Elaboration)
    - Създавата работеща архитектура която се спрява със рисковете и изпълнява нефункционалните изисквания (Delivers a working architecture that mitigates the top risks and fulfills the non-functional requirements)

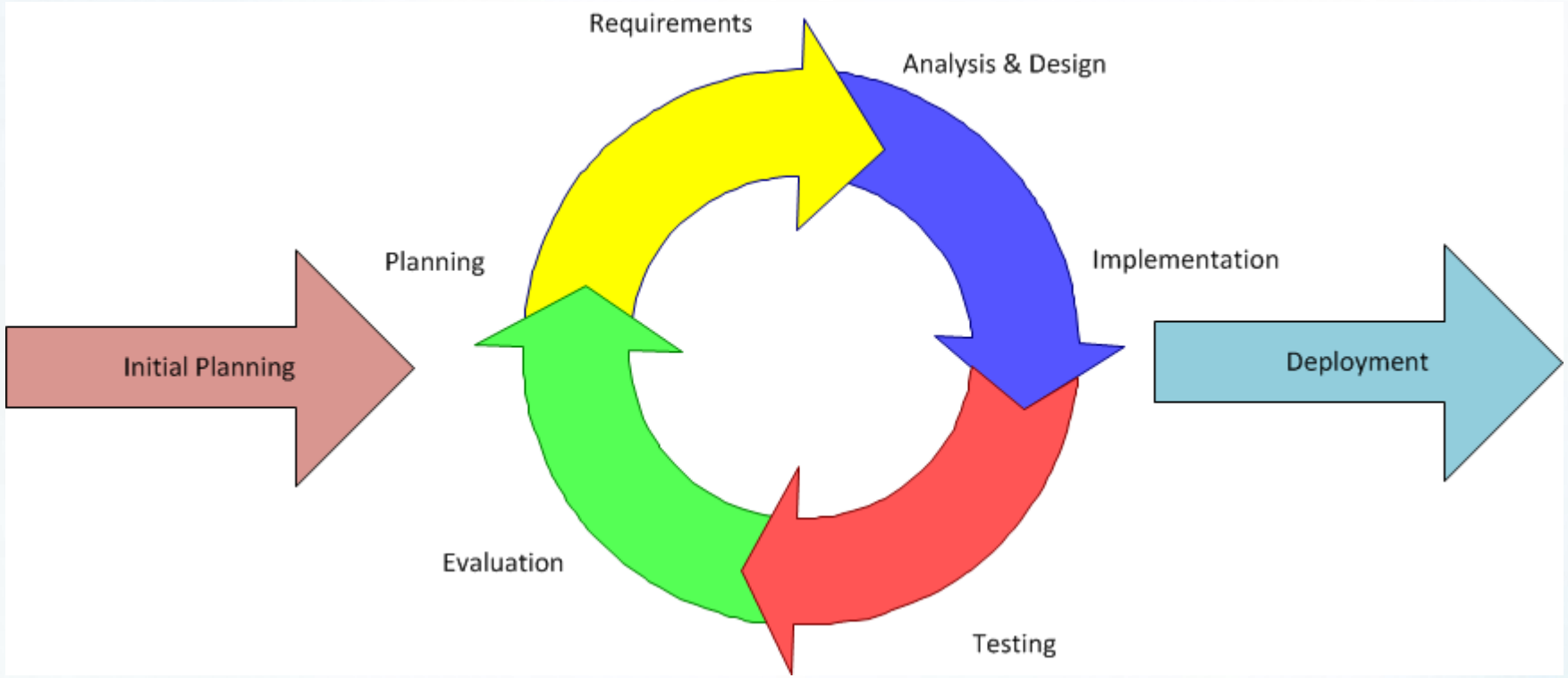
Фазите за итеративно разработване на софтуер (Iterative Development Phases) (2)

- Всеки инкремент включва тези фази:
  - Конструкция (Construction):
    - Постепенно запълва архитектурата с код който е готов за крайния потребител (Incrementally fills-in the architecture with production-ready code)
  - Предаване на продукта (Transition)
    - Представяне на системата на среда където клиента може да я види (Delivers the system into the production operating environment)

Няколко примера за Итеративно-инкрементални модели за разработка на софтуер

- Примери за итеративно-инкрементални модели
  - Agile модел
  - Rapid Application Development (RAD)
  - Rational Unified Process (RUP)

Agile модел за разработка



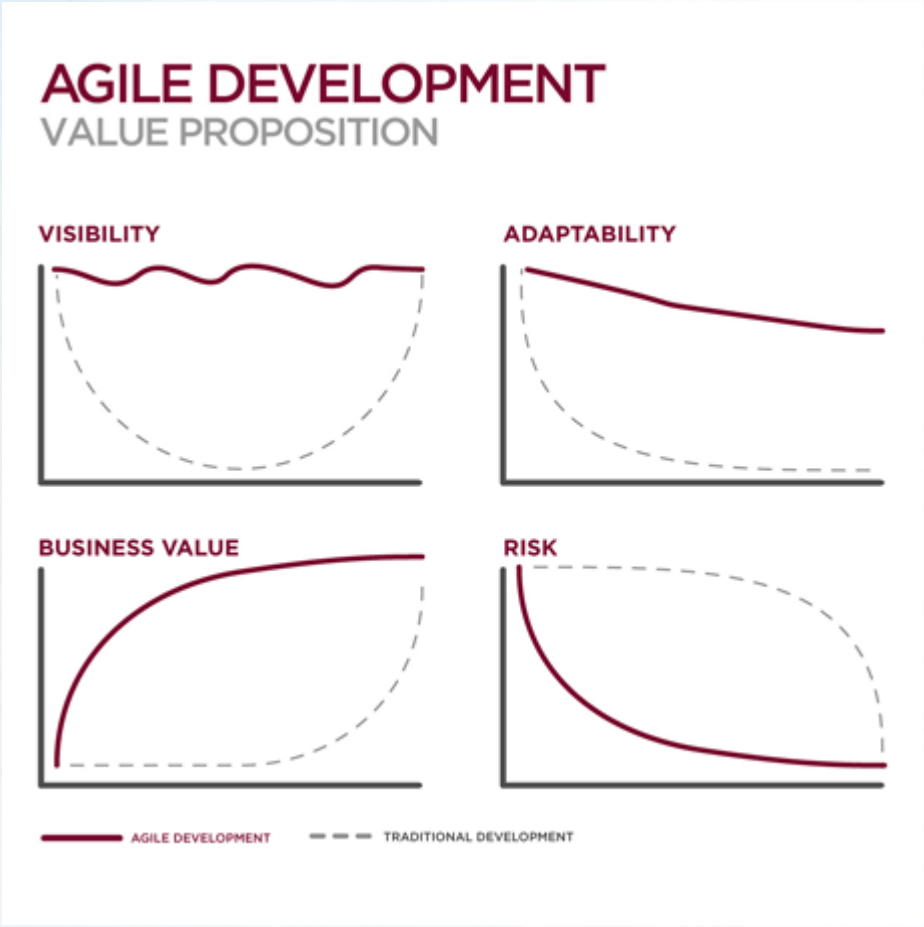
Какво представлява модела Agile

- Какво е Agile разработка на софтуер?
  - Това е структура от методологии за разработка на софтуер, които следват идеологията за итеративно-инкременталния модел за разработка
  - Изисквания и решения се вземат чрез комуникацията между добре организирани екипи със различни функции (Requirements and solutions evolve through collaboration between self-organizing, cross-functional teams)
  - Това е новото има на така наречените „леки“ модели (lightweight model) за разработка
    - Agile манифест който е публикуван през 2001

Ето какво гласи манифестът:

“Our highest priority is to satisfy the customer through early and continuous delivery of valuable software“

Основни преимущества над традициоанните модели



The Agile Manifesto—a statement of values

Individuals and interactions	over	Process and tools
Working software	over	Comprehensive documentation
Customer collaboration	over	Contract negotiation
Responding to change	over	Following a plan

Source: www.agilemanifesto.org

ExelPlus services

12-те принципа определени от Agile Манифеста (1)

- Задоволяване на нуждите на клиента чрез навременно и постоянно предоставяне на желания продукт (Satisfying 'customers' through early and continuous delivery of valuable work)
- Разделяне на големи задачи на по-малки, които могат да се свършат по-бързо (Breaking big work down into smaller components that can be completed quickly.)
- Да се осъзнае че най-добрата работа се върши от добре организирани в себе си екипи. (Recognizing that the best work emerges from self-organizing teams.)
- Предоставяне на хора които са ентусиазирани да вършат своята работа със всичко което им е необходимо за да бъде тя качествена (Providing motivated individuals with the environment and support they need and trust them to get the job done.)
- Създаване на процеси, които да гарантират стабилни усилия и резултати (Creating processes that promote sustainable efforts.)
- Поддържане на еднакво темпо за изпълнение на работата (Maintaining a constant pace for completed work.)

12-те принципа определени от Agile Манифеста (2)

- Да сме добронамерени към промените в изисванията дори и в късни етапи на проекта
- Всекидневна организиране на екипа за проекта по време на целия проект
- На определени интервали от време да се правят промени в насока подобряване ефективността на екипа
- Прогреса се определея според свършената работа (Measuring progress by the amount of completed work.)
- Желание за все по-добро преставяне (Continually seeking excellence.)
- Да се приемат промените в насока, която би довела до конкурентно преимущество

Характеристики на модела Agile (1)

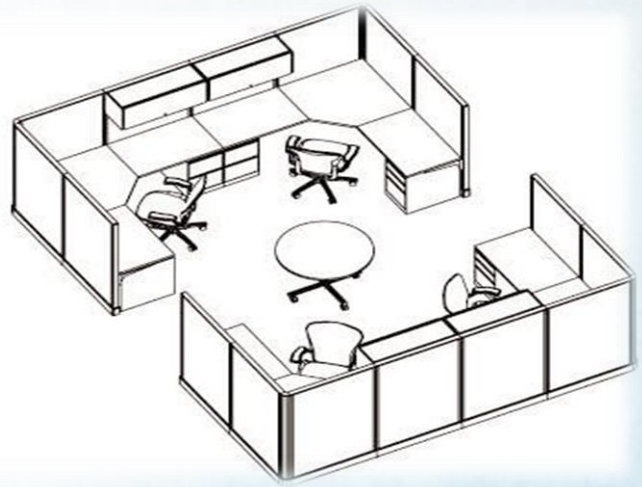
- Задачите са резбиват на малки парчета, които са по лесни за планиране (Tasks are being broken into small increments with minimal planning)
  - Не се налага постоянно дългосрочно планиране (Do not directly involve long-term planning)
  - Итерациите са малки времеви интервали (Iterations are short time frames (time boxes))
    - Периодите са в повечето случаи между седмица и един месец (Typically last one to four weeks)
- Всяка итерация включва екип който работи по цялостен процес за разработка на софтуер(Each iteration involves a team working through a full software development cycle)
  - Програмирено по двойки(E.g. pair programming)

Характеристики на модела Agile (2)

- Възможност за реализиране на продукт на края на всяка итерация с минимално количество дефекти (Available release (with minimal bugs) at the end of each iteration)
- Екипите са :
  - Cross-functional
  - Self-organizing
  - Team members take initiative and responsibility

Характеристики на модела Agile (3)

- Комуникация на живо спрямо записване на документи на хартия (Face-to-face communication over written documents)
  - Повечето екипи, които работят под Agile методологията са в отворен офис наречен bullpen
  - Малък брой на членовете в екипа (5 до 9)
  - Използване на видеоконцеренция,
  - е-мейл както и телефонни конференции за
  - екипи които не са в една и съща локация

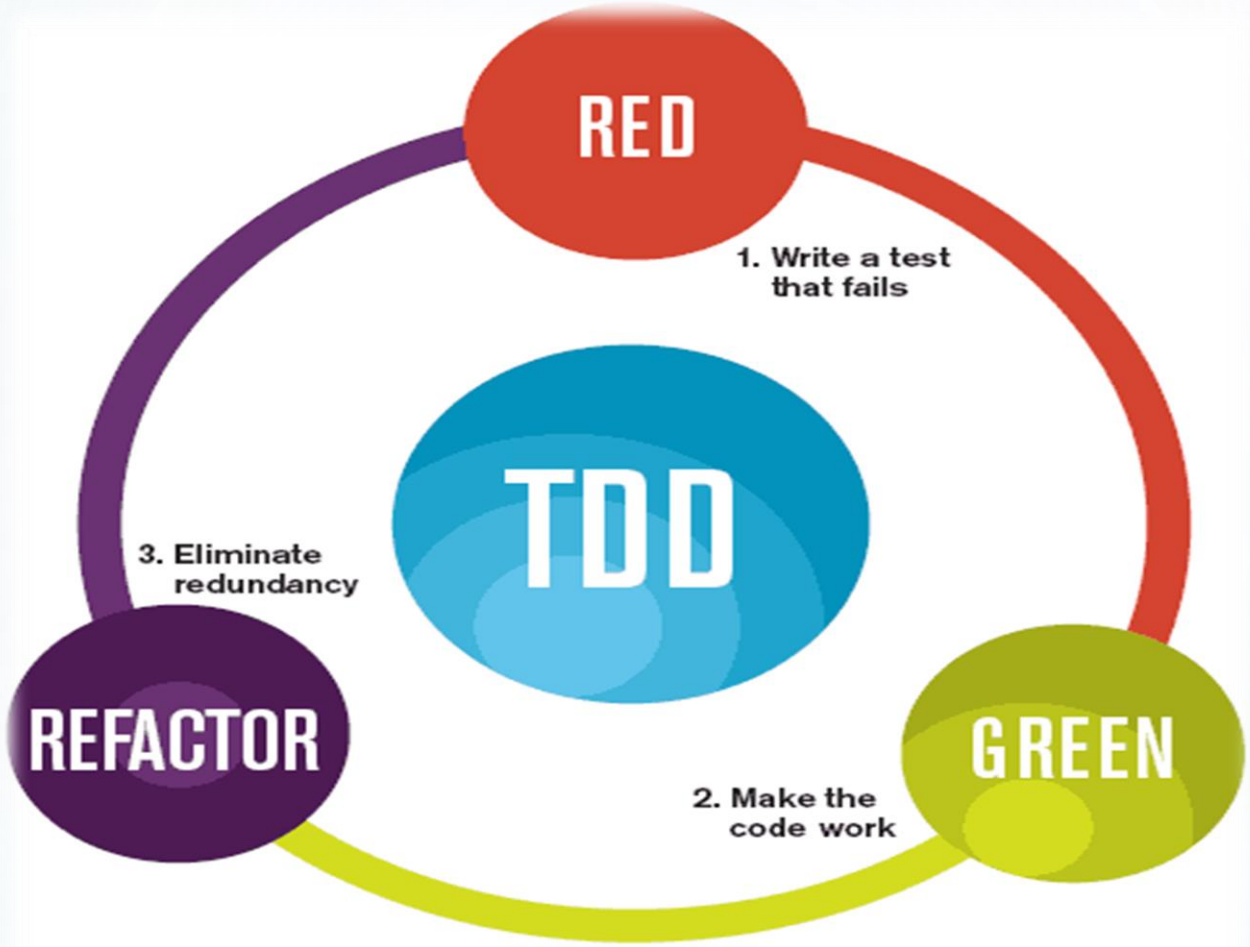


Характеристики на модела Agile (4)

- Екипите които работят под методологията на Agile имат техен представител пред клиента
  - Той е отоговорен да отговаря на върпоси и проблеми възникнали по средата на итерацията
  - Участва в края на всеки прериод в ревюирането на прогреса до момента и оценката на приоритетите(Participates in reviewing progress and re-evaluating priorities)
- Кратка всекидневна официална среща между членов екипа (Routine and formal daily face-to-face debrief meetings among team members)

https://youtu.be/TSb6phzTxdI

Разработка основана на тестване (Test Driven Development)



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

Цикълът на разработки от тип Test Driven Development

- Test-driven development (TDD) се основава на повтаряне на кратък цикъл:
  a) Писането на неуспешни (failing) автоматични тестове
     - Определят желано подобрение или нова функционалност
  b) Създаване на код, който да накара теста да мине (pass)
  c) Накрая да се промени и рефакторира кода според изградените стандарти

Преимущества на TDD

- TDD скъсява времето за обратна връзка от процеса на програмиране
- TDD предоставя детайлни спецификации под формата на тестове
- TDD спомага за разработването на висококачествен код
- TDD предоставя ясни доказателства че софтуера наистина рабти
- TDD помага да се изгради такъв дизайн който е чист т.е. създаване на модули и операции които лесно да се извикват и могат да бъдат тествани

Недостатъци на TDD

- Голяма инвестиция във времето. Понякога за по-лесни случай се губят 20% от имплементацията, а за посериозно слуяи дори повече.
- Допълнителна усложненост. Когато се пишат по сложни и комплексни случай става по-трудно за човека който ги съставя.
- Щети върху дизайна. Понякога дизайнът по начало не е ясен и тестове се пишат според него. В последсвтие те стават нерелевантни при промяна в дизайна и искат преправяне

Защо да пише тестове преди кода?

- Задълбочен анализ над изискванията (requirements)
- Задълбочен анализ над дизайнът и използваемостта на модула, който се тества
- Няма време да се пишат в последствие.
- В последствие се пишат много по-малко тестове, дори в някои случаи не се налага да се дописват

Защо да накараме теста да се провали в началото ?

- По този начин сме сигурни че самият тест няма бъгове
- Да сме сигурни че тестваме точно това което трябва
- Когато тества мине, приключваме с разработката
- Ако не може да напишем даден тест, тогава може би:
  - Не разбираме дадено изискване към продукта
  - Имаме дизайн който не може да се раздели на подходяши (testable) задачи
  - Нямаме достатъчно малък проблем или задача за тестване

Защо да рефакторираме?

- Постоянно се подобрява дизайна на кодът
- Премахване на повтаряемост, по-добра четимост на кода и по-лесната му поддръжка.
- Ще се наложи когато настъпи някаква промяна:
  - Изиксванията към продукта, нашето разбиране за тях и други фактори

Agile методологии

- eXtreme Programming (XP)
- Scrum
- Kanban
- Lean

Екстрамно програмиране (Extreme programming)

Extreme Programming

- Extreme Programming (XP)
  - Вид agile модел за разработка на софтуер
  - Предназначено да подобри софтуерното качество и готовността към промените наложени от изискванията на клиента
  - Този модел е привържаник на честите релийзи на софтуер в по-кратни цикли за разработване като времеви период
    - Насочено към подобряване на продуктивността
    - Основни точки където нови изисквания от клиента могат да се добавят

Extreme Programming принципи

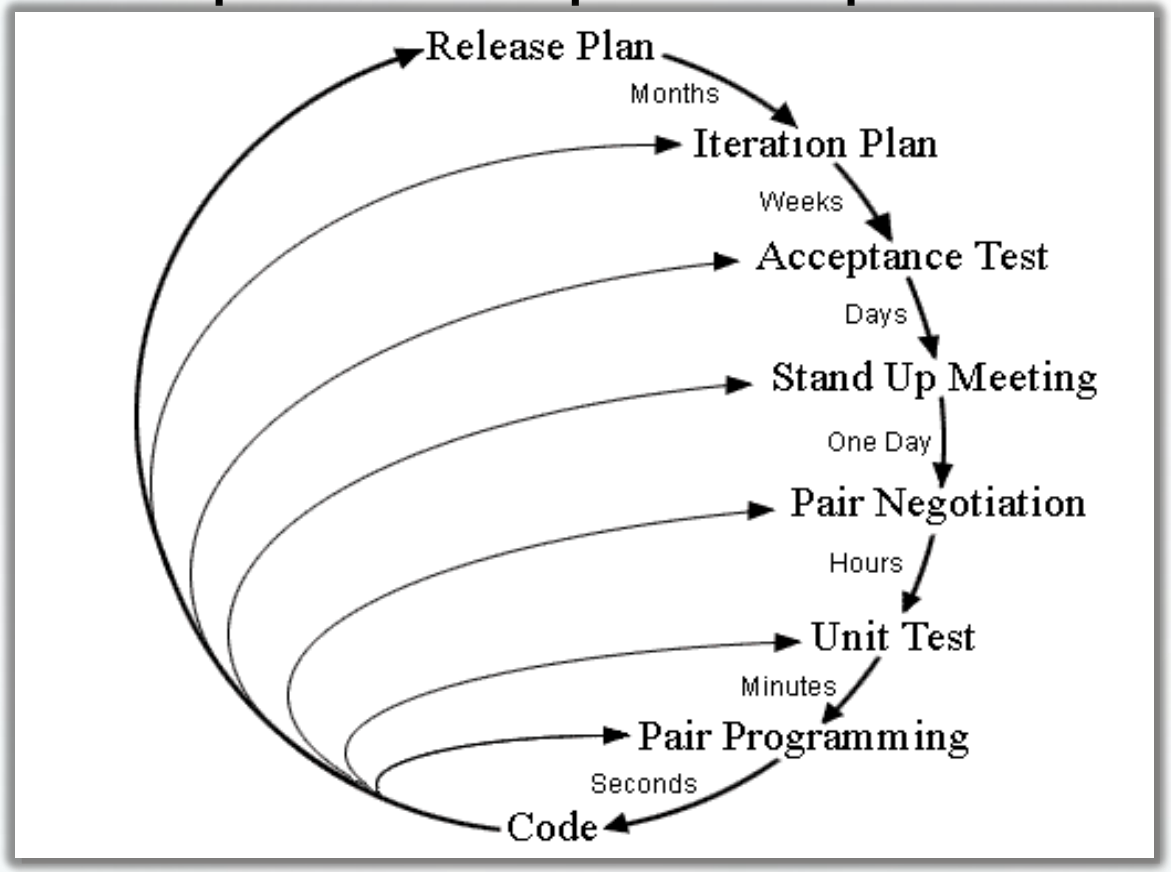
- Extreme Programming подобрява разработката на софтуер по пет основни
  - Комуникация
    - Всеки е част от екипа и комуникацията лице в лице е ежедневие
  - Опростеност
    - Прави се това което се изисква и желае от нас, но не и допълнителни неща

Extreme Programming принципи

- Обратна връзка
  - Ние ще говорим за проекта и ще си адаптираме процеса към него, а не проекта да се адаптира към нашия процес
- Респект
  - Всеки получава и отдава респект според работата на всеки участник в екипа
- Кураж
  - Ще даваме точна и ясна информация свързана с прогреса и предложеното време за работа по проекта

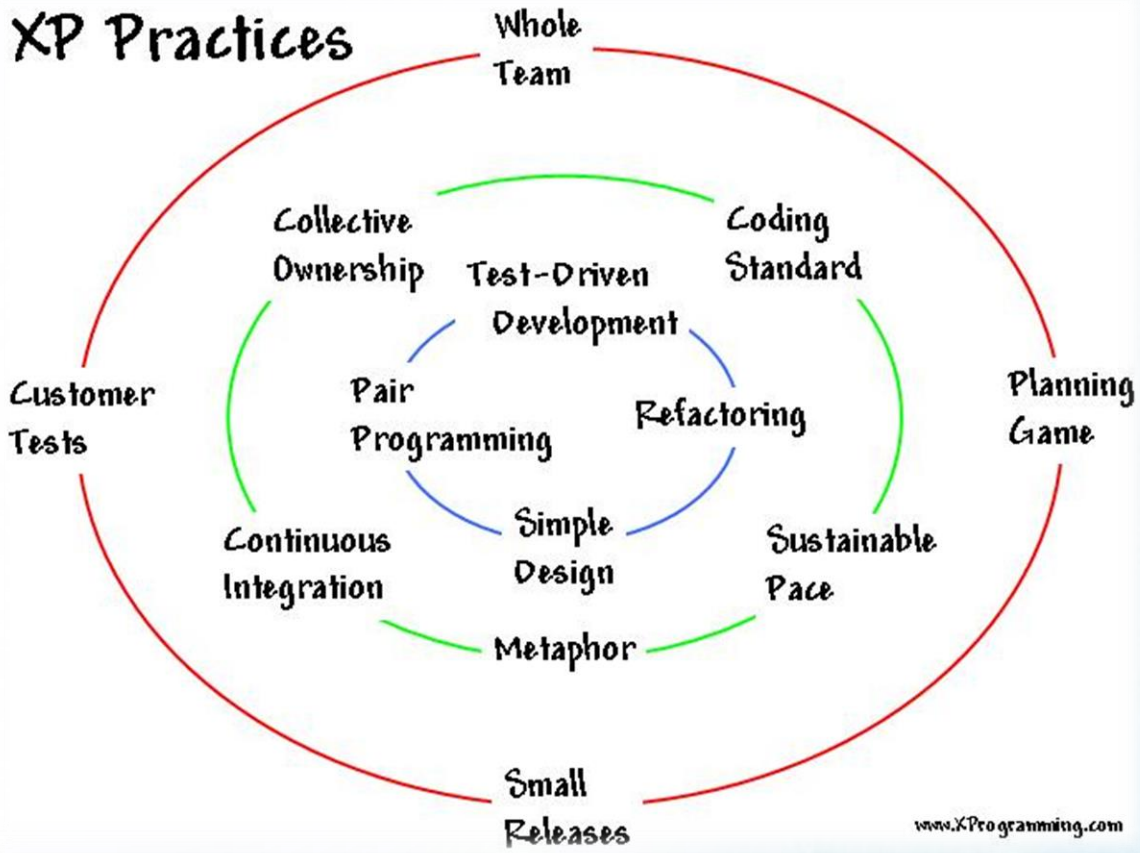
Planning / Feedback Loops

- Цикли на планиране и обратна връзка в XP



Extreme Programming: The 12 Key Practices

- The Planning Game
- Small Releases
- Metaphor
- Simple Design
- Test-Driven Development
- Refactoring
- Pair Programming
- Collective Ownership
- Continuous Integration
- 40-Hour Workweek
- On-site Customer
- Coding Standards



Lean модел на разработка

Lean принципи

- Премахване на загубите
  - Ненужни код и функционалност
  - Забавяния в процеса на софтуерна разработка
  - Неясни изисквания
  - Бюрокрация
  - Бавна комуникация между екипите и структурите

Lean принципи (2)

- Насоченост към нови знания
  - Подобряване на процеса на научаване използвайки кратки итерационни цикли
  - Акумулирането на дефекти се избягва, тъй като се изпълняват тестове още щом кодът е написан
  - Събирането на потребителските изисквания е улеснено
    - Представяне на презентации с реални случаи, от които потребителя може да си избира

Lean принципи (3)

- Решения се вземат възможно най-късно
  - Докато решенията не могат да се вземат базирани на факти
    - Не базирани на прогнози и заключения, които не са подкрепени с факти
  - Колко по-сложна е една система толкова по-голяма възможност за лесни промени трябва да се имплементира в нея

Lean принципи (4)

- Доставка колко се може по-бързо
  - Колкото по-рано е доставен крайният продукт, толкова по-рано може да се върне обратна връзка от клиента
  - Колкото по-кратки са итерациите , толкова по-добра е комуникацията и процеса на научаване в рамките на екипа
  - Скоростта подсигурява задоволяването на сегашните нужди на клиента
    - Не това което те са искали от нашия софтуер вчера

Lean принципи (5)

- Дава свобода на екипа
  - “Find good people and let them do their own job"
  - Мениджърите се научават да изслушват разработчиците
  - Хората не са само ресурси
    - Хората имат нужда от мотивация и някаква ясна представя и идея заради която да работят

Lean принципи (6)

- Създаване на една интегрирана система
  - Integrity означава че компонентите на дадена система работят добре и като цяло
  - Рефакторирането е един добър начин за добре интегрирана архитектура

Lean принципи(7)

- Гледане на системата по-мащабно, като едно цяло
  - Софтуерните системи не са само сбор от тяхните парчета и части , но и резултат от взаимодействието между тях.

Scrum - Прогрес чрез серия от Спринтове

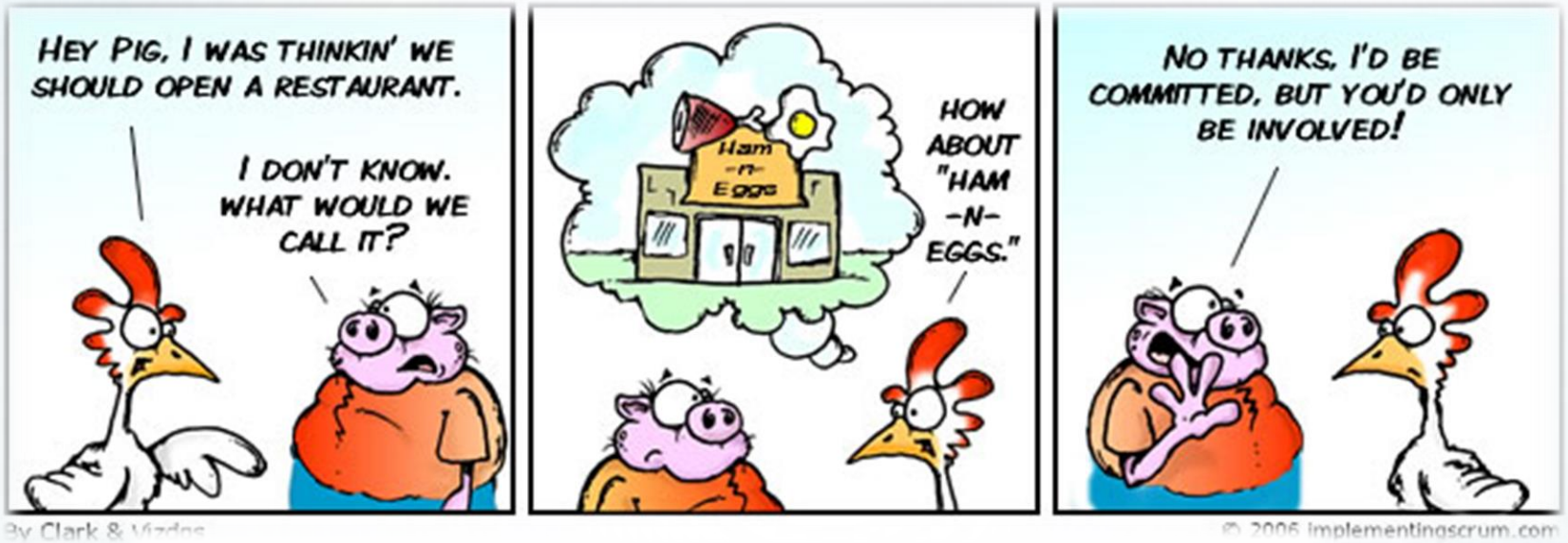
Scrum

- Scrum е интеративно-инкрементален подход за управление на сложни проекти
- Процес от тип agile , който позволява фокусиране върху доставката на най-висока бизнес стойност за възможно най-кратко време
- Позволява бързо и постоянно да се проверява софтуера, който раоти (всеки две седмици до всеки месец)

Scrum (2)

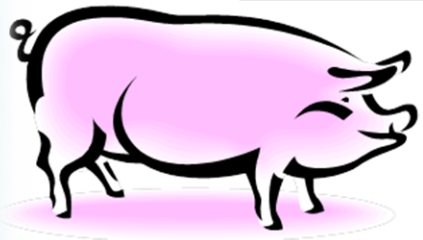
- Бизнеса определя приоритетите
- Екипите се самоорганизират, за да определят най-добрия начин за доставка на най-приоритетните подобренията и новостите (features)
- Всеки две седмици до месец всеки може да види работещ софтуер
  - И възможността да решим дали да го доставим в този вид, или да го доусъвършенстваме с още един спринт

Pigs and Chickens (1)



ROLES

Core roles - Pigs



- Product Owner
- Scrum Master
- Scrum Team

Ancillary roles - Chickens

- Users
- Stakeholders
- Managers

Pigs and Chickens (Кокошки и прасета)

- Scrum екипите съдържат три основни роли и набор от допълните такива
  - Основните роли са още наречени прасета (Pigs)
    - Те са напълно отдадени на проекта
  - Допълнителните или помощни роли са още наречени пилета (chickens)
    - Нямат формална роля и участие в проекта
  - Идва от историята за пилето и прасето

Scrum основни роли

- Основните роли в Scrum екипите са тези, които са напълно отдадени на проекта и на Scrum процеса
  - Те са тези които създават продукта (основната причина за създаването на проекта)
- Scrum основни роли:
  - Scrum Master – поддържа и определя Scrum процесите
  - Product Owner – представлява бизнеса (stakeholders)
  - Team – екип който включва около 7 човека
  - Manager – поддържа визията за продукта ясна

Scrum Master

- Scrum-а се поддържа и имплементира от Scrum Master
- The Scrum Master не е team leader
- Играе ролята на буфер между екипа и всякакви външни инстанции, с който той взимодейства

The Scrum Master

- Отговорен за премахване на препятствия
- Определя и следи за спазването на правилата
- Защитава екипа и ги фокусира върху задачите, които са им поставени
- Също се определя като лидер, който работи за своя екип и му помага (servant-leader)

Екипът

- Екипът е отговорен за доставянето на продукта
- Обикновенно съдържа 5 до 9 човека
  - Участниците в екипа са с различни функции
  - Екипът върши съществената и реална работа
    - Анализиране, дизайн, разработка, тестване, документира, комуникация и др.

Екипът (2)

- Състава на екипа може да се променя след края на даден спринт
- За предпочитане е екипа да се самоорганизира и дисциплинира
  - Въпреки че често работи с екипни лидери (team leader) или мениджъри на проекта (project manager)

The Product Owner

- The Product Owner представлява “гласа” на клиента
  - Той е отговорен за това екипа да достави допълнителна стойност към проекта и бизнеса
- Пише реални сценарии според потреблението на продукта (user stories)
  - Приоритиза ги и ги добавя в продуктовия списък (product backlog)

The Product Owner (2)

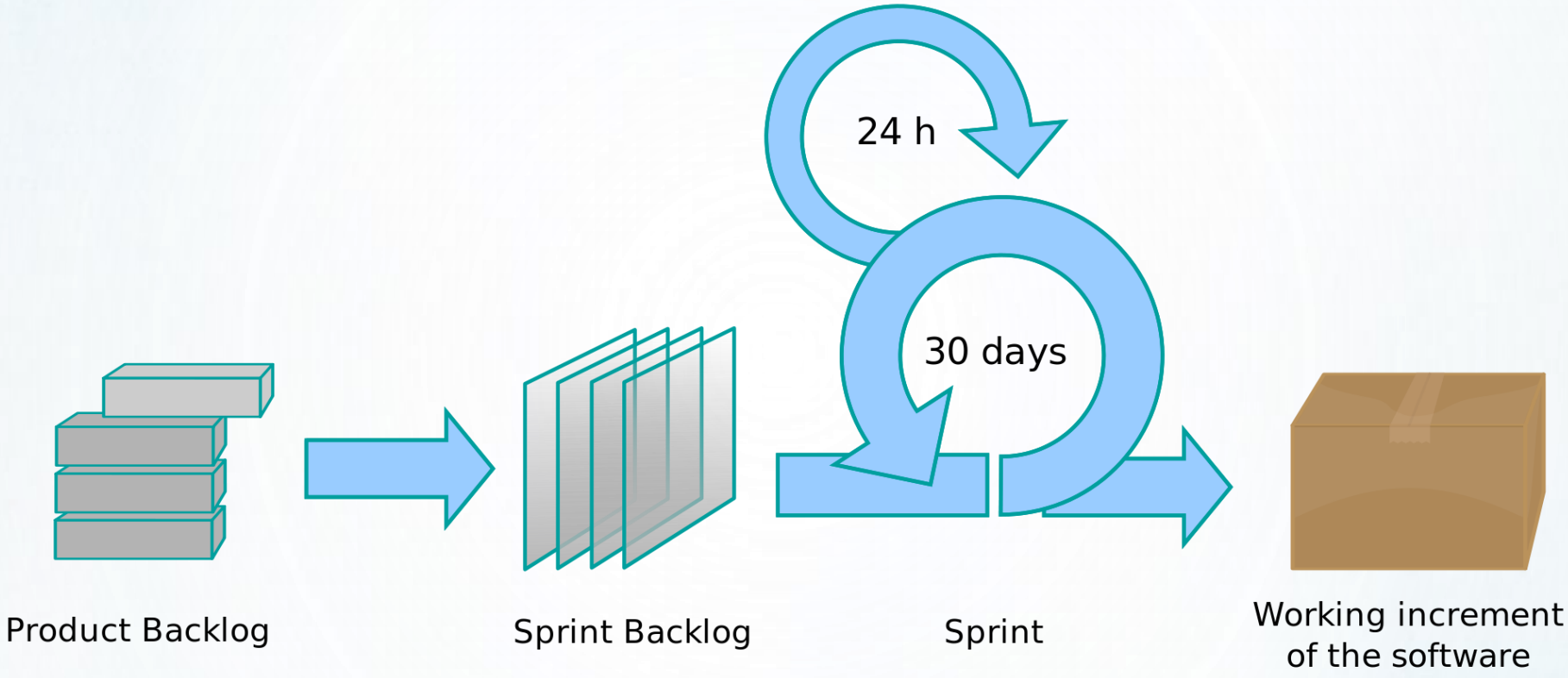
- Приема и отхвърля резултатите от работата
- Scrum екипите трябва да имат един Product Owner
  - Product owner може да е част от екипа за разработка
  - Тази роля не трябва да се изпълнява или да бъде комбинирана с тази на Scrum Master

Scrum допълнителни (Ancillary) Roles

- Stakeholders (клиенти)
  - Хората, които са основната причина за проекта
  - За тях проекта ще създаде продукт, който отговаря на техните изисквания и дали проекта е оправдан
  - Участват пряко само в процеса на прегледа на спринта (sprint reviews)

Scrum Ancillary Roles (2)

- Мениджъри(включително и Project Managers)
  - Хората които създават средата за разработка на продукта
  - Отговорни да следят за ясната визия на продукта
  - Помагайка за всекидневните проблеми, които не зависят пряко от екипа
  - Предлат инструменти и решения
  - Често това са Scrum Master


24 h
30 days
Product Backlog
Sprint Backlog
Sprint
Working increment
of the software

Scrum терминология

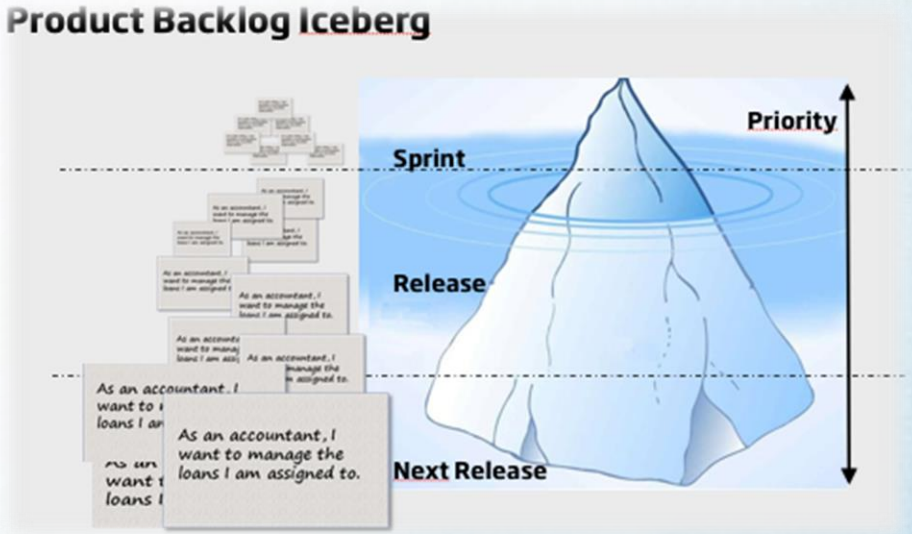
- Спринт/Итерация (Sprint/Iteration)
  - Итерация в разработка от тип Scrum
  - Обикновено две или четири седмици
- Списък с подобрения (Backlog)
  - Всички подобрения (features) които трябва да се разработят във формат User Story
- Графика на прогноза и реална работа Burn down chart (Release and Sprint)

Продуктов списък (Product Backlog)

- Списък на високо ниво с описание на всичките потенциални желания или подобрения за продукта
- Поддържа се през целия проект
- Приоритизиран от product owner
- Отворен и с възможност

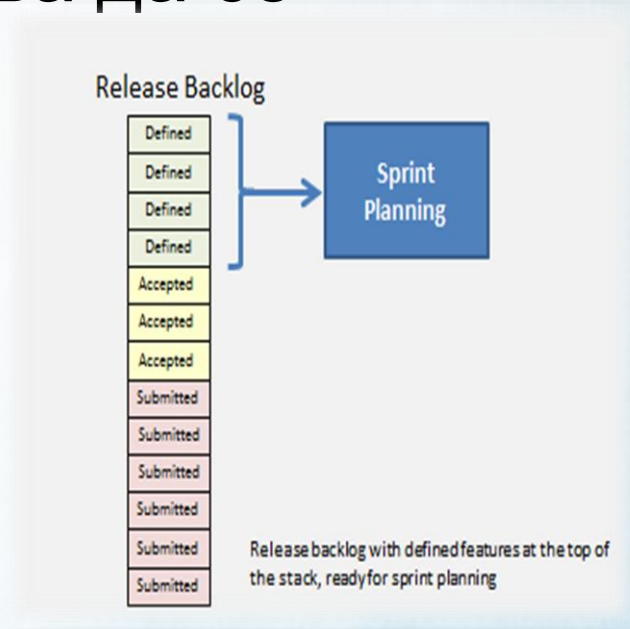
да бъде променян

- Поддържа се често с години



Release Backlog

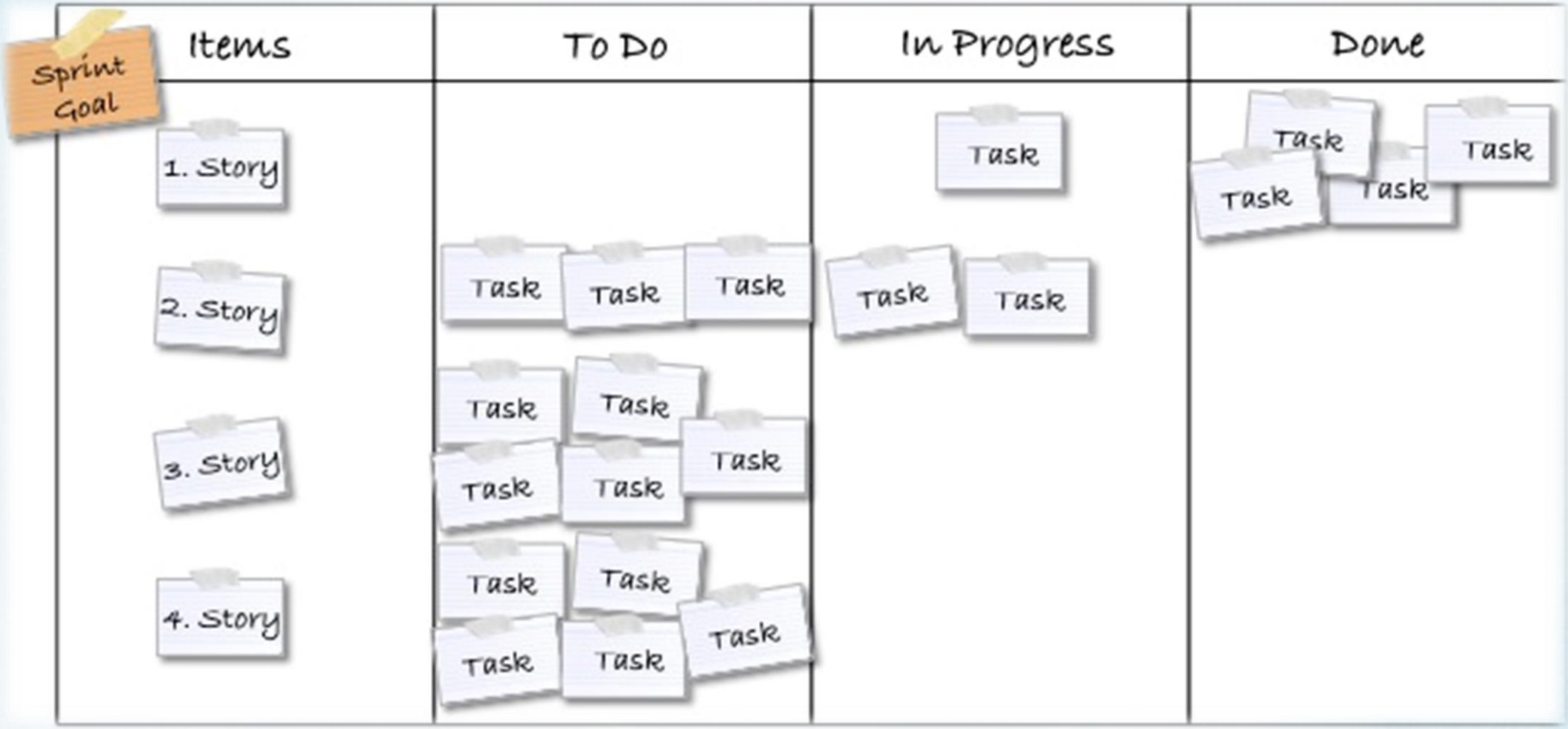
- Ограничн брой от части на Product Backlog, които трябва да се разработят за даден релийз
- Фокусиран на определени цели, които трябва да се изпълнят за определено време
- Приоритизиран от product owner
- Всички части се прогнозират и планират на високо ниво
- Поддръжа се в рамките на месеци



Sprint Backlog

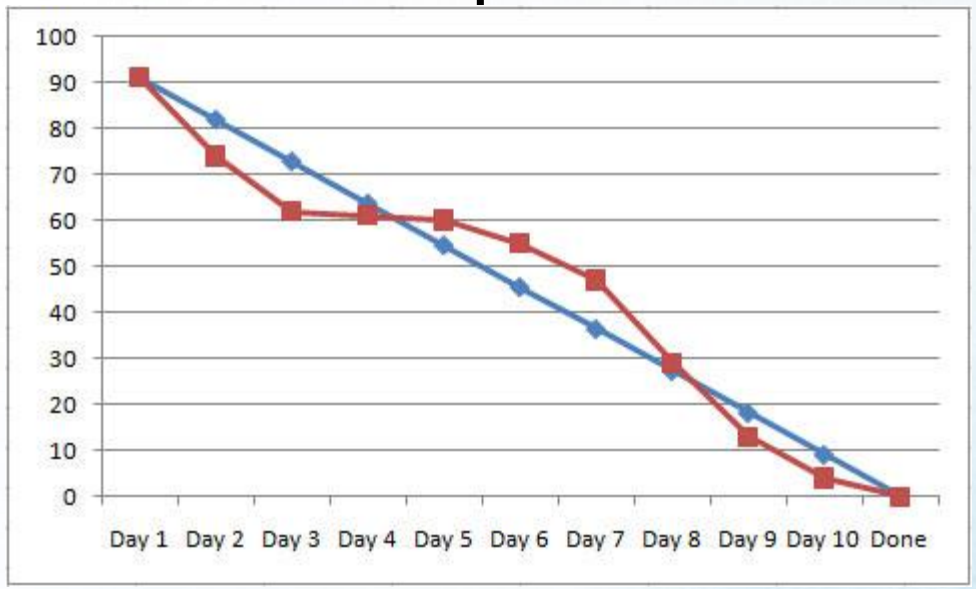
- Списъка от работа която екипът трябва да свърши в рамките на следващия спринт
- Новите подобрения се разбиват на по-малки задачи
  - Които обикновенно са между 4 и 16 часа работа
- Задачите от sprint backlog никога не се задават на определен човек
- Обикновенно важи за няколко седмици
- Съчетан с допълнителна таблица или дъска със задачи за да се виждат промените в състоянието на задачите
  - С етикети като “to do”, “in progress” and “done”

Sprint Backlog (2)



Sprint Burn Down Chart

- Sprint burn down chart е публично достъпна графика
  - Показва оставащата работа по sprint backlog
- Като се актуализира всеки ден, тя дава лесен начин за оценка на прогреса до момента от спринта



Bug Backlog

- Тип списък който включва само дефекти (bugs)
- Може да бъде част от Product Backlog
- Бъговете са описани с техния приоритет
- Включва бъгове намерени от клиентите
- Бъговете са приоритизирани от Product Owner (може и от QA lead)

Team Backlog

- Създаден от екипа
- Описва оставащата работа, която екипа трябва да свърши
- Това е списък с нещата, които са желателни да се случат, но не се заделя специално време за тях
- Елементите от списъка може да бъдат прогнозирани в бъдеще (може и да не бъдат)
- Само един човек е отговорен за него Product Owner на екипа

Scrum практики

- Sprint Planning среща
  - В началото на sprint cycle
  - От нея се определя Sprint backlog
- Daily Scrum stand-up среща
  - Всеки ден по време на спринта – отчита се статуса на проекта от всеки участник от екипа
  - Лимитиран до 15 минути
- Sprint Review среща
  - Преглед на свършената/несвършената работа

Sprint Planning среща

- Провежда се в началото на sprint cycle (всеки 7-30 дни)
- Екипът избира части от product backlog които те определят че могат да свършат
- Sprint backlog се създава
  - Задачите се определят и всеки се планира за определено време (1-16 часа)
  - Създава се съвместно
    - Не единодушно от Scrum Master

Sprint Planning среща(2)

- Ограничен е до 8 часа
  - (Първите четири часа) Product Owner + Екипът:
    - През този период се приоритизира т.н. Product Backlog
  - (Следващите четири часа) само Екипът:
    - Отбелязване на план за спринта, която създава т.н. Sprint Backlog

Ежедневна Scrum среща

- Провежда се всеки ден по време на спринта
  - Също се казва ежедневен standup
- Тази среща има специфични насоки:
  - Срещата започва винаги навреме
  - Всички може да участват, но обикновенно само основните роли говорят
  - Срещата е в рамките на 15 минути
  - Срещата винаги се провежда на едно и също място по едно и също време

Ежедневна Scrum среща (2)

- Всеки отговаря не 3 въпроса:
  - Какво прави вчера? What did you do yesterday?
  - Какво ще правиш днес? What will you do today?
  - Нещо възпрепятства ли ти работата? Is anything in your way?

Ежедневна Scrum среща (3)

- Scrum срещата помага да се елиминира допълнителната нужда от други ненужни срещи
- Ако даден проблем има нужда от разискване – това може да стане след самата среща

Sprint Review/Demo среща

- Преглед на работата която е свършена и тази която не е
- Представяне на свършената работа на клиента (stakeholders) (още т.н.“the demo”)
  - Нови подобрения и промени по архитектурата
  - Работа която не е сврършена не може да се демонстрира
  - Златно правило е тя да се проведе за около 2 часа
  - Никакви слайдове от презентация, само демонстрация
- Лимит от четити часа

Sprint ретроспекция

- Всички участници в екипа разглеждат какво се е случило в предходния спринт
- Правят се постоянни подобрения в процеса
- Два основни въпроса се задават:
  - Какво мина добре по време на спринта?
  - Какво може да се подобри в следващия спринт?
- Ограничен до три часа

https://youtu.be/PXrgS4R6cy4

Полезни линкове

- https://www.youtube.com/watch?v=ACcfYB_5iV8&list=PLebXUHgaZIX59v5YBgxRYCV7batgaCN2G
- http://scrummethodology.com/
- https://en.wikipedia.org/wiki/Waterfall_model
- http://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm
- http://www.seguetech.com/blog/2013/07/05/waterfall-vs-agile-right-development-methodology


?
Q
&
&
A
!