ЖИЗНЕН ЦИКЪЛ НА СОФТУЕРНИЯ ПРОДУКТ

Съдържание

- Жизнен цикъл – основни понятия
- Модели на жизнения цикъл
  - Каскаден
  - Инкрементален
  - Спирален
  - RUP
  - Гъвкави (agile)

Основни понятия

- **Жизнен цикъл** (ЖЦ) на програмния продукт (ПП): период на създаване и използване на ПП.
- **Начало**: възникване на идеята за създаване
- **Край**: преустановяване на използването на последното копие на ПП
- **Софтуерен процес** (СП): набор от дейности, необходими за разработката на ПП.
- **Модел на СП**: абстрактно представяне на софтуерния процес. Дефинира набор от дейности, задачи, ключови резултати и отчетни материали, необходими за изграждането на висококачествен софтуер.
- **Фаза**: отрязък от време, през което се извършват определени дейности по разработвания ПП.

Основни понятия - RUP

- **Фаза**
  - **Inception** –Планиране
  - **Elaboration** –Детайлизиране
  - **Construction** –Изграждане
  - **Transition** –Предаване
- **Итерация –** в рамките на фаза
- **Use Case -** описание на последователността от действия, включващи вариантите, които една система може да изпълни така, че да се получи определен резултат за актьора; поток от събития.

Основни етапи от разработката на софтуер

- Анализиране на изискванията
- Проектиране (Дизайн) на системата
- Имплементация
- Тестване

Requirements engineering

- *Изискванията (software requirements)* описват системата, която трябва да се разработи
  - Отговарят на въпроса "какво?", а не "как?"
  - Функционални и нефункционални изисквания
- Много е трудно изискванията да се опишат и документират изчерпателно и еднозначно
  - Изискванията винаги се променят по време на работа по проекта
  - Начална визия и изисквания и постепенно разширение и уточняване
- Добрата спецификация минимизира бъдещите промени
  - Спестява време и пари за целия проект

Проектиране (Дизайн)

- Софтуерният дизайн описва как системата ще изпълни изискванията
- Архитектурният дизайн описва:
  - Как задачата ще бъде разбита на подзадачи (модули)
  - Отговорностите на отделните модули
  - Взаимодействията между модулите
  - Интерфейсите за връзка
- Детайлен дизайн
  - структура и организация на всеки от модулите в детайли
- Обектно-ориентиран дизайн
  - класове и обекти, техните отговорности, как си взаимодействат
- Вътрешният дизайн на класовете описва как работи всеки клас

Имплементация

- ***Имплементация*** - процес на създаване на програмния код
- Кодът стриктно следва дизайна
  - Мит - писането на кода е най-важната част при разработката на СП
- Основните решения се вземат по време на анализиране на изискванията и по време на дизайна

Тестване

- ***Тестването*** проверява дали даденото решение изпълнява всички изисквания
- Целта на тестването е да намери дефекти (грешки)
  - Black-box и white-box тестове
  - Unit тестове, системни тестове
  - ...

Жизнен цикъл - основни дейности

- Въпреки наличието на различни модели на СП, основните дейности, с които се характеризира всеки от тях са:
  - **Специфициране**: дефиниране на функционални и нефункционални изисквания.
  - **Проектиране и реализация**: Изграждане на ПП в съответствие със специфицираните изисквания.
  - **Валидиране**: Тестване на ПП, за да се удостовери, че отговаря на изискванията на потребителя.
  - **Развитие**: Развитие на ПП в съответствие с променящите се нужди на потребителя.

Модели на жизнения цикъл

1	**Пълни**
1.1	**Едномерни**
1.1.1	Хронологични
1.1.2	Функционални
1.2	**Многомерни**
1.2.1	Двумерен
1.2.2	Тримерен
1.3	**Циклични**
1.3.1	Инкрементален
1.3.2	Спирален
1.4	**Комбинирани**
1.4.1	Rational Unified Process
2	**Частични**
3	Гъвкави

Хронологични (последователни) модели

- **Фази на разработка от гледна точка на потребителя**:
  - Специфициране на изискванията и анализ
  - Проектиране
  - Реализация на ПП
  - Поддръжка
- **Построени са на хронологичен принцип**, като се предполага, че:
  - Бизнес проблемът, който се решава, може напълно да бъде разбран и описан преди дизайна на решението.
  - Дизайн, който удовлетворява всички аспекти на проблема, може да бъде специфициран преди реализацията.
  - Реализацията може да се извърши преди валидиране и предаване.

Каскаден модел

- **Дефиниране на изискванията**: Функциите, ограниченията и целите на системата се определят съвместно с потребителите. Следва детайлното им дефиниране, което се използва като спецификация на системата.
- **Системен и софтуерен дизайн:** проектиране на инфраструктурната и софтуерната архитектура съгласно функционалните и нефункционални изисквания. Изграждане на обща системна архитектура. Детайлно идентифициране и описание на основните елементи на ПП и връзките между тях.
- **Имплементация (Реализация) и unit тестване**: реализация на софтуерния дизайн като набор от програми или програмни units.

Каскаден модел

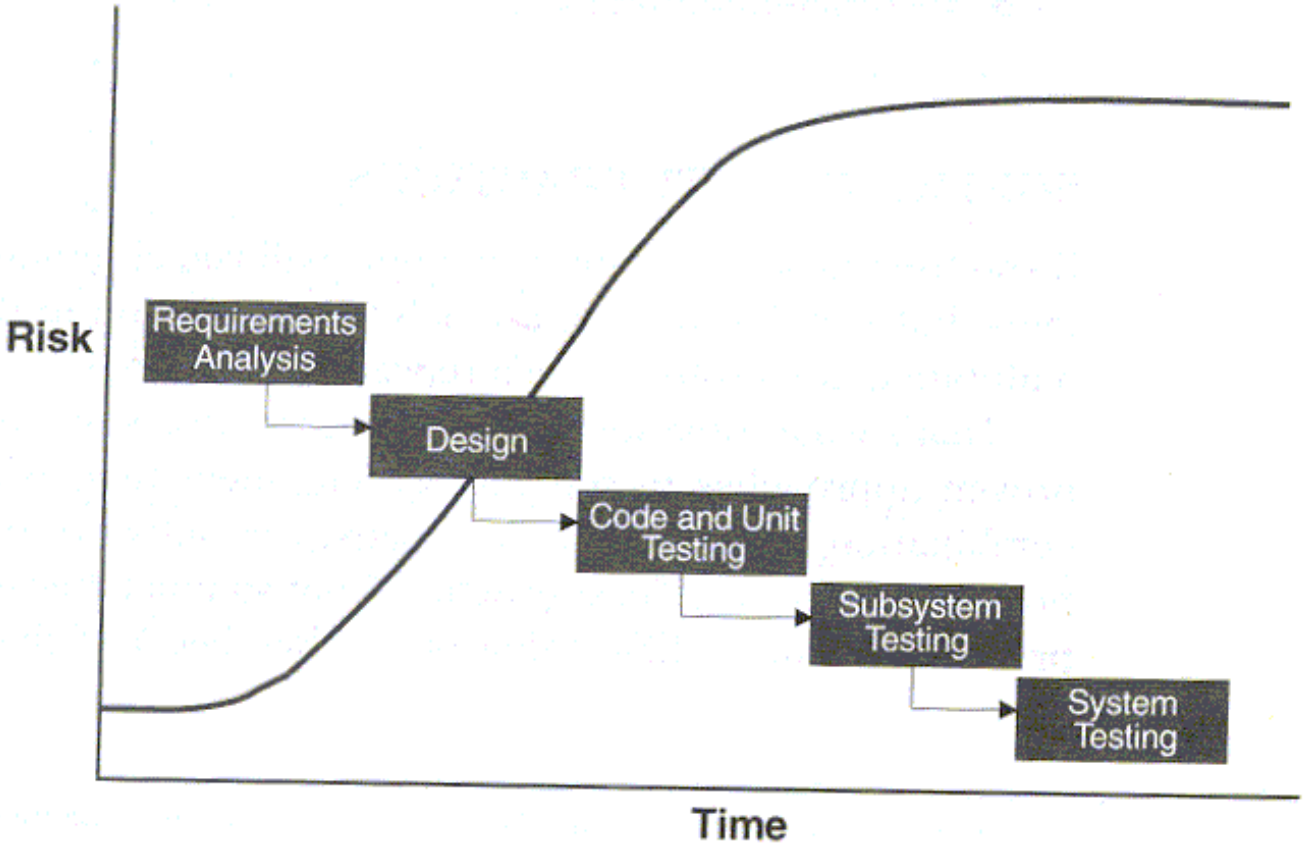
- **Интеграционно и системно тестване**: отделните компоненти се интегрират и тестват като цяла система, за да се провери, че са изпълнени функционалните и нефункционалните изисквания.
  - **Експлоатация и поддръжка**: използване на системата. Поддръжката включва отстраняване на грешки, които не са били открити в по-ранни фази и усъвършенстване и разширяване на системата при възникване на нови изисквания.

Каскаден модел – предимства и недостатъци

- **Приложимост**: Опитни разработчици работят в добре позната предметна област с позната технология за кратък период на разработката, като например поддръжка след въвеждане в експлоатация.
- **Предимства**: Лесно се планира и се следи напредъка по изпълнение на дейностите по разработка на ПП.
- **Проблеми**:
  - Често потребителите се затрудняват да формулират ясно всичките си изисквания;
  - Изискванията към ПП се променят в процеса на разработката;
  - Недостатъчно участие на потребителите;
  - Работеща версия на системата се представя на късен етап;
  - Късно откриване на грешки допуснати на ранен етап;
  - Риск от блокиране на работата по проекта, заради изчакване завършването на свързани дейности.

Хронологични модели - недостатъци

Последователните методи увеличават риска с времето

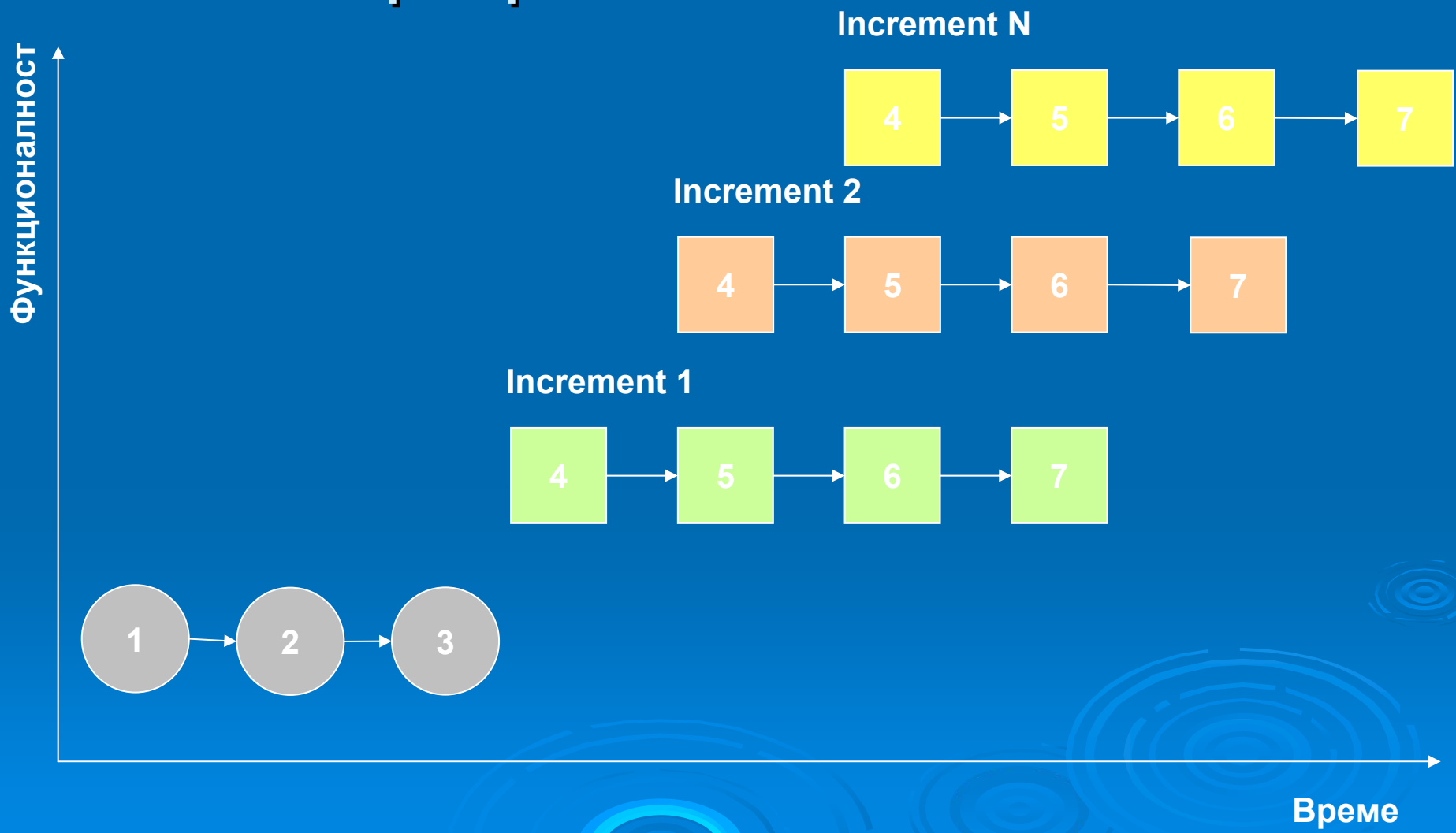


The waterfall lifecycle

Инкрементален модел

- Разработването и предаването на системата е разделено на отделни версии, наречени increments. Всяка версия реализира част от изискваната функционалност. Всеки increment разширява обхвата на реализираната функционалност на системата.
- Потребителските изисквания се подреждат по приоритет и разработката следва приоритета
- След идентифициране на отделните increments, изискванията за първия increment се детайлизират.
- След стартиране разработката на един increment, изискванията за него се замразяват, докато изискванията за по-късните increments продължават да се развиват

Инкрементален модел – графично описание



Инкрементален модел – предимства и недостатъци

- **Приложимост**:
  - Полезен в случаите, когато не е сформиран екип за изпълнение на целия проект. Ранните increments могат да се реализират от по-малък екип.
- **Предимства**:
  - С всеки increment се предоставя част от функционалността на системата – така тя е достъпна на ранен етап
  - Ранните increments играят ролята на прототипи и подпомагат детайлизирането на изискванията за по-късните increments
  - По-лесно управление на риска
  - Функционалността с най-голям приоритет се тества най-много

Инкрементален модел – предимства и недостатъци

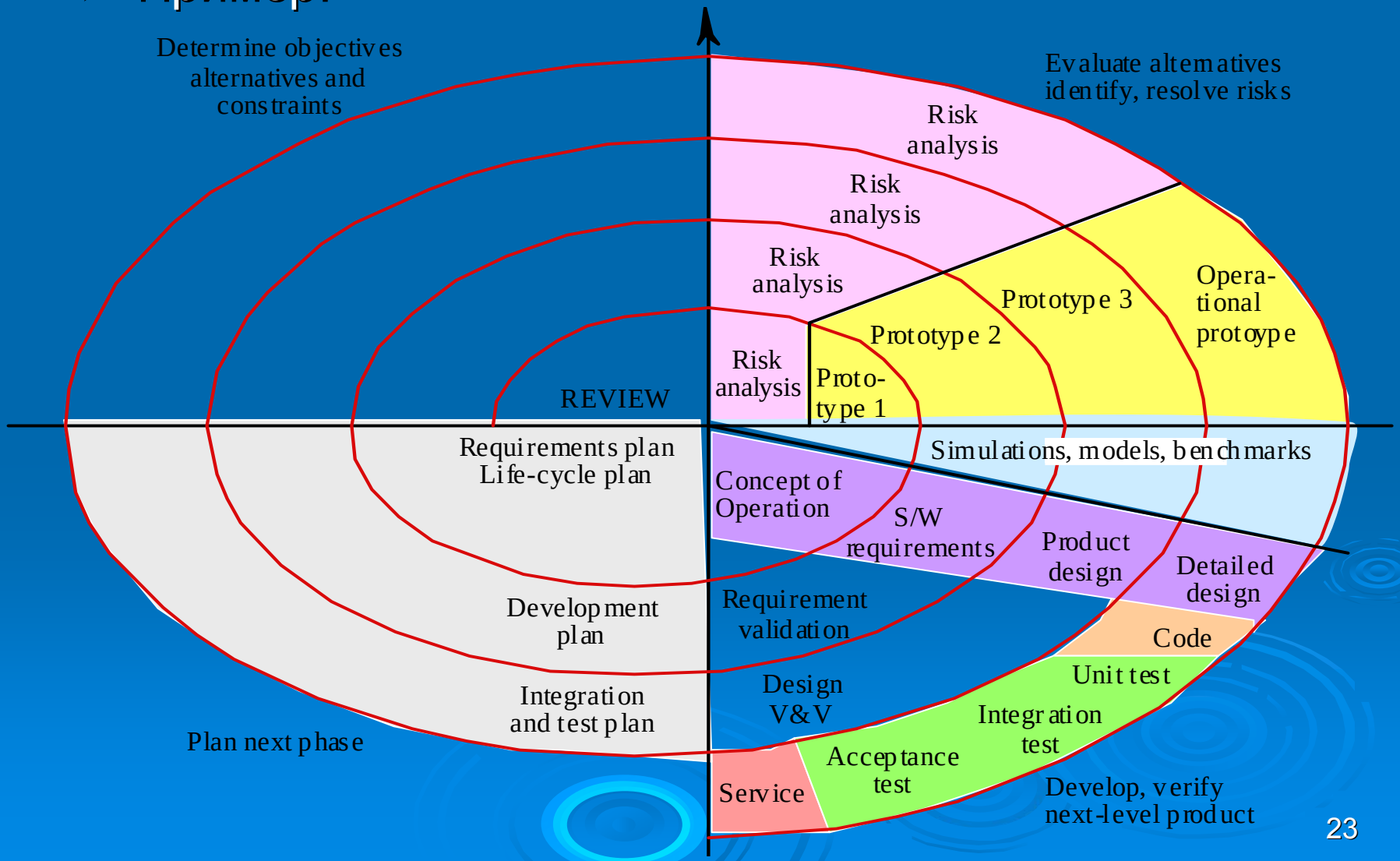
- **Проблеми**:
  - Трудности при разделянето на increments на база потребителските изисквания, тъй като increments трябва да са сравнително малки и всеки трябва да реализира завършена функционалност.
  - Всяка система изисква базова функционалност, която се ползва от отделните ѝ части. Тъй като изискванията не са детайлизирани преди реализацията на един increment, е трудно да се идентифицират общите услуги, от които се нуждаят всички increments.

Спирален модел

- Обединява еволюционния подход с изискванията на линейния хронологичен модел свързани с управление и контрол.
- Разработката се движи спираловидно като серия от инкрементални версии (!= последователност от дейности с връщане назад).
- Спиралният модел е разделен на определен брой рамкови дейности:
  - Връзка с потребителя
  - Планиране
  - Анализ на риска
  - Проектиране
  - Реализация и предаване
  - Оценка от потребителя

Спирален модел

- Пример:



Спирален модел

- Всяка обиколка представлява една фаза в процеса на разработка.
- Сектори на спиралния модел
  - Дефиниране на целите: определят се специфичните цели на фазата
  - Оценка на риска и редукция: оценяват се рисковете и се определят действия за редуциране на основните рискове
  - Разработка и валидиране: избира се модел за разработка на системата (може да е всеки от общите модели)
  - Планиране: Прави се преглед на резултатите по проекта и се планира следващата фаза на спиралата

Спирален модел – предимства и недостатъци

- **Приложимост**:
  - Разработване на големи системи
- **Предимства**:
  - Оценка на риска на всяка фаза
  - Разработка на прототип за намаляване на риска на всяко ниво
  - Функционалността с най-голям приоритет се тества най-много
- **Проблеми**:
  - Трудности при планиране на ресурсите
  - Трудности за управление и контрол
  - Промяна на обхвата
  - Необходим специализиран опит за управление на риска
  - Реализацията на модела може да струва скъпо
  - Неприложим за малки проекти

RUP - История

- Развитие на обектно-ориентираните (ОО) методи и езици за програмиране през 80-те. Разнообразие от методи за ОО анализ и дизайн.
- UML (Unified Modeling Language) - създаден през 90-те от J.Rumbaugh, G.Booch, I.Jacobson. Съдържа нотация за моделиране и разработка на ОО системи.
- RUP - архитектурно-ориентиран, итеративен и инкрементален софтуерен процес, проектиран като рамка за ОО софтуерно инженерство с UML. Базира се базира use-case модел за описание на функционалността,

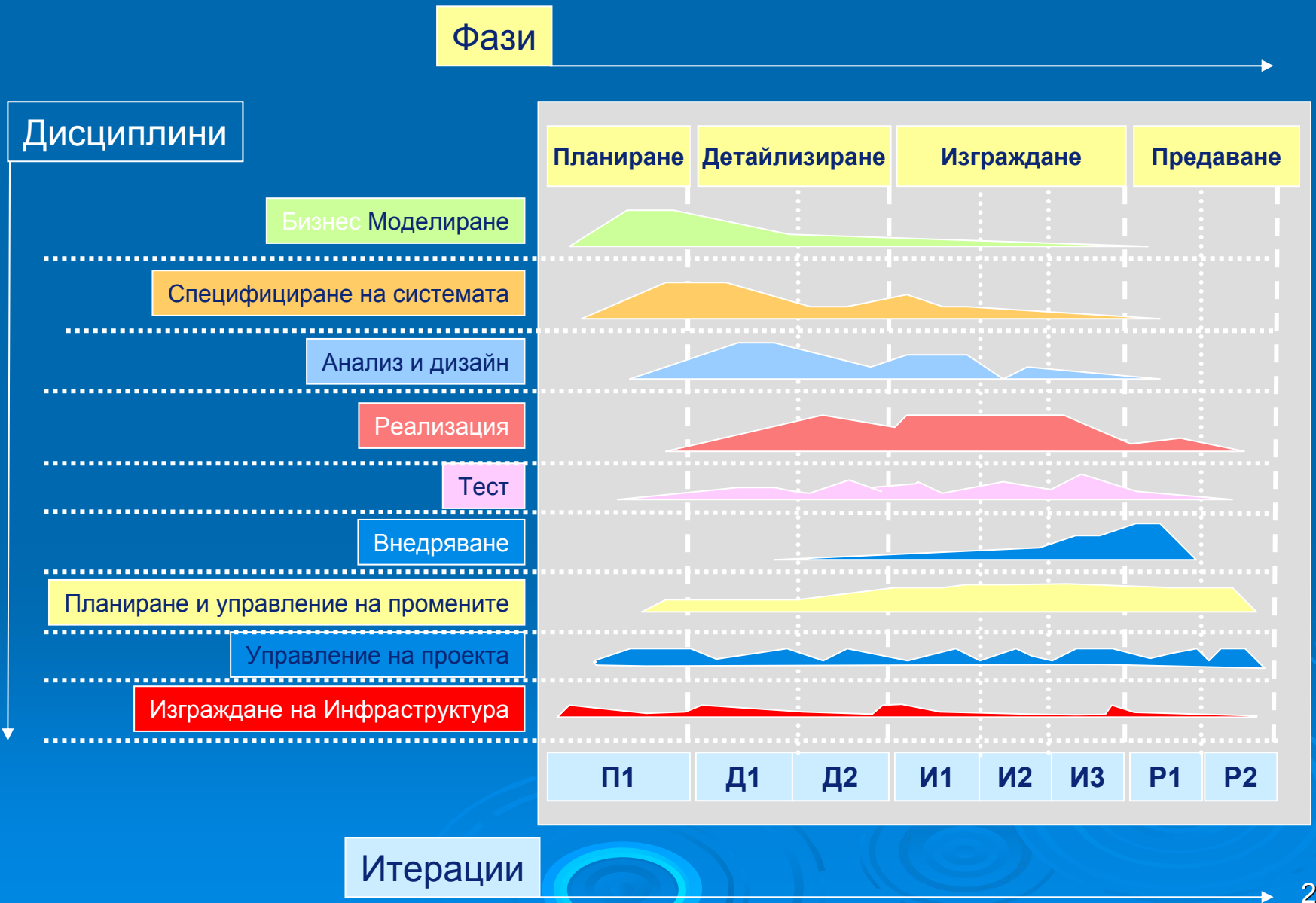
RUP - Фази

- **Фази**:
  - **Планиране (Inception)**
    - Дефиниране на обхвата на проекта – основните бизнес изисквания се формулират чрез набор от use-cases
    - Избор на обща архитектура – определяне на подсистеми
    - Планиране – ресурси, рискове, график
  - **Детайлизиране (Elaboration)**
    - Разширяване и детайлизиране на use-cases
    - Анализ и дизайн
    - Изграждане на архитектура

RUP - Фази

- **Фази**:
  - **Изграждане (Construction)**
    - Завършване на дейностите по анализ и дизайн
    - Разработване на софтуерни компоненти, които реализират отделните use-cases
    - Системни тестове (напр. unit и интеграционни тестове)
  - **Предаване (Transition)**
    - Обучение на потребители
    - Потребителски тестове
    - Отстраняване на грешки

RUP – Фази и Дейности



RUP - Добри практики

- Добри практики на RUP
  - Софтуерът се разработва итеративно
  - Изискванията се управляват
  - Използва се компонентно базирана архитектура
  - Софтуерът се моделира визуално
  - Верифицира се качеството
  - Контролират се промените в софтуера

Гъвкави (Agile) Методи

- Скръм (Scrum)
- Екстремно програмираме - XP (Extreme Programming)
- Adaptive Software Development (ASD)
- Dynamic System Development Method (DSDM)

Основни понятия

- **Роли** : Собственик на продукта (Product Owner), ScrumMaster, Екип (Team)
- **Специфични прояви**: Sprint Planning, Sprint Review, Sprint Retrospective, & Daily Scrum Meeting
- **Артефакти**: продуктов списък (Product Backlog), спринтов списък (Sprint Backlog), and Burndown Chart

Как работи Scrum