

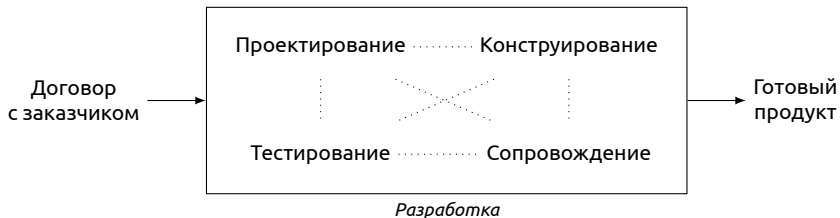
## Стандарт и модели жизненного цикла

Алексей Островский

Физико-технический учебно-научный центр НАН Украины

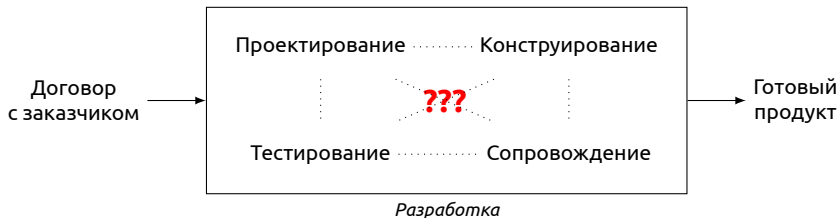
17 октября 2014 г.

## Жизненный цикл



**Жизненный цикл** — схема упорядочивания работ, касающихся проектирования и разработки программного продукта.

## Жизненный цикл

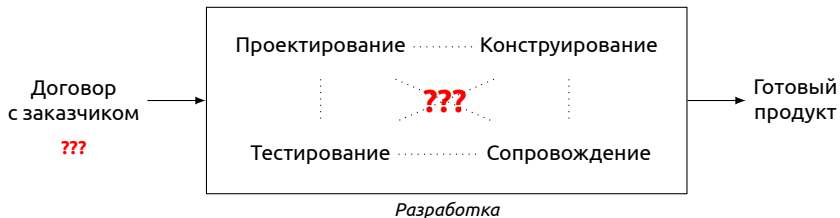


**Жизненный цикл** — схема упорядочивания работ, касающихся проектирования и разработки программного продукта.

### Проблемы:

1. Как соотносятся между собой различные процессы разработки ПО?

## Жизненный цикл

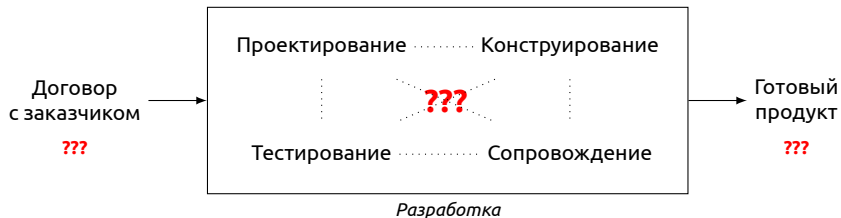


**Жизненный цикл** — схема упорядочивания работ, касающихся проектирования и разработки программного продукта.

### Проблемы:

1. Как соотносятся между собой различные процессы разработки ПО?
2. Каким образом организовано взаимодействие с заказчиком и конечными пользователями?

## Жизненный цикл



**Жизненный цикл** — схема упорядочивания работ, касающихся проектирования и разработки программного продукта.

### Проблемы:

1. Как соотносятся между собой различные процессы разработки ПО?
2. Каким образом организовано взаимодействие с заказчиком и конечными пользователями?
3. Что считается конечным продуктом разработки?

# Стандарт ISO 12207

## Содержание стандарта:

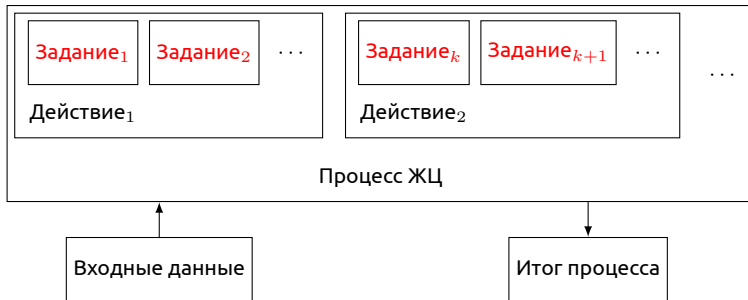
- ▶ 23 процесса разработки;
- ▶ 95 родов деятельности по разработке (англ. *activity*);
- ▶ 325 заданий (англ. *task*);
- ▶ 224 результатов выполнения процессов (англ. *outcome*).

**NB.** Стандарт определяет *составляющие* процессов разработки ПО, но не *последовательность* их выполнения.

## Структура процессов ЖЦ



## Структура процессов ЖЦ



### Определение

**Задание** (англ. *task*) — требование, рекомендация или допустимое действие для достижения определенного итога процесса.



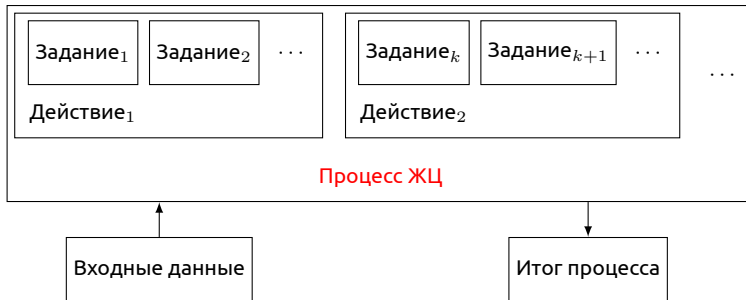
## Структура процессов ЖЦ



### Определение

**Действие** (англ. *activity*) — набор связанных заданий в пределах процесса.

## Структура процессов ЖЦ



### Определение

**Процесс** — набор взаимосвязанных действий, преобразующих поданную на вход информацию.

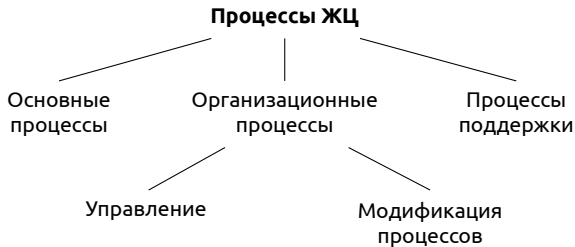
## Структура процессов ЖЦ



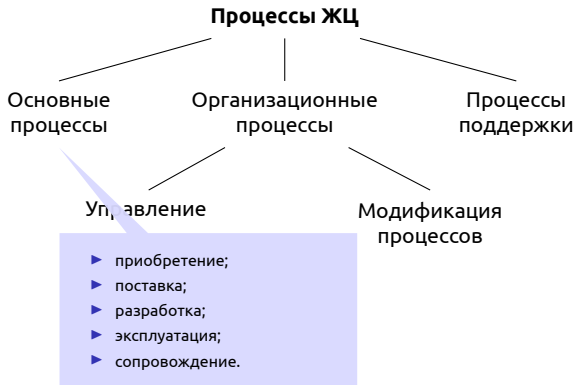
### Определение

**Итог процесса** (англ. *process outcome*) — наблюдаемый результат достижения цели выполнения процесса (программный артефакт, изменение состояния системы, выполнение требования и т. п.).

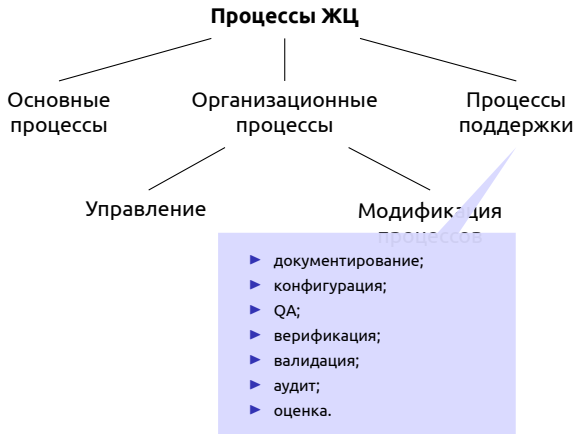
## Классификация процессов



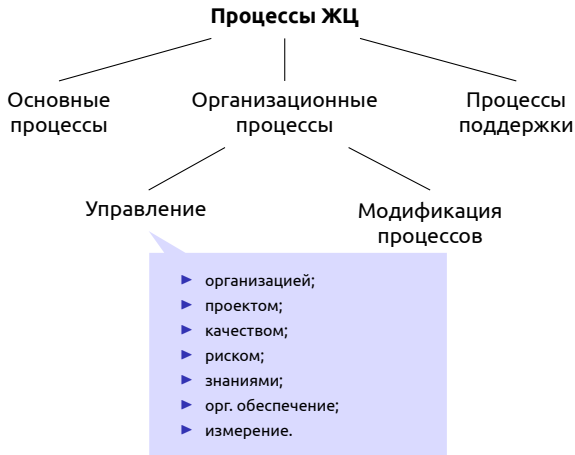
# Классификация процессов



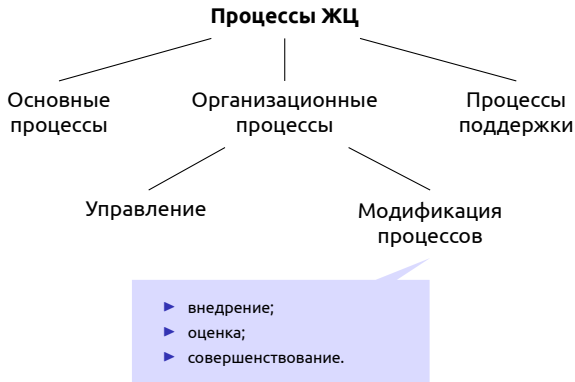
## Классификация процессов



## Классификация процессов



## Классификация процессов





## Основные процессы ЖЦ

1. **Приобретение** (англ. *acquisition*) — начальный процесс ЖЦ, определяющий действия заказчика.

### Составляющие:

- ▶ инициация и подготовка запроса на разработку;
- ▶ оформление и актуализация контракта;
- ▶ приемка ПО.

2. **Поставка** (англ. *supply*) — совместные действия заказчика и разработчика по составлению общего плана управления проектом (*project management plan*).

3. **Разработка** (англ. *development*) — действия разработчика по созданию ПО.

### Составляющие:

- ▶ анализ требований, создание дизайна системы и компонентов;
- ▶ кодирование;
- ▶ модульное, интеграционное и системное тестирование ПО.

## Основные процессы ЖЦ (продолжение)

4. **Эксплуатация** (англ. *operation*) — действия обслуживающей организации, обеспечивающей эксплуатацию системы конечными пользователями.

**Составляющие:**

- ▶ функциональное тестирование;
- ▶ проверка правильности эксплуатации;
- ▶ руководства по использованию.

5. **Сопровождение** (англ. *maintenance*) — действия организации, сопровождающей продукт (управление модификациями, поддержка функциональности, инсталляция и т. п.).

**Составляющие:**

- ▶ анализ вопросов сопровождения и модификации;
- ▶ разработка планов модификации;
- ▶ миграция;
- ▶ вывод из эксплуатации.

# Модели жизненного цикла

## Определение

**Модель жизненного цикла** — это схема выполнения работ и задач в рамках процессов, обеспечивающая разработку, эксплуатацию и сопровождение программного продукта.

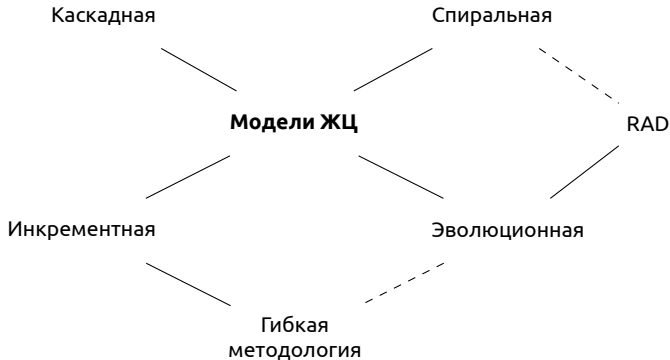
**Составляющие** модели:

- ▶ разработка требований или технического задания;
- ▶ разработка эскизного или технического проекта;
- ▶ программирование и проектирование рабочего проекта;
- ▶ пробная эксплуатация;
- ▶ сопровождение и улучшение;
- ▶ снятие с эксплуатации.

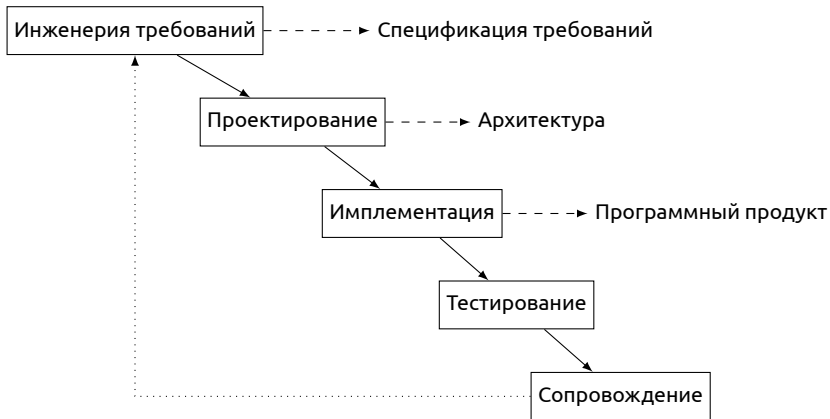
## Цели моделей ЖЦ

- ▶ **Планирование и распределение работ** между разработчиками;
- ▶ **управление проектом** разработки;
- ▶ обеспечение **взаимодействия** между разработчиками и заказчиком;
- ▶ **контроль работ, оценка промежуточных артефактов** ЖЦ на соответствие требованиям;
- ▶ **оценка конечного продукта и затрат** на его получение;
- ▶ **согласование** промежуточных результатов с заказчиком;
- ▶ проверка правильности конечного продукта (**тестирование**), оценка его соответствия требованиям;
- ▶ **усовершенствование процессов** ЖЦ по результатам разработки.

## Классификация моделей ЖЦ



## Каскадная модель



**Каскадная модель** (англ. *waterfall model*) — применение традиционного инженерного подхода к разработке ПО.

## Каскадная модель (продолжение)

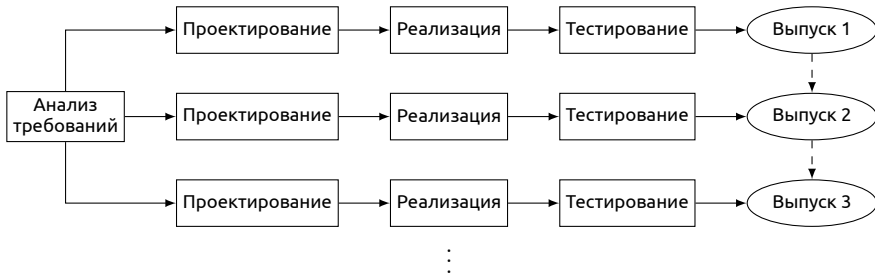
**Основная идея:** значительное внимание уделяется инженерии требований и проектированию, чтобы застраховаться от возможных затратных ошибок.

**Недостатки модели:**

- ▶ жесткое ограничение последовательности действий по разработке;
- ▶ игнорирование меняющихся нужд пользователей, факторов операционной среды, что приводит к изменению требований во время разработки;
- ▶ большой период между внесением и обнаружением ошибки.

**Целесообразность применения:** комплексные системы, для которых долгий цикл сопровождения более важен, чем затраты на разработку.

# Инкрементная модель



**Инкрементная модель** — разработка продукта итерациями, каждая из которых завершается выпуском работоспособной и осмысленной версии.



## Инкрементная модель (продолжение)

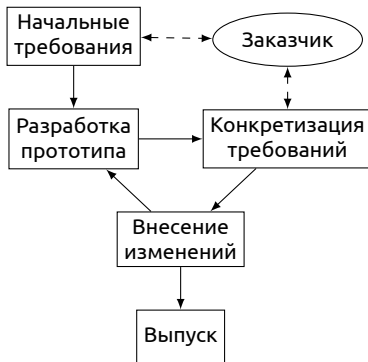
**Основная идея:** последовательное наращивание функциональных возможностей программного продукта с применением на каждой итерации всех процессов каскадной модели.

**Недостатки модели:** требования фиксированы на протяжении всего процесса разработки.

### Целесообразность применения:

- ▶ необходима быстрая реализация возможностей системы;
- ▶ существует декомпозиция системы на составляющие части, реализуемые как самостоятельные промежуточные или готовые продукты;
- ▶ возможно увеличение финансирования на разработку отдельных частей продукта.

## Эволюционная модель



**Эволюционная модель** — разработка ПО с использованием функциональных прототипов, которые *эволюционируют* в элементы конечного продукта.

## Эволюционная модель (продолжение)

### Основные идеи:

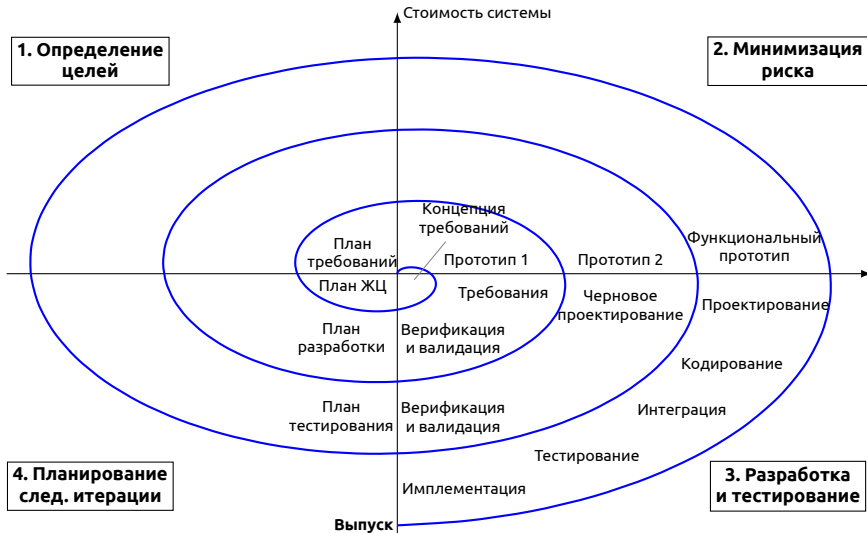
- ▶ создание множества прототипов (т. е. неполных версий) продукта для определения и уточнения требований пользователей;
- ▶ интенсивное использование средств автоматизации: визуальных сред разработки пользовательского интерфейса, СУБД, ЯП высокого уровня абстракции, генераторов кода и т. п.

### Недостатки модели:

- ▶ дополнительные затраты на разработку прототипов;
- ▶ недостаточный анализ системы ( $\Rightarrow$  неоптимальная архитектура);
- ▶ риск интерпретации заказчиком прототипов как финального продукта.

**Целесообразность применения:** проекты, для которых важен пользовательский интерфейс.

# Спиральная модель



## Спиральная модель (продолжение)

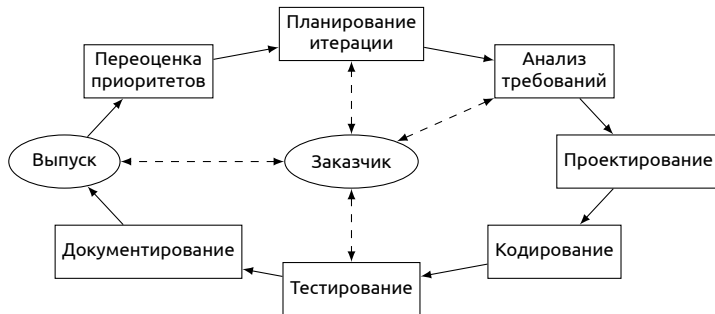
### Основные идеи:

- ▶ контроль риска — раннее тестирование наиболее сложных частей ПО; выбор распределения между работами (анализ требований, проектирование, создание прототипов, тестирование); выбор уровня детализации.
- ▶ выбор модели процесса разработки (каскадная модель, прототипирование) на каждой итерации.

**Недостатки модели:** проблема выбора момента начала новой итерации.

**Целесообразность применения:** комплексные дорогостоящие проекты, для которых критически важна минимизация риска.

## Гибкая методология разработки ПО



- |                                    |   |                                       |
|------------------------------------|---|---------------------------------------|
| <i>Личности и взаимодействие</i>   | > | <i>процессы и инструменты</i>         |
| <i>Работающее ПО</i>               | > | <i>исчерпывающая документация</i>     |
| <i>Сотрудничество с заказчиком</i> | > | <i>согласование условий контракта</i> |
| <i>Реагирование на изменения</i>   | > | <i>следование плану</i>               |

— Agile Manifesto, 2001

## Принципы гибкой методологии

1. **Личности и взаимодействие** — важность самоорганизации и взаимодействий между разработчиками (напр., парное программирование); многофункциональность каждого исполнителя (проектирование, кодирование, тестирование, ...).
2. **Работающий продукт** — работающее ПО лучше отражает процесс разработки для заказчика лучше, чем документы.
3. **Сотрудничество с заказчиком** — доработка и конкретизация требований в процессе разработки; постоянное присутствие представителя заказчика при разработке ПО.
4. **Реагирование на изменения** — фокус на быстрое внедрение изменений и непрерывную разработку (англ. *continuous development*).

## Методы гибкой разработки

- ▶ **Непрерывная интеграция** (англ. *continuous integration*) — частая (несколько раз в день) автоматизированная сборка программного продукта, чтобы выявить интеграционные проблемы.
- ▶ **Проблемно-ориентированное проектирование** (англ. *domain-driven design*) — создание концептуальных моделей предметной области с целью упростить ее понимание разработчиками.
- ▶ **Парное программирование** (англ. *pair programming*) — 1-й разработчик пишет код, 2-й проверяет его на правильность.
- ▶ **Разработка через тестирование** (англ. *test-driven development*) — написание набора тестов, проверяющих функциональность элемента ПО, с последующим кодированием для прохождения этого набора.



## Методы гибкой разработки (продолжение)

- ▶ **Автоматизированное модульное тестирование** (англ. *unit testing*) — немедленная проверка всех изменений, вносимых в код.
- ▶ **Шаблоны проектирования** (англ. *design patterns*) — типовые конструктивные элементы программной системы, задающие взаимодействие нескольких компонентов, а также роли и сферы ответственности исполнителей.
- ▶ **Рефакторинг кода** (англ. *code refactoring*) — преобразование кода без изменения функциональности программной системы с целью создания общей архитектуры системы.

## Характеристики гибкой методологии

### Недостатки модели:

- ▶ неоптимальная архитектура системы вследствие отсутствия требований, фиксированных на протяжении всего процесса разработки;
- ▶ риск снижения качества продукта из-за множества изменений, вносимых без достаточного тестирования и проверки на соответствие общей архитектуре системы.

### Целесообразность применения:

- ▶ программные проекты с необходимостью частых выпусков;
- ▶ ПО с быстро меняющимися требованиями (напр., веб-сервисы).

## Выводы

1. Процессы, согласно которым разрабатывается ПО, описаны в стандарте ISO 12207. В то же время, этот стандарт не содержит последовательность выполнения процессов.
2. Различные подходы к расписанию процессов жизненного цикла ПО представлены в моделях ЖЦ — каскадной, итеративной, эволюционной и спиральной; основными различиями моделей являются их подход к инженерии требований и (а)цикличность процессов ЖЦ.
3. Более современный подход к разработке — гибкая методология программирования (*agile development*), в которой основной фокус делается не на планировании, а на взаимодействии внутри коллектива разработчиков и с заказчиком ПО.

# Материалы



**Лавріщева К. М.**

**Програмна інженерія (підручник).**

К., 2008. — 319 с.



**US Department of Health & Human Services**

**Selecting a Development Approach.**

<http://www.cms.gov/Research-Statistics-Data-and-Systems/.../SelectingDevelopmentApproach.pdf>

(неплохой обзор различных методологий разработки)



**Fowler, Martin**

**The New Methodology.**

<http://martinfowler.com/articles/newMethodology.html>

(описание гибкой методологии разработки ПО)

Спасибо за внимание!