1. Процесс, проект, персонал, продукт
2. Этапы водопадного процесса с точки зрения ГОСТ
   1. Формирование требований к АС.
   2. Обследование объекта и обоснование необходимости создания АС.
   3. Разработка концепции АС.
   4. Техническое задание.
   5. Эскизный проект.
   6. Технический проект.
   7. Рабочая документация.
   8. Ввод в действие.
   9. Сопровождение АС.
3. Для каждого риска определяются три величины: вероятность осуществления риска, ущерб, наносимый проекту данным риском в случае осуществления, оценка стоимости устранения риска. Для всех величин используется одна шкала, например от 1 до 10. Первые два числа вычитаются из 11, третье берется как есть и все полученные числа перемножаются, а результат считается приоритетом риска. Чем меньше число, тем выше приоритет.
4. Внешние входы, внешние выходы, внешние запросы, внутренние логические файлы, внешние логические файлы.
5. 1, 3 – тесты функций

2, 4 – модульные тесты

5 – тесты интерфейсов

6 – интеграционные тесты

7 – регрессионные тесты

8 – системные тесты

9 – тесты удобства и простоты использования (юзабилити)

10 – инсталляционные тесты

11 – приемосдаточные тесты

1. Минимальное кардинальное число – минимальное число связей при ассоциации сущностей
2. Классификатор – графическое представление объекта на диаграмме, содержащее заголовок, состояния (член-данные) и секцию поведения (методы)
3. Стереотип - дополнительное указание на принадлежность описываемого типа данных к некоторому специальному множеству типов данных
4. На диаграммах развертывания UML узел может помещать внутрь другого узла, если это соответствует архитектуре приложения.
5. Между прецедентами (вариантами использования ) может существовать и отображаться обобщение прецедента — стрелка с незакрашенным треугольником (треугольник ставится у более общего прецедента).
6. Отличие include, extend в диаграмме вариантов использования

**include***(стрелки идут от базового варианта)* иллюстрирует **что именно использует базовый вариант для выполнения операции**

Так например - Include - хорошо иллюстрирует ту ситуацию, что **восстановление работоспособности компьютера** неизбежно связано с одним из трёх действий (предположим, что иных вариантов нет):

1. ремонт или замена аппаратных компонентов
2. обнаружение и удаление вируса
3. переустановка системы

Таким образом от вариант использования "восстановить работоспособность ПК" можно уточнить именно с помощью таких вот "включений" - Include . В данном случае вариант использования не выполним без одного из перечисленных выше действий.

В то время как **extend указывает на возможность особенного использования базового варианта *(стрелки идут к базовому варианту от специальных)***

Так , например - если вернуться к ситуации с компьютером - то с помощью extend - можно было бы расширить уже вариант использвоания "обнаружение и удаление вируса " - с помощью опции**"обнаружение и удаление с последующей установкой системы защиты"** - которая, впрочем, не обязательно привлекается -если поставлена задача удаления вируса.

1. Да, может, если имеется ввиду сложное состояние
2. Тип объекта на диаграмме последовательностей обозначается как «: Тип»
3. Процедура разработки архитектуры:
   1. Разбейте систему на замкнутые модули
   2. Сравните со стандартными архитектурами, улучшите декомпозицию
   3. Сделайте выбор среди представленных альтернативных архитектур.
   4. Добавьте к классам, полученным на основе анализа требований, классы, обеспечивающие согласование с выбранной архитектурой. Например, в системе, управляемой событиями, это могут быть классы, контролирующие переходы между состояниями.
   5. Примените существующий каркас и (или) образец проектирования, если найдете полезный.
   6. Распределите классы по пакетам. В идеале должно быть 4-8 пакетов (для больших приложений используйте вложенные пакеты). Каждый пакет должен иметь смысл в контексте приложения.
   7. Удостоверьтесь, что связанность между частями высока (степень взаимосвязей между элементами модуля), а сцепление – низкое (степень взаимодействия модуля с другими модулями).
   8. Рассмотрите возможность добавление façade-класса (объекта) для управления интерфейсами пакетов.
4. Формула Бонэма позволяет оценить трудозатраты и длительность проекта по количеству строк кода.
5. <http://habrahabr.ru/post/137408/>
6. **git revert 1b6d** отменит коммит с заданным хэшем. Откат будет сохранен в виде нового коммита.
7. **git add –p**
8. **git reset --hard HEAD**
9. **git reset [--mixed]**
10. **git reset <file>** - если файл в индексе; git checkout --<file>
11. Команда git status выводит информацию обо всех изменениях,  
    внесенных в дерево директорий проекта по сравнению с последним коммитом рабочей ветки; отдельно выводятся внесенные в индекс и неиндексированные  
    файлы. Использовать ее крайне просто:  
    **git status**  
    Кроме того, git status указывает файлы с неразрешенными конфликтами слияния и файлы, игнорируемые git.
12. **git checkout master**
13. **git log**
14. **Интерпретатор** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Interpreter*) — [поведенческий](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5_%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D1%8B_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) [шаблон проектирования](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), решающий часто встречающуюся, но подверженную изменениям, задачу. Также известен как Little (Small) Language
15. Для регулярных грамматик?
16. Singleton гарантирует, что у [класса](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) есть только один экземпляр, и предоставляет к нему глобальную точку доступа.
17. Недостатки шаблона Singleton:
    1. глобальные объекты могут быть вредны для объектного программирования, в некоторых случаях приводят к созданию немасштабируемого проекта;
    2. усложняет написание модульных тестов и следование [TDD](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D1%87%D0%B5%D1%80%D0%B5%D0%B7_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5).
18. Observer cоздает механизм у класса, который позволяет получать экземпляру объекта этого класса оповещения от других объектов об изменении их состояния, тем самым наблюдая за ними[[2]](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D0%B1%D0%BB%D1%8E%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)#cite_note-autogenerated20130623-1-2).
19. Это означает что связи между субъектами и наблюдателями сведены к минимуму. Таким образом, субъект и наблюдатель могут находиться на разных уровнях абстракции системы и могут использоваться повторно по отдельности.
20. Push model лучше использовать в многопоточных средах.
21. Pull model лучше использовать в однопоточных средах.
22. <http://ionrails.com/2009/09/19/ruby_require-vs-load-vs-include-vs-extend/>
23. <http://ionrails.com/2009/09/19/ruby_require-vs-load-vs-include-vs-extend/>
24. http://habrahabr.ru/post/143483/
25. Abstract Factory предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов
26. ХЗ

**Фабричный метод** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Factory Method* также известен как Виртуальный конструктор ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Virtual Constructor*)) — [порождающий шаблон проектирования](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D0%BE%D0%B6%D0%B4%D0%B0%D1%8E%D1%89%D0%B8%D0%B5_%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D1%8B_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), предоставляющий подклассам интерфейс для создания экземпляров некоторого класса. В момент создания наследники могут определить, какой класс создавать. Иными словами, Фабрика [делегирует](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BB%D0%B5%D0%B3%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) создание объектов наследникам родительского класса. Это позволяет использовать в коде программы не специфические классы, а манипулировать абстрактными объектами на более высоком уровне.

1. **Iterator** — [поведенческий](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5_%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D1%8B_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) [шаблон проектирования](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F). Представляет собой объект, позволяющий получить последовательный доступ к элементам [объекта-агрегата](https://ru.wikipedia.org/wiki/%D0%90%D0%B3%D1%80%D0%B5%D0%B3%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) без использования описаний каждого из агрегированных объектов.

Например, такие элементы как [дерево](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_(%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)), [связанный список](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D1%8F%D0%B7%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D1%81%D0%BF%D0%B8%D1%81%D0%BE%D0%BA), [хэш-таблица](https://ru.wikipedia.org/wiki/%D0%A5%D1%8D%D1%88-%D1%82%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D0%B0) и [массив](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%81%D0%B8%D0%B2_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) могут быть пролистаны (и модифицированы) с помощью объекта *Итератор*.

Перебор элементов выполняется объектом итератора, а не самой коллекцией. Это упрощает интерфейс и реализацию коллекции, а также способствует более логичному [разделению обязанностей](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BE%D1%82%D0%B2%D0%B5%D1%82%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D0%B8).

1. Методы интерфейса begin(), end()
2. Visitor применяется когда необходимо для ряда классов сделать похожую (одну и ту же) операцию.
3. Builder отделяет конструирование сложного объекта от его представления, так что в результате одного и того же процесса конструирования могут получаться разные представления.
4. **Замыкание** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *closure*) в программировании — [функция](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), в теле которой присутствуют ссылки на переменные, объявленные вне тела этой функции в окружающем коде и не являющиеся её параметрами. Говоря другим языком, замыкание — функция, которая ссылается на [свободные переменные](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F) в своём [контексте](https://ru.wikipedia.org/w/index.php?title=%D0%9B%D0%B5%D0%BA%D1%81%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BA%D0%BE%D0%BD%D1%82%D0%B5%D0%BA%D1%81%D1%82&action=edit&redlink=1).

|  |  |
| --- | --- |
| **Посетитель** | |
| *Visitor* | |
| **Тип:** | поведенческий |
| **Назначение:** | не изменяя основного [класса](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), добавить в него новые операции. |
| **Структура:** | [VisitorDiagram.svg](https://commons.wikimedia.org/wiki/File:VisitorDiagram.svg?uselang=ru) |
| **Применяется в случаях:** | когда необходимо для ряда классов сделать похожую (одну и ту же) операцию. |
| **Плюсы:** | * новая функциональность в несколько классов добавляется сразу, не изменяя [код](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4) этих классов; * позволяет получить информацию о типе объекта; * двойная диспетчеризация; * возможность описания своего [алгоритма](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) для каждого типа [объектов](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)). |
| **Минусы:** | * при изменении обслуживаемого класса нужно поменять код у шаблона; * затруднено добавление новых классов, поскольку нужно обновлять иерархию посетителя и его сыновей. |