Технологии разработки программного обеспечения

2020 / 2021, 1 курс, 2 семестр

Пудов Сергей Григорьевич

Лекция 8

- Continuous integration
- Базовые стратегии разработки ПО
  - Каскадная
  - Инкрементная
  - Эволюционная (спиральная)

Continuous integration

- ***Непрерывная интеграция*** (CI, англ. Continuous Integration) — это практика разработки программного обеспечения, которая заключается в слиянии рабочих копий в общую основную ветвь разработки несколько раз в день и выполнении частых автоматизированных сборок проекта для скорейшего выявления и решения интеграционных проблем. В обычном проекте, где над разными частями системы разработчики трудятся независимо, стадия интеграции является заключительной. Она может непредсказуемо задержать окончание работ. Переход к непрерывной интеграции позволяет снизить трудоёмкость интеграции и сделать её более предсказуемой за счет наиболее раннего обнаружения и устранения ошибок и противоречий. Впервые названа и предложена Гради Бучем в 1991 г.[1]

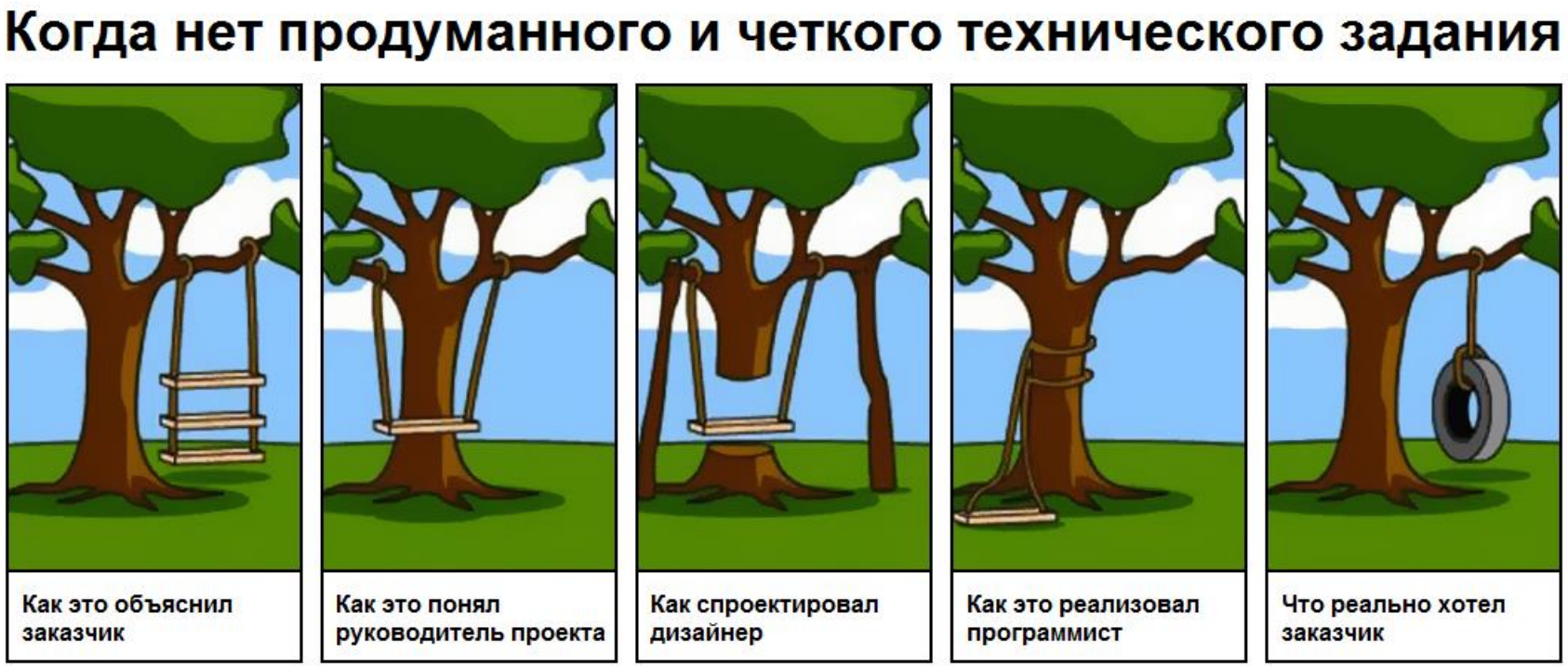
- https://ru.wikipedia.org/wiki/Непрерывная интеграция

Continuous integration

- На выделенном сервере организуется служба, в задачи которой входят:
  - получение исходного кода из репозитория;
  - сборка проекта;
  - выполнение тестов;
  - развёртывание готового проекта;
  - отправка отчетов.
- Локальная сборка может осуществляться:
  - по внешнему запросу,
  - по расписанию,
  - по факту обновления репозитория и по другим критериям.

https://travis-ci.org/

Командная работа ☺



Базовые стратегии разработки ПО

Начальный этап:

кодирование – устранение ошибок

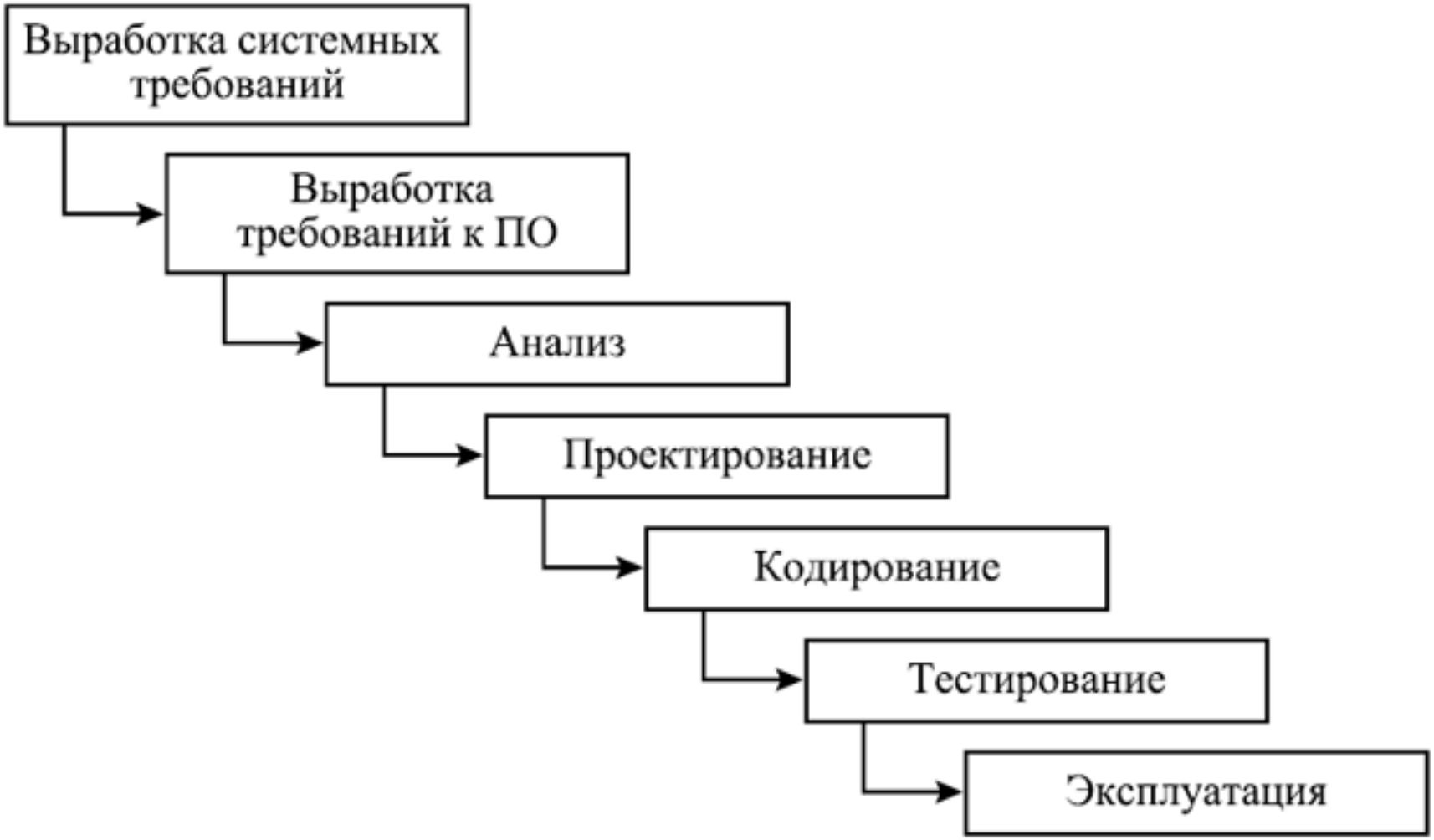
Три базовые стратегии:

- каскадная;
- инкрементная;
- эволюционная.

Определяются характеристиками:

- проекта;
- требований к продукту;
- команды разработчиков;
- команды пользователей.

Каскадная модель (*waterfall model*)



Каскадная модель

Основными достоинствами каскадной стратегии, проявляемыми при разработке соответствующего ей проекта, являются:

1) **стабильность** требований в течение ЖЦ разработки;

2) необходимость только **одного прохода** этапов разработки, что обеспечивает простоту применения стратегии;

3) **простота** планирования, контроля и управления проектом;

4) доступность для **понимания** заказчиками.

Каскадная модель

К основным **недостаткам** каскадной стратегии, проявляемым при ее использовании в проекте, ей не соответствующем, следует отнести:

1) **сложность** полного формулирования требований в начале процесса разработки и невозможность их динамического изменения на протяжении ЖЦ;

2) **линейность** структуры процесса разработки; разрабатываемые ПС или системы обычно слишком велики и сложны, чтобы все работы по их созданию выполнять однократно; в результате возврат к предыдущим шагам для решения возникающих проблем приводит к увеличению финансовых затрат и нарушению графика работ;

3) **непригодность** промежуточных продуктов для использования;

4) **недостаточное участие пользователя** в процессе разработки ПС – только в самом начале (при разработке требований) и в конце (во время приемочных испытаний); это приводит к невозможности предварительной оценки пользователем качества программного средства или системы.

Каскадная модель

Области применения каскадной стратегии определяются ее достоинствами и ограничены ее недостатками. Использование данной стратегии наиболее эффективно в следующих случаях:

1) при разработке проектов с четкими, неизменяемыми в течение ЖЦ требованиями и понятной реализацией;

2) при разработке проектов невысокой сложности, например:

- создание программного средства или системы такого же типа, как уже разрабатывались разработчиками;
- создание новой версии уже существующего программного средства или системы;
- перенос уже существующего продукта на новую платформу;

3) при выполнении больших проектов в качестве составной части моделей ЖЦ, реализующих другие стратегии разработки

Инкрементная модель

- Данная стратегия основана на полном определении всех требований к разрабатываемому программному средству (системе) в начале процесса разработки. Однако полный набор требований реализуется постепенно в соответствии с планом в последовательных циклах разработки.

- Результат каждого цикла называется инкрементом.

- Первый инкремент реализует базовые функции программного средства. В последующих инкрементах функции программного средства постепенно расширяются, пока не будет реализован весь набор требований. Различия между инкрементами соседних циклов в ходе разработки постепенно уменьшаются.

- Результат каждого цикла разработки может распространяться в качестве очередной поставляемой версии программного средства или системы.

Инкрементная модель

Достоинства:

- возможность получения функционального продукта после реализации каждого инкремента;
- короткая продолжительность создания инкремента; это приводит к сокращению сроков начальной поставки, позволяет снизить затраты на первоначальную и последующие поставки программного продукта;
- предотвращение реализации громоздких спецификаций требований; стабильность требований во время создания определенного инкремента; возможность учета изменившихся требований;
- снижение рисков по сравнению с каскадной стратегией;
- включение в процесс пользователей, что позволяет оценить функциональные возможности продукта на более ранних этапах разработки и в конечном итоге приводит к повышению качества программного продукта, снижению затрат и времени на его разработку.

Инкрементная модель

Недостатки:

- необходимость полного функционального определения системы или программного средства в начале ЖЦ для обеспечения планирования инкрементов и управления проектом;
- возможность текущего изменения требований к системе или программному средству, которые уже реализованы в предыдущих инкрементах;
- сложность планирования и распределения работ;
- проявление человеческого фактора, связанного с тенденцией к оттягиванию решения трудных проблем на поздние инкременты, что может нарушить график работ или снизить качество программного продукта.

Инкрементная модель

Область применения

- при разработке проектов, в которых большинство требований можно сформулировать заранее, но часть из них могут быть уточнены через определенный период времени;
- при разработке сложных проектов с заранее сформулированными требованиями; для них разработка системы или программного средства за один цикл связана с большими трудностями;
- при необходимости быстро поставить на рынок продукт, имеющий базовые функциональные свойства;
- при разработке проектов с низкой или средней степенью рисков;
- при выполнении проекта с применением новых технологий

Эволюционная модель

Эволюционная стратегия представляет собой многократный проход этапов разработки. Данная стратегия основана на частичном определении требований к разрабатываемому программному средству или системе в начале процесса разработки. Требования постепенно уточняются в последовательных циклах разработки. Результат каждого цикла разработки обычно представляет собой очередную поставляемую версию программного средства или системы.

Как и при инкрементной стратегии, при реализации эволюционной стратегии зачастую используется прототипирование. В данном случае основной целью прототипирования является обеспечение полного понимания требований

Эволюционная модель

Достоинства:

- возможность уточнения и внесения новых требований в процессе разработки;
- пригодность промежуточного продукта для использования;
- возможность управления рисками;
- обеспечение широкого участия пользователя в проекте, начиная с ранних этапов, что минимизирует возможность разногласий между заказчиками и разработчиками и обеспечивает создание продукта высокого качества;
- реализация преимуществ каскадной и инкрементной стратегий.

Эволюционная модель

Недостатки:

- неизвестность точного количества необходимых итераций и сложность определения критериев для продолжения процесса разработки на следующей итерации; это может вызвать задержку реализации конечной версии системы или программного средства;
- сложность планирования и управления проектом;
- необходимость активного участия пользователей в проекте, что реально не всегда осуществимо;
- необходимость в мощных инструментальных средствах и методах прототипирования;
- возможность отодвигания решения трудных проблем на последующие циклы, что может привести к несоответствию полученных продуктов требованиям заказчиков.

Эволюционная модель

Область применения:

- при разработке проектов, для которых требования слишком сложны, неизвестны заранее, непостоянны или требуют уточнения;
- при разработке сложных проектов, в том числе:
  - больших долгосрочных проектов;
  - проектов по созданию новых, не имеющих аналогов ПС или систем;
  - проектов со средней и высокой степенью рисков;
  - проектов, для которых нужна проверка концепции, демонстрация технической осуществимости или промежуточных продуктов;
- при разработке проектов, использующих новые технологии.

Продолжение в следующей лекции…

Backup

Процесс работы над проектом

http://csc-software-development.readthedocs.io/ru/2018/workflow.html#id6

GitHub Wiki

GitHub Wikis are a place in your repository where you can share long-form content about your project, such as how to use it, how it's been designed, manifestos on its core principles, and so on. Whereas a README is intended to quickly orient readers as to what your project can do, wikis can be used to provide additional documentation.

Wikis can be edited directly on GitHub, or you can work with a text editor offline and simply push your changes. Wikis are collaborative by design. By default, only collaborators on your repository can make changes to wikis, but you can configure this to be enabled for all users on public repositories.

https://help.github.com/articles/about-github-wikis/

Adding wiki pages via the online interface

https://help.github.com/articles/adding-wiki-pages-via-the-online-interface/

You can add new wiki pages directly to your repository using our web interface.

- On GitHub, navigate to the main page of the repository.
- Under your repository name, click Wiki.
- From the top menu bar, click New Page.
- Wiki pages can contain any markup that GitHub supports. The default choice is Markdown, but you can use the "Edit mode" drop-down menu to switch to a different markup language.
- Use the text editor to add your page's content. You can also use the wiki toolbar at the top for entering text using a graphical WYSIWYG editor.
- Enter a commit message describing the new file you're adding.
- To commit your changes to the wiki, click Save Page.

GitHub issues

Issues are a great way to keep track of tasks, enhancements, and bugs for your projects. They're kind of like email—except they can be shared and discussed with the rest of your team. Most software projects have a bug tracker of some kind. GitHub's tracker is called **Issues**, and has its own section in every repository.

https://guides.github.com/features/issues/

GitHub issues

- A **title** and **description** describe what the issue is all about.
- Color-coded **labels** help you categorize and filter your issues (just like labels in email).
- A **milestone** acts like a container for issues. This is useful for associating issues with specific features or project phases (e.g. *Weekly Sprint 9/5-9/16* or *Shipping 1.0*).
- One **assignee** is responsible for working on the issue at any given time.
- **Comments** allow anyone with access to the repository to provide feedback.