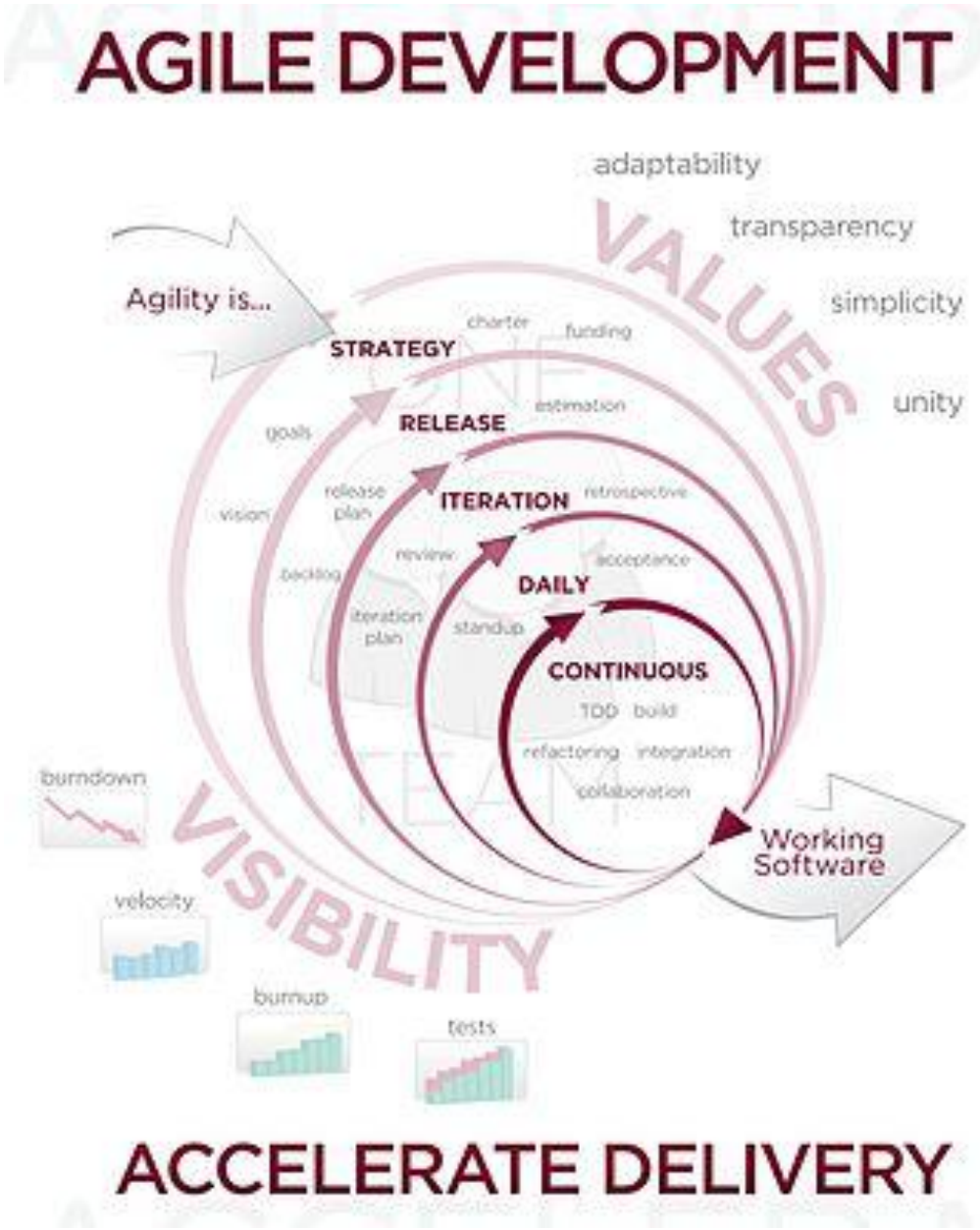
Лекция № 2

Тема 1. Технологии разработки программного обеспечения (ПО).
Часть 2. Гибкие технологии разработки ПО.

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ



1.3. Гибкие технологии разработки ПО. Основные положения Agile Manifesto

Гибкая методология разработки (англ. *Agile software development*) — это концептуальный каркас, в рамках которого выполняется разработка программного обеспечения.
Гибкие технологии разработки программного обеспечения минимизируют риски благодаря разделению процесса разработки на маленькие промежутки времени – ***итерации*** (1-4 недели). Каждая итерация может рассматриваться как полноценный проект по разработке ПО. Так, ***итерация*** может ***включать*** в себя все основные процессы разработки, такие как планирование, анализ требований, проектирование, реализация, тестирование и документирование.
Обычно, результатом итерации не является продукт, готовый к выходу на рынок. Но целью каждой итерации является получение стабильной версии продукта. В конце каждой итерации происходит переоценка приоритетов проекта, что значительно сокращает риски.

Основные положения Agile Manifesto [http://agilemanifesto.org]

Agile — семейство процессов разработки, а не единственный подход в разработке программного обеспечения, и определяется Agile Manifesto. Agile не включает практик, а определяет ценности и принципы, которыми руководствуются успешные команды.

Agile Manifesto разработан и принят 11-13 февраля 2001 года на лыжном курорте The Lodge at Snowbird в горах Юты. Манифест подписали представители следующих методологий Extreme programming, Scrum, DSDM, Adaptive Software Development, Crystal Clear, Feature-Driven Development, Pragmatic Programming.

Agile Manifesto содержит 4 основные идеи и 12 принципов.
Примечательно что, Agile Manifesto не содержит практических советов.

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Люди и взаимодействие **важнее** процессов и инструментов

Работающий продукт **важнее** исчерпывающей документации

Сотрудничество с заказчиком **важнее** согласования условий контракта

Готовность к изменениям **важнее** следования первоначальному плану

То есть, не отрицая важности того, что справа, мы всё таки больше ценим то, что слева.

Основополагающие принципы Agile-манифеста

1. Наивысшим приоритетом для нас является удовлетворение потребностей заказчика, благодаря регулярной и ранней поставке ценного программного обеспечения.
2. Изменение требований приветствуется, даже на поздних стадиях разработки. Agile-процессы позволяют использовать изменения для обеспечения заказчику конкурентного преимущества.
3. Работающий продукт следует выпускать как можно чаще, с периодичностью от пары недель до пары месяцев.
4. На протяжении всего проекта разработчики и представители бизнеса должны ежедневно работать вместе.
5. Над проектом должны работать мотивированные профессионалы. Чтобы работа была сделана, создайте условия, обеспечьте поддержку и полностью доверьтесь им.
6. Непосредственное общение является наиболее практичным и эффективным способом обмена информацией как с самой командой, так и внутри команды.

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

7. Работающий продукт — основной показатель прогресса.
8. Инвесторы, разработчики и пользователи должны иметь возможность поддерживать постоянный ритм бесконечно. Agile помогает наладить такой устойчивый процесс разработки.
9. Постоянное внимание к техническому совершенству и качеству проектирования повышает гибкость проекта.
10. Простота — искусство минимизации лишней работы — крайне необходима.
11. Самые лучшие требования, архитектурные и технические решения рождаются у самоорганизующихся команд.
12. Команда должна систематически анализировать возможные способы улучшения эффективности и соответственно корректировать стиль своей работы.

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Краеугольным камнем гибких технологий программирования является **разработка через тестирование**:

автоматические тесты пишутся для любой части реализации, которая гипотетически «может сломаться»;

тесты пишутся непосредственно *перед* написанием соответствующего кода;

существующий код никогда не меняется без написания соответствующих тестов;

выполняется регулярный запуск всех автоматических тестов.

Все гибкие методологии имеют общие характеристики:

✔ итеративная разработка;

✔ фокус на взаимодействии и коммуникации;

✔ полный или частичный отказ от создания дорогостоящих промежуточных артефактов проекта.

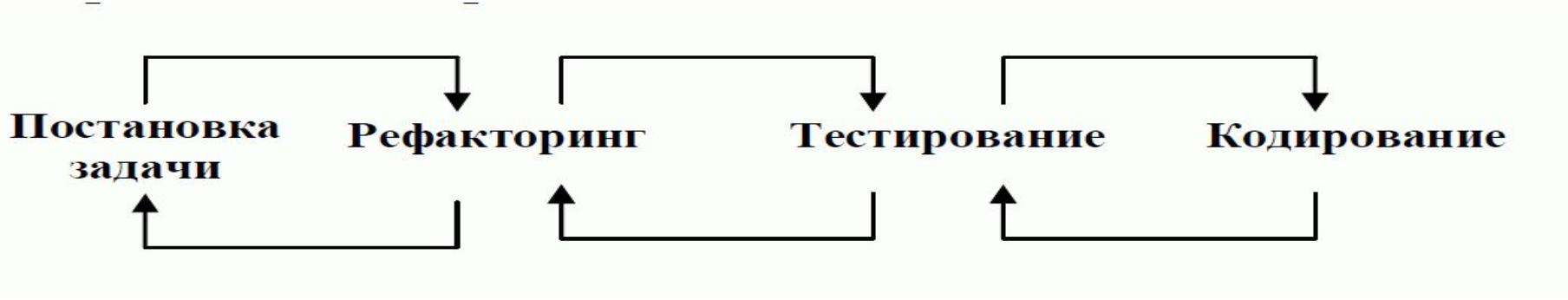
Проектирование в гибких технологиях

✔Отказ от длительного проектирования перед началом работы и выполнение проектирования на протяжении всего выполнения проекта.

✔В начале проекта выполняется лишь *формирование общего представления*. Для этого используются *системные метафоры*, на основе которых формируется высокоуровневая схема проекта.

✔Процесс разработки состоит из большого количества очень коротких циклов. Конечный результат этапа планирования – список задач, подлежащих реализации на следующей итерации.

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ



- ✔ Разработчики получают задачу, берут соответствующий фрагмент разрабатываемого кода, выполняют рефакторинг, необходимый для упрощения написанного кода, составляют тесты, а только затем создают сам код, который должен пройти тесты.

- ✔ Поскольку циклы «*дизайн–тест–код*» непродолжительны, а заказчик часто получает работающие версии программного продукта, обратная связь осуществляется непрерывно и служит для контроля, что проектирование и кодирование продвигаются в нужном направлении.

- ✔ Так как изменения на каждом цикле малы, решения, от которых приходится отказываться, невелики, в результате чего можно быстро реагировать на изменения с наименьшими затратами.

1.3.1. Экстремальное программирование (Extreme Programming)

Экстрема́льное программи́рование (англ. *Extreme Programming*, *XP*) — одна из гибких методологий разработки программного обеспечения. Авторы методологии — Кент Бек, Уорд Каннингем, Мартин Фаулер и другие.

Экстремальное программирование — является наиболее известной из гибких методологий.

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Экстрема́льное программи́рование строится на 12 принципах, которые можно объединить в 4 группы:

Короткий цикл обратной связи

1. Разработка через тестирование
2. Игра в планирование
3. Заказчик всегда рядом
4. Парное программирование

Непрерывный, а не пакетный процесс

5. Непрерывная интеграция
6. Рефакторинг
7. Частые небольшие релизы

Понимание, разделяемое всеми

8. Простота
9. Метафора системы
10. Коллективное владение кодом или выбранными шаблонами проектирования
11. Стандарт кодирования

Социальная защищенность программиста

12. 40-часовая рабочая неделя

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

✔**Основная идея экстремального программирования (XP)** — устранить высокую стоимость изменений, вносимых в ПО в процессе как разработки, так и эксплуатации.

✔Цикл разработки в XP состоит из очень коротких итераций. Четырьмя базовыми действиями в цикле являются:

- выслушивание заказчика
- проектирование
- кодирование
- тестирование.

✔Заказчик постоянно присутствует в группе разработчиков.

✔При принятии решений всегда стремятся выбрать самое простое, тесты пишутся еще до написания кода.

✔Сборка системы выполняется ежедневно.

Тестирование в XP

Тестирование модулей (unit testing):

✔позволяет разработчикам убедиться, что код работает корректно, и без опасений выполнять рефакторинг (refactoring);

✔помогает не авторам кода понять, зачем нужен тот или иной фрагмент кода и как он функционирует.

Приемочное тестирование (acceptance testing):

✔позволяет убедиться в том, что система действительно обладает заявленными возможностями и функционирует корректно.

TDD (Test Driven Development):

✔пишется тест (не проходит);

✔пишется код, чтобы тест прошел;

✔выполняется рефакторинг кода.

1.3.2. Scrum

Подход впервые описали Хиротака Такеути и Икудзиро Нонака в статье *The New Product Development Game* (*Гарвардский Деловой Обзор*, *январь-февраль 1986*).

Впервые метод Scrum был представлен на общее обозрение задокументированным, чётко сформированным и описанным совместно Сазерлендом и Швабером на OOPSLA'96 в Остине.

Швабер объединил усилия с Майком Бидлом в 2001 году, чтобы детально описать метод в книге «Agile Software Development with SCRUM».

Scrum — это набор принципов, на которых строится процесс разработки, позволяющий в жёстко фиксированные небольшие промежутки времени (*спринты* от 2 до 4 недель) предоставлять конечному пользователю работающее ПО с новыми возможностями, для которых определён наибольший приоритет. Возможности ПО к реализации в очередном спринте определяются в начале спринта на этапе планирования и не могут изменяться на всём его протяжении. При этом строго-фиксированная небольшая длительность спринта придаёт процессу разработки предсказуемость и гибкость.

Общие положения Scrum

3 роли:

владелец продукта (*Product Owner*) - отвечает за определение требований к продукту

команда (*Team*) - группа самостоятельных и инициативных разработчиков, ответственных за реализацию проекта

скрам-мастер (*ScrumMaster*) - отвечает за решение всех организационных проблем и соблюдение методологии *Scrum*.

3 фазы проекта:

Подготовка (*Pregame*)*:* общий план проекта, список основных требований к продукту, высокоуровневая архитектура продукта.

Реализация (*Game*)*:* итеративное развитие продукта.

Завершение (*Postgame*)*:* действия, необходимые для подготовки продукта к выходу на рынок.

Реализация проекта в Scrum

✔Фаза реализации разбита на последовательность итераций - *спринтов* (*Sprint*).

✔В результате каждого спринта в продукте реализуется новый, заметный для владельца продукта, объем функциональности.

✔В конце каждого спринта продукт остается в работоспособном состоянии.

✔Спринт начинается с сессии планирования (*Sprint Planning Meeting*) - определяется объем функциональности, которая будет реализована в течение спринта.

✔Ежедневно проводится собрание участников проекта - *скрам-сессия* (*Daily Scrum Meeting*).

✔По завершению спринта проводится демонстрационная сессия (*Sprint Review Meeting*).

Документация в Scrum

Всего 3 документа:

***журнал продукта* (*Product Backlog*)**

высокоуровневый список функциональных и технических требований, необходимых для реализации продукта

***журнал спринта* (*Sprint Backlog*)**

детализированный список функциональных и технических требований, необходимых для успешного завершения итерации

***график спринта* (*Burndown Chart*).**

показывает ежедневное изменение общего объема работ, оставшегося до завершения итерации.

1.3.3. Microsoft Solutions Framework

В 1994 году, стремясь достичь максимальной отдачи от IT-проектов, Microsoft выпустила в свет пакет руководств по эффективному проектированию, разработке, внедрению и сопровождению решений, построенных на основе своих технологий.

Microsoft Solutions Framework — концепция разработки программного обеспечения от Microsoft. MSF предлагает методики для планирования, проектирования, разработки и внедрения IT-решений. MSF обладает высокой гибкостью, масштабируемостью, отсутствием жестких инструкций и способен удовлетворить нужды организации или проектной группы любого размера.

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Команда состоит из ролей, каждая из которых ориентирована на достижение определенных целей:

- ✔Управление продуктом
- ✔Управление программой
- ✔Разработка
- ✔Тестирование
- ✔Удовлетворение потребителя
- ✔Управление выпуском

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Процесс разработки по MSF является итеративным и включает в себя следующие основные фазы:

- ✔Выработка концепции
- ✔Планирование
- ✔Разработка
- ✔Стабилизация
- ✔Внедрение

Модель процессов MSF содержит весь жизненный цикл создания решения, включая его внедрение — вплоть до момента, когда решение начинает давать отдачу.

1.3.4. Feature Driven Development

FDD разработана Джеффом Де Люка (Jeff De Luca) для проекта, рассчитанного на 15 месяцев и в котором участвовали 50 человек,по разработки программного обеспечения для одного крупного Сингапурского банка в 1997. Джефф Де Люка выделил 5 процессов, охватывающие развитие общей модели, листиниг,планирование, проектирование и построение функций.

FDD (Feature Driven Development) – функционально-ориентированная разработка. Используемое в FDD понятие функции или свойства (feature) системы достаточно близко к понятию прецедента использования, используемому в RUP, существенное отличие — это дополнительное ограничение: «каждая функция должна допускать реализацию не более, чем за две недели». То есть если сценарий использования достаточно мал, его можно считать функцией. Если же велик, то его надо разбить на несколько относительно независимых функций.

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

FDD включает пять процессов, последние два из которых повторяются для каждой функции:

- ✔Разработка общей модели
- ✔Составление списка необходимых функций системы
- ✔Планирование работы над каждой функцией
- ✔Проектирование функции
- ✔Конструирование функции

ГИБКИЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Разработчики в FDD делятся на «хозяев классов» и «главных программистов». Главные программисты привлекают хозяев задействованных классов к работе над очередным свойством. Работа над проектом предполагает частые сборки и делится на итерации, каждая из которых предполагает реализацию определенного набора функций.

К сожалению, такие области, как обеспечение качества, оценка рисков данная модель обходит стороной. FDD подходит для небольших проектов, срок реализации которых составляет несколько месяцев. Для долгосрочных проектов, со сроком производства год или более, предпочтительно выбрать другую, более формализованную модель.

1.4. Сравнение традиционных и гибких технологий разработки ПО

Подготовить реферат

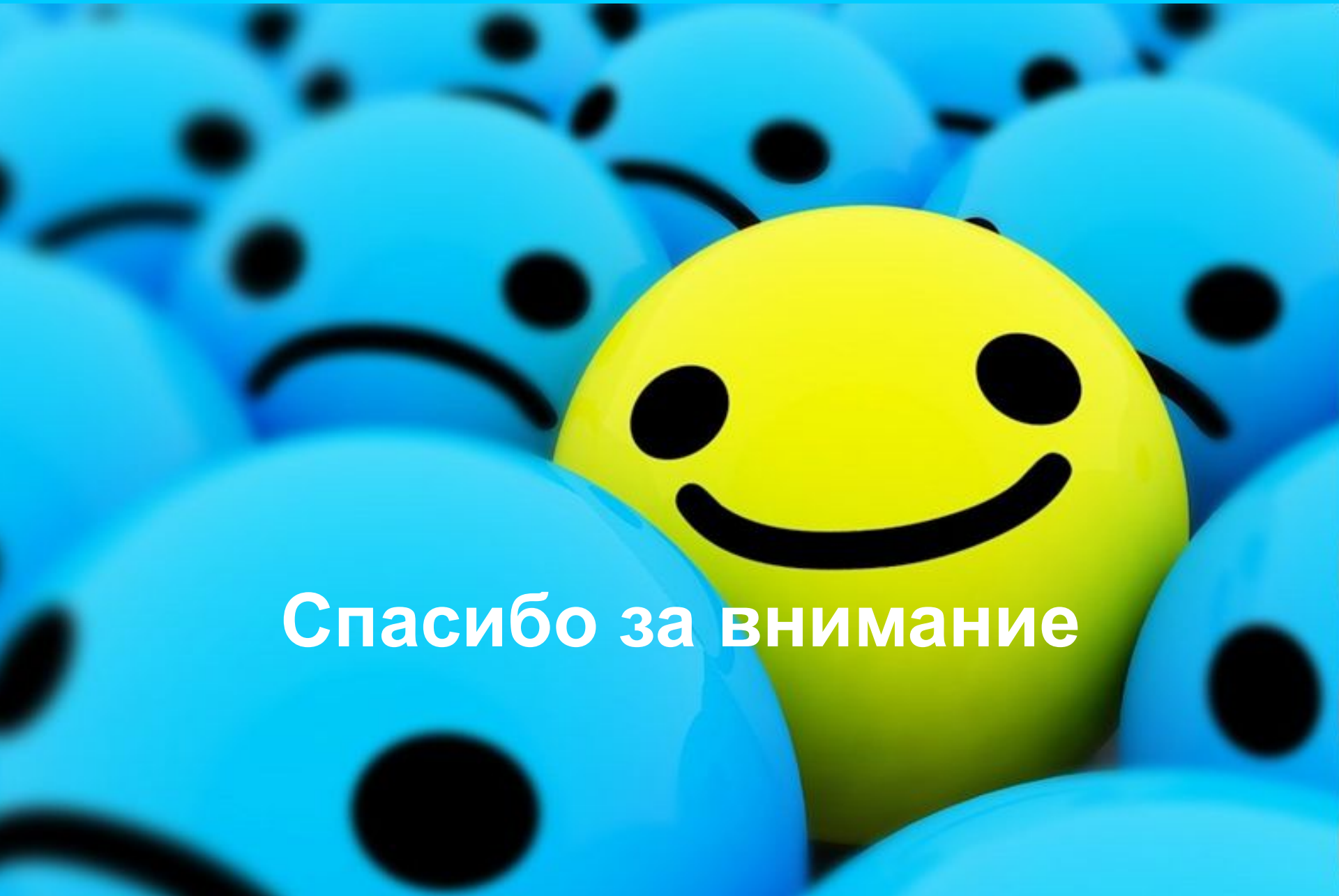
Краткие выводы по гибким методологиям

Гибкие методологии используют итеративный подход, при котором детально планируется только ограниченный объем работ, связанный с выпуском очередного релиза.

Практически все гибкие методологии ориентированы на максимально неформальный подход к разработке. Если проблему можно решить в разговоре, то лучше именно так и сделать. Оформлять принятое решение в виде бумажного или электронного документа нужно только тогда, когда без этого невозможно обойтись.

Также большинство методологий используют довольно ограниченный набор документов, моделей и работ для описания процесса разработки. Это делает их достаточно простыми (по крайней мере, на первый взгляд) для внедрения. Хотя часто эта простота достигается за счет того, что многие, безусловно, необходимые работы только упоминаются, а не описываются в методологии.

И практически все они не предусматривают специфических усилий, которых требует разработка принципиально новой архитектуры системы, например, если команда переходит с клиент-серверной архитектуры на использование интернет-технологий.



Спасибо за внимание