ЭКСТРЕМАЛЬНОЕ ПРОГРАММИРОВАНИЕ

ЦЕЛИ **XP**

Выполнение поставленной задачи полностью;

Выполнение поставленной задачи в срок;

Возможность любому члену команды в минимальные сроки внести изменения в любую часть кода;

Быстрое понимание новыми членами команды архитектуры проекта;

Быстрое обучение молодых разработчиков

ЦЕННОСТИ **XP**

Коммуникация (Communication)

Простота (Simplify)

Обратная связь (Feedback)

Смелость (Courage)

ПРАКТИКИ **XP**

1. Игра в планирование
2. Тестирование
3. Парное программирование
4. Рефакторинг
5. Простой дизайн
6. Коллективное владение кодом
7. Постоянная интеграция
8. Заказчик на месте с разработчиками
9. Частые выпуски версий
10. 40-часовая рабочая неделя
11. Стандарты кодирования
12. Метафора системы

ИГРА В ПЛАНИРОВАНИЕ (PLANNING GAME)

Задачи команды

1. **Оценка времени для реализаций каждого из пожеланий;**
2. **Оценка времени связанной с выбором технологии**
3. **Распределение задачи между командами;**
4. **Оценка рисков, связанных с каждым из пожеланий;**
5. **Определение порядка в котором будет реализованы задачи.**

Задачи заказчика

1. **Определяет набор пожеланий и конкретизирует каждую итерацию;**
2. **Определяет дату завершения работы(Realize)**
3. **Определяет приоритеты задач**

Планирование должно выполняться как можно чаше

ПЕРВИЧНОЕ ОБСЛЕДОВАНИЕ

В начале работы над проектом разработчики и заказчики обсуждают новую систему, чтобы выявить наиболее существенные функции.

Однако они не пытаются идентифицировать *все вообще* функции. По мере развертывания работ заказчики будут обнаруживать все новые и новые функции. Поток функций не иссякнет вплоть до момента завершения проекта. Вновь выявленная функция разбивается на одну или несколько *пользовательских историй*, которые записываются на учетной карточке или чем-то подобном.

Пример, «Вход в систему», «Добавление пользователя», «Удаление пользователя», «Изменение пароля»

Разработчики совместно оценивают истории в баллах. Эти оценки относительны, а не абсолютны.

ОБЪЕДИНЕНИЕ, РАЗБИЕНИЕ И СКОРОСТЬ

Если история слишком велика, то ее следует разбить на меньшие части.

Если история слишком мала, ее следует объединить с другими

История «Пользователи могут безопасно переводить деньги на свой счет, со своего счета или с одного счета на другой» может быть разбита на истории:

- Пользователь может войти в систему
- Пользователь может выйти из системы
- Пользователь может положить деньги на свой счет
- Пользователь может снять деньги со своего счета
- Пользователь может перевести деньги с любого из своих счетов на другой

ПЛАНИРОВАНИЕ ВЫПУСКА

Каждую неделю мы завершатся реализация нескольких историй. Сумма оценок завершенных историй дает *скорость*.

Зная скорость, заказчик может получить представление о стоимости каждой истории, а также о ее значимости для бизнеса и о приоритете. Это позволяет заказчику решить, какие истории реализовывать первыми.

Разработчики и заказчик согласуют дату первого выпуска проекта. Обычно это занимает 2–4 месяца. Заказчик указывает, какие истории реализовать в этом выпуске и примерный порядок реализации. Не разрешается выбирать больше историй, чем укладывается в текущую скорость.

ПЛАНИРОВАНИЕ ИТЕРАЦИИ

Разработчики и заказчик определяют длительность итерации: обычно 1 или 2 недели. И снова заказчик решает, какие истории реализовать на первой итерации, но не может выбрать больше историй, чем позволяет текущая скорость.

Порядок реализации историй в пределах итерации – вопрос решаемый самими разработчиками.

Заказчик не может изменять истории после начала итерации. Он вправе изменять или переупорядочивать любые истории, кроме тех, над которыми разработчики уже трудятся.

Итерация заканчивается в заранее оговоренный день, даже если не все истории реализованы. Оценки всех завершенных историй суммируются, и вычисляется скорость на данной итерации. Эта величина ис-пользуется для планирования следующей итерации.

ОПРЕДЕЛЕНИЯ ПОНЯТИЯ «ГОТОВО»

История не считается реализованной, пока не пройдут *все* приемочные тесты. Эти тесты автоматизированы.

Пишут их заказчики, бизнес аналитики, специалисты по контролю качества, тестировщики и даже программисты в начале каждой итерации.

Тесты определяют детали истории и являются неоспоримым авторитетом в вопросе о поведении историй.

ПЛАНИРОВАНИЕ ЗАДАЧ (1)

В начале новой итерации разработчики и заказчики вырабатывают план. Разработчики разбивают истории на задачи.

Задачей называется объем работы, с которым один разработчик может справиться за 4–16 часов.

Истории анализируются совместно с заказчиком, и задачи формулируются максимально полно.

Список задач записывается в лекционном блокноте, на доске или на любом другом носителе.

Затем разработчики выбирают себе задачи, которые хотели бы реализовать, присваивая каждой задаче произвольную оценку в баллах.

ПЛАНИРОВАНИЕ ЗАДАЧ (2)

Разработчик может выбрать любую задачу вне зависимости от специализации.

Каждый разработчик знает, сколько баллов он заработал на задачах на предыдущей итерации; это число составляет *бюджет* разработчика.

Никто не берет себе задач на большую сумму, чем его бюджет.

В середине итерации команда устраивает рабочее совещание. В этот момент должна быть реализована половина *историй*, запланированных на даннуюитерацию. Если это не так, то команда пытается перераспределить задачи и ответственность таким образом, чтобы к концу итерации все истории были реализованы.

ПОДВЕДЕНИЕ ИТОГОВ ИТЕРАЦИЙ

Раз в две недели текущая итерация заканчивается и начинается новая.

В конце каждой итерации заказчику демонстрируется текущая версия системы.

Заказчика просят оценить внешний вид и функциональность проекта. Заказчик выражает свое мнение в виде набора новых пользовательских историй.

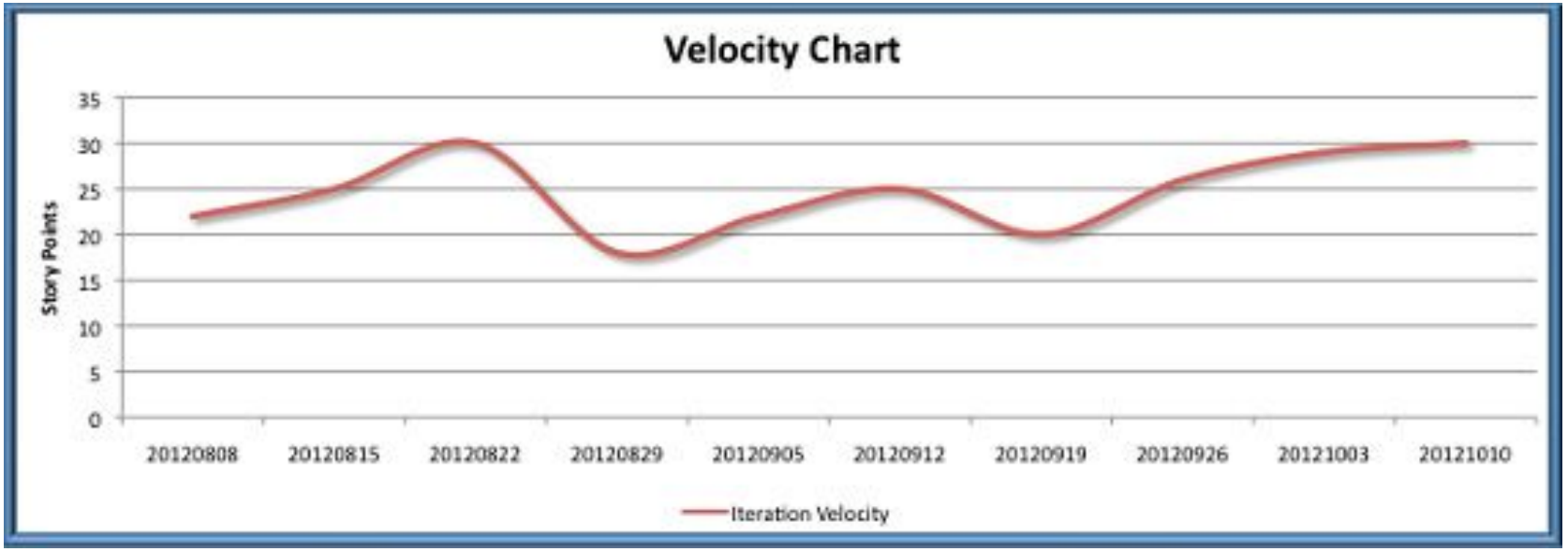
Заказчика часто информируют о ходе работы над проектом.

Он может измерить скорость.

Он может оценить, как быстро продвигается команда, и запланировать высокоприоритетные истории на ранней стадии.

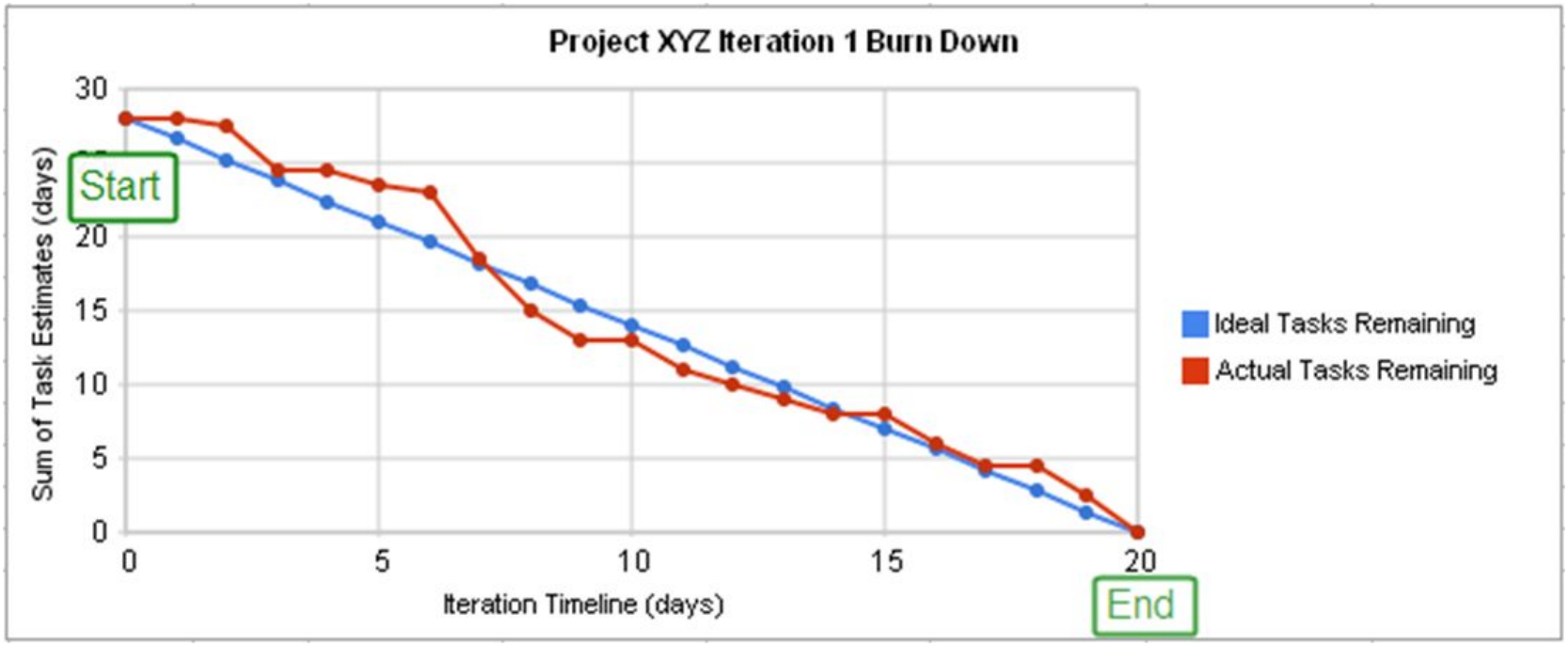
МОНИТОРИНГ (1)

Мониторинг и управление XP-проектом сводятся к записи результатов каждой итерации и использованию этих данных для прогнозирования следующих итераций. Рассмотрим, например, рис.. Этот график называется *диаграммой скорости*.



МОНИТОРИНГ (2)

Диаграмма выгорания показывает, сколько баллов оставалось набрать в конце каждой недели до следующей важной контрольной точки. Наклон этого графика – индикатор даты завершения.



ТЕСТИРОВАНИЕ (TESTING)

Тестирование применяемое в XP

- Тестирование модулей (unit testing)
- Приемочное тестирование (acceptance testing)

РАЗРАБОТКА ЧЕРЕЗ ТЕСТИРОВАНИЕ

1. Не писать код, пока не будет написан автономный тест, который не проходит.
2. Не писать автономный тест большего объема, чем необходимо для неудачного завершения или неудачной компиляции.
3. Не писать больше кода, чем необходимо для того, чтобы ранее не проходивший тест завершился успешно.

Написание теста до кода способствует проектированию кода, который было бы *удобно вызывать*.

Предварительное написание тестов *заставляет писать слабо связанные программы*!

Тесты могут выступать как чрезвычайно полезная документация. Она всегда актуальна. Она не лжет.

ПРОГРАММИРОВАНИЕ ПАРАМИ (PAIR PROGRAMMING)

Преимущества парного программирования

- Любые решения принимаются не одним программистов а двумя;
- Какую бы часть системы не взять, в ней будут хорошо разбираться как минимум два человека;
- В паре партнеры контролируют друг друга – снижение вероятности написания не корректного, не аккуратного кода;
- Партнеры в парах постоянно меняются, это дает возможность всем членам команды знать функционал системы

РЕФАКТОРИНГ (REFACTORING)

Refactoring – процесс изменения программной системы таким образом, что ее внешнее поведение не изменяется, а внутренняя структура улучшается.

У каждого программного модуля есть три функции:

1) та функция, которую он реализует во время выполнения
2) модуль должен допускать изменение. Почти все модули на протяжении своей жизни изменяются, и задача разработчика – обеспечить возможность и простоту внесения изменения. Модуль, который трудно изменять, следует считать непригодным и нуждающимся в исправлении, даже если он работает.
3) модуль должен быть доступным для понимания. Разработчики, не знакомые с модулем, должны иметь возможность прочитать и понять его, не прилагая сверхъестественных умственных усилий. Модуль, который не информативен, также непригоден и нуждается в исправлении.

ПРОСТОЙ ДИЗАЙН (SIMPLE DESIGN)

Цели Simple Design:

- Спроектировать систему один раз и на всегда невозможно. В XP – проектирование происходит постоянно.

Simple Design :

- Обеспечение корректного срабатывания всех тестов;
- Отсутствие дублирующего кода;
- Хорошо выраженные намерения программиста для конкретного участка кода;
- Минимальное количество классов и методов;

КОЛЛЕКТИВНОЕ ВЛАДЕНИЕ КОДОМ **(COLLECTIVE CODE OWNERSHIP)**

Каждый член команды может вносить изменения в любой код системы;

Разработчик может выбрать любую задачу. Специалисты по базе данных не обязаны ограничиваться только задачами, связанными с базой.

Специалисты по разработке ГИП могут при желании выбрать задачу, касающуюся базы данных. Хотя на первый взгляд это может показаться неэффективным, но выгода очевидна: чем больше каждый разработчик знает о проекте *в целом*, тем успешнее и информированнее вся команда.

Добиваемся, чтобы за проект отвечала команда целиком, независимо от специализации.

УЛУЧШЕНИЕ ДИЗАЙНА

Плохой код не должен доживать до заката. Код все время должен оставаться максимально чистым и выразительным.

ПОСТОЯННАЯ ИНТЕГРАЦИЯ (CONTINUOUS INTEGRATION)

Интеграция модулей в систему выполняется постоянно, несколько раз в день.

Обеспечение интеграции осуществляется за счет систем совместной работы.

ЗАКАЗЧИК В КОМАНДЕ **(ON-SITE CUSTOMER)**

Постоянная доступность заказчика, который может ответить на возникшие вопросы, пускай даже по телефону;

Наличие заказчика в команде позволяет работать с оптимальной скоростью.

«… ***Я предпочел бы, чтобы нужный человек был доступен мне, по крайне мере один час в день, чем если бы ненужный человек был доступен мне в течении всего рабочего дня, или нужный человек – в течении одного дня в неделю*****» Кен Ауэр.**

ЧАСТЫЕ ВЫПУСКИ ВЕРСИЙ **(SMALL REALIZE)**

Версии должны выпускаться в эксплуатацию, как можно чаще. Работа над каждой версией должна занимать как можно меньше времени. При этом каждая версия должна быть осмысленной.

40 – ЧАСОВАЯ РАБОЧАЯ НЕДЕЛЯ (40-HOUR WEEK)

«… желаю быть свежим и энергичным каждое утро, ровно как уставшим и удовлетворенным каждый вечер » K.Beck Extrem Programming.

СТАНДАРТЫ КОДИРОВАНИЯ (CODING STANDARDS)

Весь код выглядит так, будто его писал один – очень квалифицированный – человек.

Благодаря стандартам:

- Члены команды не тратят время на глупые споры о вещах, которые фактически никак не влияют на скорость работы над проектом;
- Обеспечивается эффективное выполнение остальных практик.

МЕТАФОРА СИСТЕМЫ (SYSTEM METAPHOR)

Команда поддерживает общее видение работы программы.

Метафора системы - это архитектура системы.

Метафора системы дает команде представление о том, каким образом система работает в настоящее время, в какие места добавляются новые компоненты и какую фору они должны принять.

ПОСТАНОВКА ЗАДАЧИ

Разработать Web-портал для хранения и анализа информации о скидках на товары.

Основные функциональные требования к системе:

Возможность регистрации нового пользователя в системе.

Возможность бана пользователя за нарушения разглашения информации о скидках

Возможность бана скидки. Если более 10 разных пользователей в течении 10 дней оценит негативно скидку(-5) она банится, при этом статус пользователя, который ее разместил понижается на 1 бал.

Возможность добавлять информацию о скидке указывая название магазина, название товара, скидка, базовая цена товара (по желанию).

Возможность выводить списки всех скидок по определенной категории, магазину, району.

Возможность находить товар со скидкой по определенному критерию.

Возможность группировать товары по категориям и все отображать на экране браузера.

Возможность получать информацию о текущих скидках на электронный ящик, только в том случае, если оплатил ваучер.

Возможность получать информацию на электронный адрес или на экран браузера о предстоящих скидках, в том случае если оплатил ваучер.

Возможность добавлять комментарии к текущим или выполненным скидкам, а также оценивание их по 10 бальной шкале от -5 до +5.

Архитектура приложения

Наличие модульной системы.

ЮНИТ-ТЕСТЫ (UNIT-TESTS)

Цель:

- Проверка работоспособности программы;
- Формирование простых требований к разрабатываемому коду.

Системы unit-tests

- Встроенные в среду разработки;
- Resharper;
- jUnit.
- и т.д.