Agile

введение

расскажет Михаил В. Шохирев

Клуб программистов
Шадринск
2024

О чём пойдёт речь

Agile: ***введение***
*(~2**5 минут)*

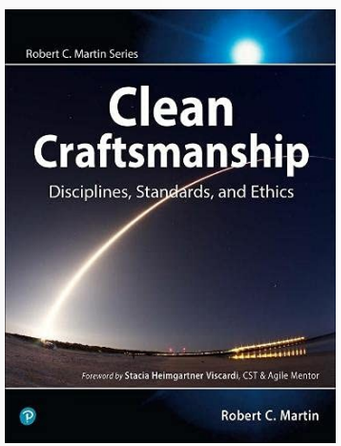
Боб Мартин ~ ***лучше первоисточника ничего нет.***
Waterfall ~ ***традиционный подход к разработке: поэтапный.***
Agile ~ ***адаптивный подход к разработке: эволюционный.***
Agile Manifesto ~ ***от принципов к практике.***
Технологии Agile ~ ***разное применение принципов.***
X. P. ~ ***“экстремальное программирование”: Agile в работе.***
4 ~ ***основных положений X. P.***
12 ~ ***основных приёмов X. P.***
Agile ~ ***гибкий подход: ручейки вместо водопада.***
Команда ~ ***«спецназ» быстрого реагирования делает Agile.***
Agile ~ ***польза, критика, выводы.***

Первоисточник

Uncle” Bob Martin

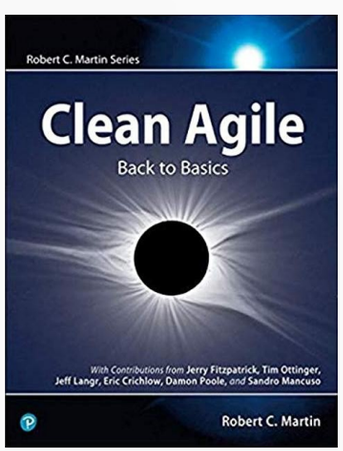
“Дядюшка” **Роберт Мартин**, *как один из создателей движения Agile, изложил историю, основы и применение принципов на практике в своей книге*
«Clean Agile: Back to Basics» (2019) /
«Чистый Agile. Основы гибкости» (2020)

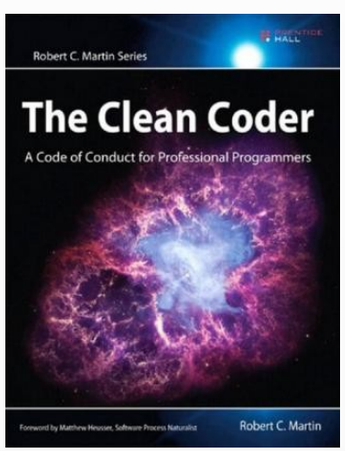
Книги «дядюшки Боба»

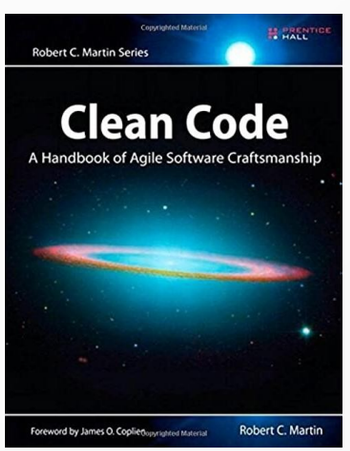


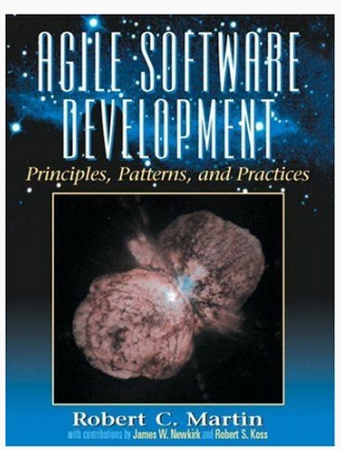
Robert "Uncle Bob" Martin

с 1970 – профессионально разрабатывает ПО,
с 1990 – международный консультант в этой области, с 1996 по 1999 – главный редактор журнала "C++ Report",
в 2001 организовал встречу группы, которая положила начало Agile-методологиям разработки программ.









Традиционный подход к разработке

Waterfall

«Водопад»:

Требования: *определить и проанализироать требования заказчика*.

Проектирование: *формализованно описать и утвердить проект*.

Программирование: *написать программы по спецификациям*.

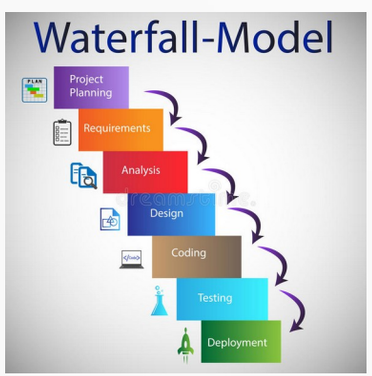
Отладка: *найти и исправить ошибки кодирования*.

Тестирование: *показать, что всё работает, как нужно*.

Документирование: *теперь можно написать мануалы*.

Внедрение: *когда всё готово и протестировано.*

Сопровождение: *при эксплуатации.*



«Утвердить (зафиксировать) проект – и спокойно программировать...»

Традиционный подход *не работает!*

~~Waterfall~~

«Водопад»:

Требования: *всё время меняются*.

Проектирование: *невозможно закончить полностью и в срок*.

Программирование: *по неполному проекту, решения на ходу*.

Отладка: *поэтому ошибки неизбежны*.

Тестирование: *недостаточное и несвоевременное*.

Документирование: *утомительное и после времени*.

Внедрение: *когда «поджимают сроки»*.

Сопровождение: *фактически – это разработка.*

«Эта сказка хороша – начинай сначала!».

Ни зафиксировать, ни спокойно програмировать не удаётся…

«Manifesto for Agile Software Development»

Впоследствии к ***Agile Manifesto*** присоединились многие другие разработчики.

Ещё больше разработчиков стало практиковать принципы Agile в своей работе.

Со временем появилось несколько популярных технологий, формализующих принципы Agile.

В 2001 году 17 разработчиков программного обеспечения собрались в курортном местечке Snowbird (штат Юта, США), чтобы обсудить легковесные методы разработки.

Это были: **Kent Beck** (Extreme Programming), **Ward Cunningham** (Extreme Programming), **Dave Thomas** (PragProg, Ruby), **Jeff Sutherland** (Scrum), **Ken Schwaber** (Scrum), **Jim Highsmith** (Adaptive Software Development), **Alistair Cockburn** (Crystal), **Robert C. Martin** (SOLID), **Mike Beedle** (Scrum), **Arie van Bennekum**, **Martin Fowler** (OOAD & UML), **James Grenning**, **Andrew Hunt** (PragProg, Ruby), **Ron Jeffries** (Extreme Programming), **Jon Kern**, **Brian Marick** (Ruby, TDD), and **Steve Mellor** (OOA).

Вместе они опубликовали ***Manifesto for Agile Software Development***.

«Agile Manifesto» (2001)

«Мы постоянно открываем для себя более совершенные методы разработки программного обеспечения, непосредственно занимаясь разработкой и помогая в этом другим. Благодаря проделанной работе мы смогли осознать, что:

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**.»

Kent Beck, *Mike Beedle, Arie van Bennekum,* ***Alistair Cockburn***, ***Ward Cunningham***, ***Martin Fowler***, *James Grenning, Jim Highsmith,* ***Andrew Hunt***, *Ron Jeffries, Jon Kern, Brian Marick,* ***Robert C. Martin***, *Steve Mellor, Ken Schwaber, Jeff Sutherland,* ***Dave Thomas*** + многие другие позже.

Принципы манифеста Agile

1. **Удовлетворение потребностей заказчика** путём ранней и регулярной поставки значимого ПО.
2. **Изменения требований приветствуется**, даже на поздних стадиях разработки.
3. **Работающий продукт выпускается часто**, с периодичностью в недели, а не месяцы.
4. **Разработчики и представители бизнеса тесно работают вместе** на протяжении всего проекта.
5. **Над проектом работают мотивированные профессионалы**, которым заказчик доверяет.
6. **Непосредственное общение — лучший способ взаимодействия** с командой и внутри команды.
7. **Работающий продукт** — основной показатель прогресса разработки.
8. **Поддерживать устойчивый процесс разработки в постоянном ритме.**
9. **Постоянное внимание к техническому совершенству и качеству проектирования**.
10. Простота — **искусство минимизации лишней работы** — крайне необходима.
11. Наилучшие проектные решения создаются **самоорганизующимися командами**.
12. Команда систематически анализирует пути улучшения и **корректирует стиль своей работы**.

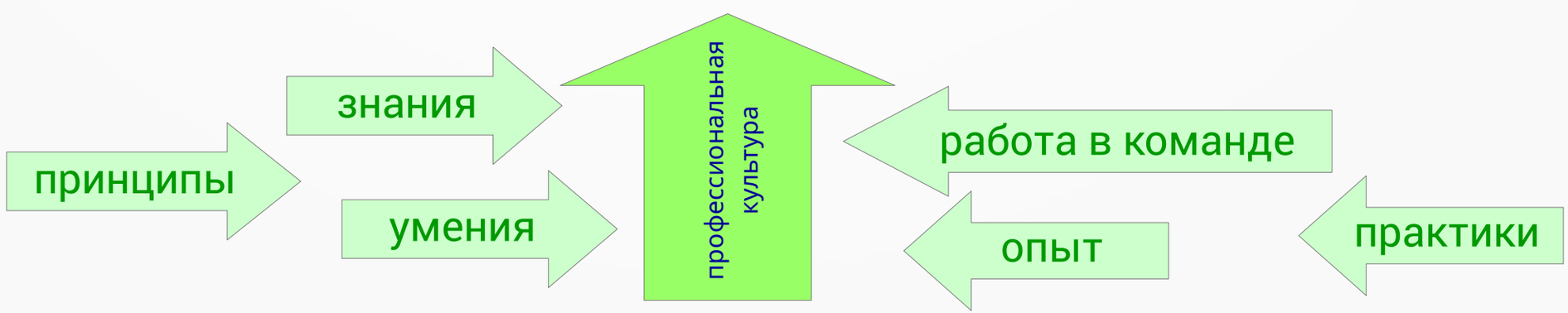
Это не просто провозглашённые принципы

— они давно и успешно применяются на практике!

Технологии: *гибкие* ⟺ тяжеловесные

- *Адаптация* вместо предопределённости.
- *Наращивание* архитектуры, а не её полная проработка.
- Проектирование *эволюционное* вместо предварительного.
- Ясный *исходный код + тесты*, а не объёмная документация.
- Полезный *результат*, а не соблюдение утверждённого проекта.
- Делать только *необходимое*, а не “с запасом на будущее”.
- Ориентация на *человека*, а не на процесс разработки.

Главное – культура и личности разработчиков!



“Гибкие” технологии разработки

- Adaptive Software Development (ASD).
- Agile Unified Process (AUP).
- Crystal Method.
- Disciplined Agile Delivery (DAD).
- Dynamic System Development Method (DSDM).
- Extreme Programming (XP).
- Feature-Driven Development (FDD).
- Getting Real (*for web-based software*← 37signals).
- Lean Software Development (LSD).
- OpenUP (← RUP).
- Rapid Application Development (RAD).
- Scrum (Sprint Continious Rugby Unified Methodology).

agile

“able to move quickly & easily”

- проворный, шустрый,
- быстрый;
- ловкий;
- подвижный, поворотливый;
- быстрореагирующий;
- сообразительный;
- манёвренный;
- энергичный;
- гибкий *(о разработке)*.

X. P. = eXtreme Programming*

4 основные положения X. P.:

I. **Короткий цикл обратной связи** (Fine-Scale Feedback = **FSF**): *за 1 итерацию разработки в 1-4 недели (оптимально 1-2 недели) много не напортачишь.*

II. **Непрерывный процесс** (Continuous Process = **CP**): *соблюдение правил (заказчиками и разработчиками) помогает безостановочно двигаться к цели с известной скоростью.*

III. **Понимание, разделяемое всеми** (Shared Understanding = **SU**): *минимизирует ошибки и сокращает время взаимодействия в команде.*

IV. **Социальная защищённость программиста** (Programmer Welfare = **PW**): *стабильная ритмичная разработка без авралов и переработки.*

** Название методологии исходит из идеи применить полезные традиционные методы и практики разработки программного обеспечения, подняв их на новый «экстремальный» уровень программирования.*

X. P. = eXtreme Programming

12 основных приёмов X.P.:

I. **Короткий цикл обратной связи** (Fine-Scale Feedback): *за 1 итерацию разработки в 1-4 недели (оптимально 1-2 недели) много не напортачишь.*

1. **Заказчик всегда рядом** (Onsite Customer = Whole Team = **WT**): *конечный пользователь (product owner) всегда на связи для вопросов и уточнений; он одобряет план на итерацию.*
2. **Игра в планирование** (Planning Game = **PG**): *направляет разработку продукта через планирование объёма работ и установку приоритетов для пользовательских историй.*
3. **Разработка через тестирование** (Test-Driven Development = **TDD**): *весь код покрыт тестами, код качественный, ему можно доверять; требования выполнены.*
4. **Парное программирование** (Pairing = Pair Programming = **PP**): *коллеги работают совместно, знают код, заменяют друг друга* ← **CCO**. → *Collaborative programming.*

User Story & iterative development

0. Заказчик (как умеет) описывает “пожелания” о задаче, которую должна решать программа и основные требования к ней.
   Это описание заведомо неполное и может содержать противоречивые высказывания.
1. Программисты разделяют описание задачи на короткие «пользовательские истории».
   Каждая «история» – описание простых действий пользователя (в разрабатываемой программе) для достижения конкретного результата.
2. Программисты записывают «истории» на карточки и определяют для каждой из них [относительный] уровень сложности [голосованием по одобренной всеми шкале].
3. Программисты встречаются с заказчиком в начале сессии разработки. Он выбирает (согласовывая с программистами), какие «истории» программисты будут реализовывать в текущей сессии разработки.
4. Программисты в течение сессии реализуют выбранные «истории», тестируют их на соответствие требованиям.
5. Программисты демонстрируют результат разработки заказчику. Заказчик высказывает замечания и предложения, которые оформляются в виде новых «историй».

Разработка движется от сессии к сессии (повторяются шаги со 2 по 5).

Игра в планирование

Игра в планирование – это простой метод, который позволяет создать план реализации проекта.

Шаг 1: Описание или выбор истории.
На этом этапе заказчик должен описать определённую "пользовательскую историю" или выбрать её из своего списка. Разработчики будут эти истории анализировать и определять для них приоритеты. В них будет заключаться основа будущего плана проекта.

Шаг 2: Оценивание истории.
Этот этап проводится в команде разработчиков. Они должны оценить историю в соответствии с критериями времени и стоимости, которые потребуются для её реализации – в относительных абстрактных *Баллах истории*.

Шаг 3: Определение приоритетов для историй.
Это начало планирования проекта. На этом этапе заказчик размещает оцененную историю в план. Всем историям назначаются приоритеты в соответствии с его требованиями к разрабатываемому продукту.

Шаг 4: Процесс повторяется.
Когда определена цена и приоритет для одной истории в плане проекта, процесс повторяется, пока план не будет составлен.

Карточки историй можно располагать на доске kanban.

X. P. = eXtreme Programming

12 основных приёмов X.P.:

II. **Непрерывный процесс** (Continuous Process): *соблюдение правил помогает равномерно двигаться к цели по графику.*

5. **Рефакторинг** (Refactoring = Design Improvement = **DI**): *обязательная реструктуризация кода (без изменения его поведения) поддерживает исходники "в чистоте".*
6. **Частые небольшие релизы** (Small Releases = **SR**): *помогают пользователю оценить функциональность, видеть прогресс, вносить изменения, менять приоритеты.* ← **CI**.
7. **Непрерывная интеграция** (Continuous Integration = **CI**): *частое тестирование и сборка проекта, до нескольких раз в день; готовность к развёртыванию.*

Приёмочное тестирование (Acceptance Tests = **AT**): *автоматизированное тестирование ожидаемой функциональности по тестам заказчика / аналитика.*

X. P. = eXtreme Programming

12 основных приёмов X.P.:

III. **Понимание, разделяемое всеми** (Shared Understanding): *минимизирует ошибки и сокращает время взаимодействия в команде и с заказчиком.*

8. **Метафора системы** (Metaphor = System Metaphor = **SM**): *система однозначно понимаемых терминов, единая для заказчиков и разработчиков* → **DDD** ← **CC**.
9. **Простота проектирования** (Simple Design = **SD**): *выбирается наиболее простой способ реализовать функциональность.*
10. **Стандарт оформления кода** (Coding Standard = Coding Conventions = **CC**): *соблюдаемые* всеми *соглашения о стиле и именованию* → **SM**.
11. **Коллективное владение кодом** (Collective Ownership = Collective Code Ownership = **CCO**): *все разработчики знают исходный код, могут заменять друг друга* ← **PP**.

Simple Design (X. P.)

1. **Covered by Tests** = Покрыть весь код тестами.

   Работает правильно

   Надо стремиться к 100% покрытию строк (line coverage) и покрытию ветвлений (branch coverage). Пригодный для тестирования код — это несвязанный код. Фактически тесты проверяют не только поведение, но и степень несвязанности.

2. **Minimize Duplication** = Минимизировать дублирование кода.

   Без дублирующего кода

   Чем больше дублируется код, тем выше риск хрупкости. Надо исключать дублирование и управлять непреднамеренным дублированием (accidental duplication).

3. **Maximize Expression** = Максимизировать выразительность.

   Понятная для изменений

   В красноречивом (и простом) коде назначение переменных, функций и типов понятно по их именам, в нём легко увидеть структуру алгоритма. Код в совокупности с тестами показывает функцию каждого элемента системы и способ его применения.

4. **Minimize Size** = Минимизировать размер.

   Обозримая, модульная

   После того, как прошли все тесты, максимально красноречиво проявлено предназначение кода и минимизировано дублирование, нужно поработать над уменьшением размеров кода без нарушения трех других принципов.

Архитектура: простота

Лучшая архитектура будет у самого простого проекта, который поддерживает всю необходимую функциональность, одновременно обеспечивая наибольшую гибкость для изменений.

> *«Простой» означает «без запутанных взаимосвязей», «без неправильных зависимостей» между уровнями программной системы.* → ***S. O. L. I. D.***

Прагматичные принципы:

Делай проще, глупец (KISS = Keep It Simple, Stupid): выбирай наиболее простой способ реализовать функциональность.

Тебе это не нужно (YAGNI = You Ain't Gonna Need It): реализовать только необходимое, не закладывать излишнюю ("на будущее") функциональность.

"Сначала сделай, чтобы программа заработала, потом сделай её как положено": это 2 неразрывных действия в одной транзакции разработки (transform + refactor).

X. P. = eXtreme Programming

12 основных приёмов X.P.:

IV. **Социальная защищённость программиста** (Programmer Welfare): *стабильная ритмичная разработка без авралов и переработки.*

12. **Стабильная нагрузка** (Sustainable Pace = **SP**): *40-часовая рабочая неделя (40-hour week), стабильная работоспособность, уверенное планирование.*

Метод “вчерашняя погода”. Команда говорит: “За последнее время мы делали 4-6 фич в неделю, какие 4-6 фич мы будем делать на следующей неделе?” Владелец продукта (product owner) должен выбрать, какие именно пользовательские истории будут реализованы на этой неделе.

Ограничение количества задач метадами WIP-лимитирования: «Сделай больше в будущем, делая меньше прямо сейчас».

X. P. = экстремальная разработка

4 основные положения X. P.:

I. **Короткий цикл обратной связи** (Fine-Scale Feedback = **FSF**): *за 1 итерацию разработки в 1-4 недели (оптимально 1-2 недели) много не напортачишь.*

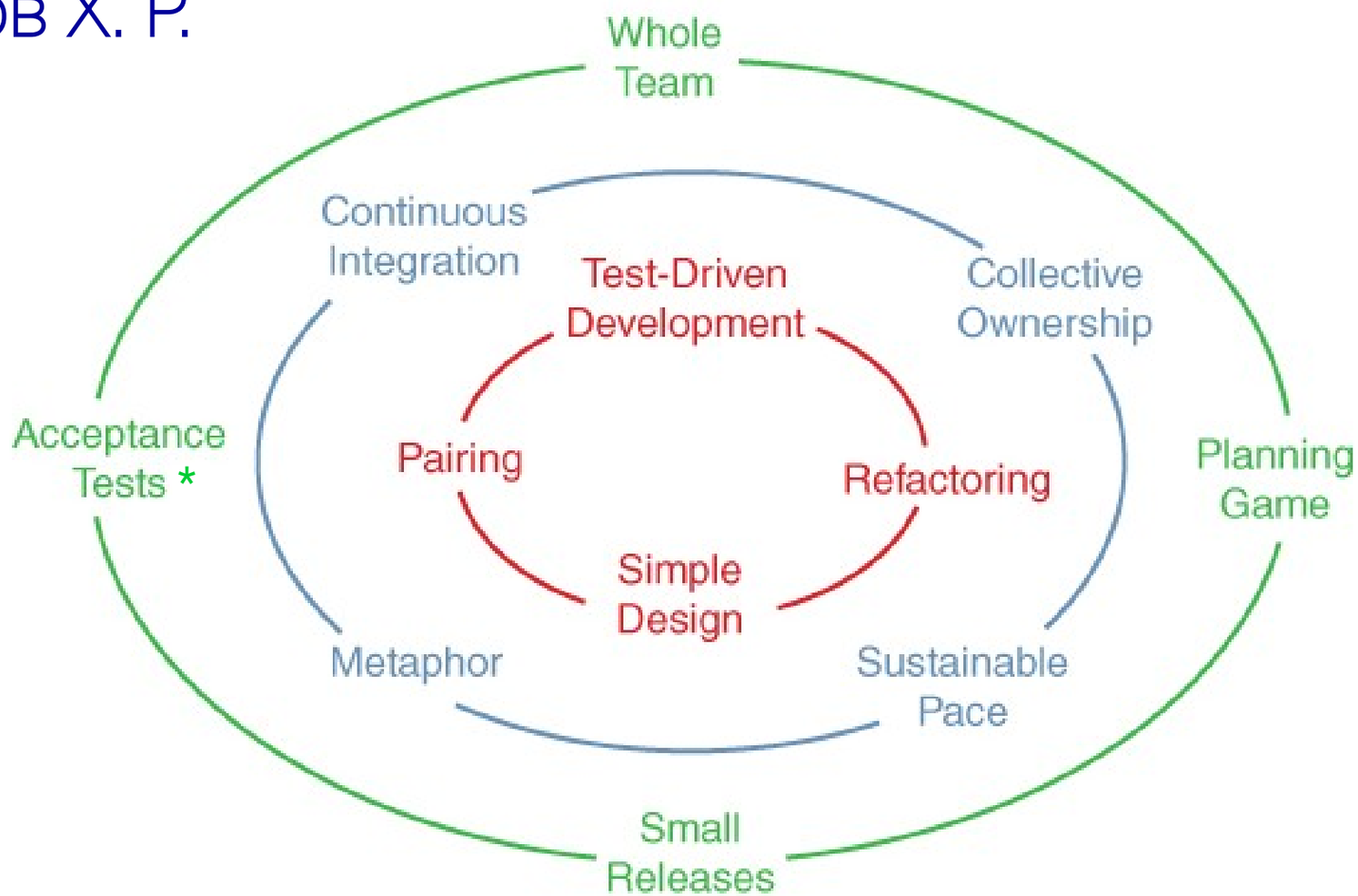
II. **Непрерывный процесс** (Continuous Process = **CP**): *соблюдение правил (заказчиками и разработчиками) помогает безостановочно двигаться к цели с известной скоростью.*

III. **Понимание, разделяемое всеми** (Shared Understanding = **SU**): *минимизирует ошибки и сокращает время взаимодействия в команде.*

IV. **Социальная защищённость программиста** (Programmer Welfare = **PW**): *стабильная ритмичная разработка без авралов и переработки.*

Circle of Life *(Ron Jeffries)*

12 приёмов Х. Р.

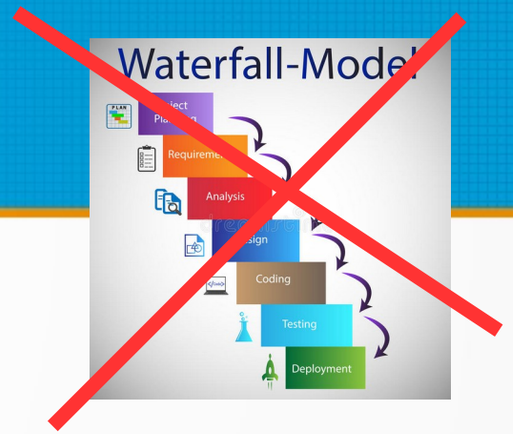


* *Вместо* Coding Conventions

Экстремальное программирование

Непрерывный процесс

Гибкий подход к разработке



Agile

Короткие итерации разработки, во время которых делаются:

Требования: *«истории пользователя» только на эту итерацию.*

Проектирование: *наиболее простая реализация «user stories»*.

Парное программирование *через* тестирование.

Отладка: *практически не требуется при TDD*.

Тестирование: *приёмочное — показать, что требования выполнены*.

Минимальное документирование: *т. к. есть “чистый код” и тесты.*

Внедрение: *в конце каждой итерации продукт готов к работе.*

Сопровождения *нет – есть только разработка!*

Заказчик хочет развития программы – разработчик её адаптирует!

Программа жива, если может меняться

*Заказчики хотят менять **всё**:*

- *выходные отчёты;*
- *форматы данных;*
- *структуру информации;*
- *функциональность программы;*
- *алгоритмы обработки;*
- *используемые устройства;*
- *пользовательский интерфейс;*
- *взаимодействие с внешним ПО;*
- *…*

И это нормально!

“Пользователь не знает, чего он хочет, пока не увидит то, что он получил”.

Эд Йордан

“Процесс автоматизации изменяет взгляд пользователя на объект автоматизации”.

Гленфорд Майерс

“В начале разработки вы можете быть уверенными только в одном – что вы делаете совсем не тот продукт.

Ни один продукт не выдерживает столкновения с пользователем. Как только вы дадите продукт в руки пользователя, вы обнаружите, что сделанный вами продукт не устраивает его по сотне разных причин. И если вы не можете изменить его, не превратив в кашу, то вы обречены”.

Роберт Мартин

Практически изменяемая программа

Программа, котрую **можно изменять**

- в требуемом объёме;
- в запланированное время;
- в рамках имеющегося бюджета;
- без дополнительного персонала;
- без искажения её работы.

Для пользователя критически важна *стоимость внесения изменений* в программную систему (время, люди, деньги).

Продолжительность модификации и стоимость изменений *должны быть пропорциональны объёму требований* пользователя (соответствовать его ожиданиям)*.

* Заказчик ожидает, что незначительные изменения в программе не должны выполняться долго и стоить дорого.

Agile – прагматичный подход

Agile:

- **мы понимаем сложность работы и неопределённость ситуации в течение всей разработки ПО**, поэтому
- **мы научились быстро адаптировать ПО к новым требованиям**, но это возможно при условии,
- **что заказчик доверяет нам** и
- **вовлечён в работу нашей команды.**
- **Только совместно можно сделать качественный продукт.**

Agile – это активное взаимодействие заказчика и разработчиков,
а не противостояние.

Польза от X. P.

X. P.
не догма,
а пример.

Для заказчика / руководства

- Стабильный темп разработки.
- Уверенное планирование.
- Соблюдение сроков разработки.
- Время модификации и стоимость изменений пропорциональны объёму требований.
- Эффективное управление процессом разработки: пользователь видит прогресс, оценивает новую функциональность, оперативно вносит изменения, определяет приоритеты.
- Экономия времени и денег за счёт ненужной функциональности.

Для разработчиков

- Уверенность в качестве кода.
- Полноценная замена временно отсутствующих разработчиков.
- Готовность кода для развёртывания.
- Социальная защищённость программиста.
- Ритмичная разработка без авралов и переработки.
- Стабильная нагрузка на работников.
- Условия для работоспособности и творчества.
- 40-часовая рабочая неделя без переработки.

Адаптивный подход к разработке

Суть Agile **в том,**
чтобы воспитать *команду разработчиков*,
способных вместе эффективно создавать продукт,
адаптируя непрерывный процесс разработки
к изменяющимся условиям и требованиям.
Нет** Agile **без культуры и профессионализма.

Agile – это средство разработчиков
для организации совместной работы (с заказчиком),
а не средство их контроля руководством.

Профессионалы в команде

1. Делать именно то, что нужно

Столкнувшись с запутанными проблемами, ценные сотрудники думают о потребностях своей команды. Они выходят за пределы назначенной им задачи и берутся за ту работу, которую надо сделать на самом деле, сосредоточивать усилия там, где это наиболее полезно..

2. Вести и уступать

Когда ясно, что надо что-то делать, но неясно, кто за это отвечает, ценные сотрудники вмешиваются и возглавляют процесс, а когда задача выполнена, уступают руководство и столь же легко следуют указаниям. Готовность и вести за собой, и подчиняться создаёт в организации культуру смелости, инициативности и гибкости.

3. Доводить до конца

Ценные сотрудники не уступают и полностью завершают работу без постоянного надзора, даже если становится сложно и путь усеян непредвиденными препятствиями.

4. Спрашивать и корректировать

Ценные сотрудники быстрее коллег адаптируются к изменчивым условиям, потому что видят в новых правилах и целях шанс учиться и развиваться с учётом критики. Попутно они способствуют обучению и новаторству в коллективе, помогают организации не отстать от жизни, приобретают репутацию легко обучаемых людей, которые совершенствуются сами и служат примером для всей команды.

5. Облегчать работу другим

Они помогают другим нести груз, но не потому, что берут на себя чужие задачи, а потому, что с ними легко работается. Благодаря им в коллективе появляются жизнерадостность и невозмутимость, становится меньше драматизма, политиканства и стресса, дело спорится. Они создают позитивную, продуктивную рабочую атмосферу, укрепляют культуру сотрудничества, становятся людьми, с которым хочется иметь дело.

Критика Agile *(и возражения)*

- Agile подходит только для небольших команд.
- Принципы Agile неприменимы в больших проектах.
- Agile противоречит строгим срокам, оговорённым в договоре.
- Agile не работает в новых командах (start-ups).

- Большие коллективы делятся на небольшие команды, которые работают по Agile.
- Большой проект можно разрабатывать по Agile, если он хорошо структурирован на под-проекты.
- Agile предполагает реализацию одобренной функциональности в контролируемые заказчиком сроки.
- Новые команды могут работать по Agile, набирая опыт работы и доверие заказчиков.

Agile не будет работать,
если методы и приёмы применяются формально (“только ритуал”),
но не соблюдаются основные принципы.

Agile не работает без честности и свободы обмена мнениями.

А вот это – совсем не Agile!

- Сначала сделаем, как получится, а потом улучшим это.
- Agile хорош тем, что ничего не надо планировать!
- Наймём juniors – пусть пишут по заданиям наших seniors.
- Пусть заказчик формально одобряет всё, что мы предложим.
- Запланируем «с запасом» и будем делать не напрягаясь.
- Будем всё время “допиливать”, чтобы заказчик постоянно платил.
- Если вначале каждого спринта собираемся на scrum-митинги, то мы точно разрабатываем по Agile.
- Будем зарабатывать на обучении методикам Agile!

Готов ответить на вопросы

???

Словарик терминов и сокращений

- agile software development ~ гибкие / адаптивные методологии разработки ПО
- AT = acceptance test ~ приёмочный тест
- BDD = Behaviour-Driven Development ~ разработка, определяемая поведением
- burndown chart ~ диаграмма сгорания задач в текущем спринте
- CCC = Collaboration, Coordination, Communication ~ agile practices
- CCO = collective code ownership ~ совместрое владение кодом (Х. Р.)
- CIS = continuous integration system ~ система непрерывной интеграции
- DDD = Domain-Driven Design ~ предметно-/проблемно-ориентированное проектирование
- DI = dependency inversion ~ инвертирование зависимостей
- DoD = Definition of Done ~ критерий готовности
- DSL = domain-specific language ~ специализированный язык для конкретной предметной области
- FSF = Fine-Scale Feedback ~ короткий цикл обратной связи (в Agile)
- FP = functional programming ~ функциональное программирование
- GoF = Gang of Four ~ «банда четырёх»
- I/O = input/output ~ ввод-вывод
- IDE = integrated development environment ~ интегрированная среда разработки
- Kanban board ~ доска канбан: backlog → to-do → in-process → approved → done
- OOD = object-oriented design ~ объектно-ориентированного проектирование
- OOP = object-oriented programming ~ объектно-ориентированное программирование

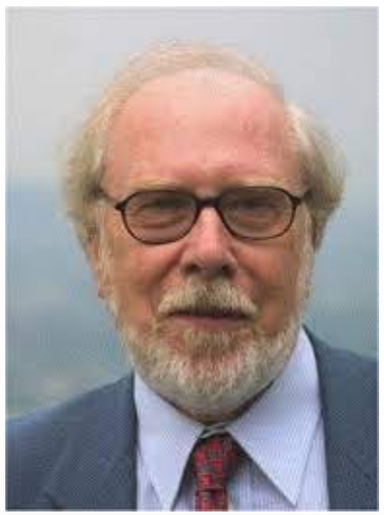
Словарик терминов и сокращений

- PO = product owner ~ владелец продукта
- PP = pair programming ~ парное программирование
- project backlog ~ журнал пожеланий проекта
- QA = quality assurance ~ контроль качества
- SCRUM = Sprint Continious Rugby Unified Methodology
- S.O.L.I.D. ~ 5 основных принципов OOD
- SP = structured programming ~ структурное программирование
- story points ~ относительные оценки трудоёмкости для реализации пожелания
- TDD = Test-Driven Development ~ разработка через тестирование, тест-ориентированная разработка
- TPP = Transformation Priority Premise ~ предположения о приоритетах преобразований
- user stories ~ пользовательские истории – неформальное описание требований к разрабатываемой программе
- UML = Unified Modeling Language ~ унифицированный язык моделирования
- Uncle Bob = Robert Cecil Martin ~ Роберт “дядюшка Боб” Мартин
- UT = unit test ~ модульный тест
- VCS = version control system ~ система управления версиями исходного кода
- WIP-limit = work in progress limit ~ ограничение числа незавершенных задач
- XP = eXtreme Prpgramming

Мои учителя

Phillip Plauger

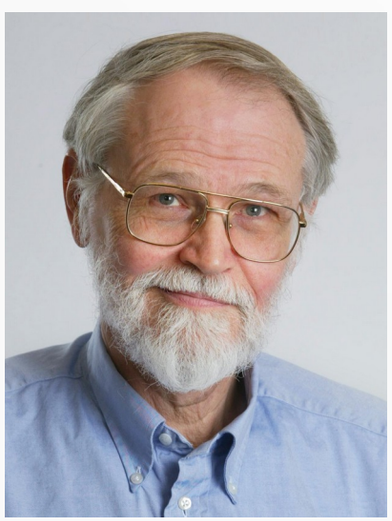
Glenford Myers



Niklaus Wirth

Robert Martin

Kent Beck



Brian Kernighan

Andy Hunt

Dave Thomas

Книги, которые *нужно* прочитать *(IMHO)*

1. Бек К. Экстремальное программирование = **Extreme Programming Explained: Embrace Change** (2004) — СПб: Питер, 2002.
2. Бек К. Экстремальное программирование. Разработка через тестирование = **Test-Driven Development by Example** (2002) — СПб: Питер, 2017.
3. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д. Приемы объектно-ориентированного проектирования. Паттерны проектирования = **Design Patterns: Elements of Reusable Object-Oriented Software** (1994) — СПб: Питер, 2015.
4. Мартин Р. Быстрая разработка программного обеспечения = **Agile Software Development, Principles, Patterns, and Practices** (2002) — М.: Вильямс, 2004.
5. Мартин Р. Идеальный программист = **The Clean Coder. A Code of Conduct for Professional Programmers** (2011) — СПб: Питер, 2012.
6. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения = **Clean Architecture. A Craftsman's Guide to Software Structure and Design** (2017) — СПб: Питер, 2018.
7. Мартин Р. Чистый код. Создание, анализ и рефакторинг = **Clean Code. A Handbook of Agile Software Craftsmanship** (2009) — СПб: Питер, 2018.
8. Мартин Р. Чистый Agile. Основы гибкости = **Clean Agile. Back to Basics** (2020) — СПб.: Питер, 2020.
9. Мартин Р. Идеальная работа. Программирование без прикрас = **Clean Craftsmanship: Disciplines, Standards, and Ethics** (2021) — СПб.: Питер, 2022.

Книги, которые *стоит* прочитать *(IMHO)*

10. Метц С. Ruby. Объектно-ориентированное проектирование = **Practical Object-Oriented Design. An Agile Primer Using Ruby** (2012) — СПб.: Питер, 2017.
11. Фаулер М. Рефакторинг. Улучшение проекта существующего кода = **Refactoring. Improving the Design of Existing Code**. — М.: Диалектика, 2019.
12. Томас Д., Хант Э. Программист-прагматик. Путь от подмастерья к мастеру = **The Pragmatic Programmer: From Journeyman to Master**. — М.: Лори, 2004.
13. Майерс Г. Надёжность программного обеспечения = **Software Reliability: Principles and Practices**. — М.: Мир, 1980.
14. Майерс Г. Искусство тестирования программ = **The Art Of Software Testing**. — М. Финансы и статистика, 1982; 3-е изд. — М., СПб.: Диалектика, 2019.
15. Ван Тассел Д. Стиль, разработка, эффективность, отладка и испытание программ = **Program Style, Design, Efficiency, Debuggind And Testing**.— М.: Мир, 1981.
16. Вирт Н. Алгоритмы + структуры данных = программы = **Algorithms + Data Structures = Programs**. — М.: Мир, 1985.
17. Керниган Б., Плоджер Ф. Инструментальные средства программирования на языке Pascal = **Software Tools in Pascal**. — М.: Радио и связь, 1985.
18. Буч Г. Объектно-ориентированное проектирование с примерами применения = **Object-Oriented Design with Applications**. — М.: Конкорд, 1992.

Полезные ссылки

- https://basecamp.com/gettingreal # “Getting Real” @ 37 Signals
- https://it.wikireading.ru/37243 # “Getting Real” на русском
- https://en.wikipedia.org/wiki/Software_design_pattern # Образцы проектирования
- https://blog.cleancoder.com/uncle-bob/2014/06/30/ALittleAboutPatterns.html
- https://en.wikipedia.org/wiki/Kata_(programming)
- https://habr.com/ru/post/171883/ #Стартап-ловушка
- https://medium.com/@pablo127/effective-estimation-review-of-uncle-bobs-presentation-a2150f5f68ac
- https://codingjourneyman.com/2014/10/06/the-clean-coder-estimation/ # PERT
- https://ru.wikipedia.org/wiki/Покер_планирования # Относительное планирование
- https://8thlight.com/blog/micah-martin/2012/11/17/transformation-priority-premise-applied.html
- https://habr.com/ru/company/piter/blog/427853/ # Размышления о TDD
- https://habr.com/ru/post/206828/ # Test-Driven Development — телега или лошадь?
- https://software-testing.ru/library/5-testing/72---tdd-- # Ошибки начинающих TDD-практиков
- https://habr.com/ru/post/459620/ # Всё, что вы хотите узнать о Driven Development
- https://habr.com/ru/company/edison/blog/313410/ # Как объяснить бабушке, что такое Agile
- https://habr.com/ru/post/131926/ # Почему Agile вам не подходит
- https://habr.com/ru/post/695554/ # У вас не Agile
- https://worksection.com/blog/work-in-progress.html # WIP-лимит в жизни, в работе, в семье
- https://habr.com/ru/post/352282/ # Continuous Integration для новичков
- https://habr.com/ru/company/southbridge/blog/691782/ # CI/CD: как, зачем, для чего
- https://lifehacker.ru/cennyj-sotrudnik/ # 5 установок, которые отличают ценных сотрудников от обычных