**SDLC**

**Software Development Lifecycle**

**Жизненный цикл разработки ПО**

Компьютерное программное обеспечение или просто **программное обеспечение** - это набор данных или компьютерных инструкций, которые говорят компьютеру, как работать.

**Виды ПО:**

Desktop (Word, Excel, Chrome, Safari, Photoshop, Starcraft)

Web (Facebook, YouTube, Amazon, Netflix)

Mobile (Pokemon Go, SnapChat, Instagram, Uber)

Industrial hardware programs (Robots, Networks, industrial machines)

**Разработка программного обеспечения** — это род деятельности и процесс, направленный на создание и поддержание работоспособности, качества и надежности программного обеспечения, используя технологии, методологию и практики из информатики, управления проектами, математики, инженерии и других областей знания.

**SDLC** - это процесс в индустрии разработки ПО, в который входит дизайн, разработка и тестирование ПО, которое соответствует ожиданиям заказчика, выполняется в срок и запланированные трудозатраты. Он состоит из детального плана, описывающего, как разрабатывать, поддерживать, менять или улучшать ПО.

**SDLC** - период времени, который начинается с момента принятия решения о необходимости создания ПО и заканчивается в момент полного изъятия из эксплуатации.

**Этапы SDLC**

1. Идея
2. Planning and Requirement Analysis (планирование и анализ требований): участники – product owner, менеджеры по продажам, бизнес аналитики, product managers; анализ и планирование рисков, затрат, определение технических подходов.
3. Defining Requirements (определение требований): создание документации и утверждение от заказчика, прописывание целей, определение задач, создание спецификации требований к ПО (SRS – Software Requirement Specification).
4. Designing the Product Architecture (разработка архитектуры проекта): на основе SRS разрабатывается DDS – Design Document Specification.
5. Building and Developing the Product (создание и разработка ПО): фактическая разработка и сборка – программный код, фичи, модули на основе DDS.
6. Testing the Product (тестирование продукта): регистрация, отслеживание и исправление багов.
7. Deployment in the Market and Maintenance (ввод в эксплуатацию и поддержка): релиз в ограниченном сегменте, сбор отзывов, разработка обновлений.
8. Вывод из эксплуатации.

***МОДЕЛИ SDLC***

**1. Waterfall model.**

1) Самый ранний подход SDLC, который использовался для разработки программного обеспечения.

2) Модель водопада иллюстрирует процесс разработки программного обеспечения в линейном последовательном потоке. Это означает, что любой этап в процессе разработки начинается, только если предыдущий этап завершен. В этой модели водопада фазы не перекрываются.

3) В этой модели водопада, как правило, результат одной фазы действует как вход для следующей фазы последовательно.

**Requirements 🡪**

**Analysis 🡪**

**Design 🡪**

**Coding 🡪**

**Testing 🡪**

**Operations**

**Waterfall model - фазы.**

1) Сбор и анализ требований – Все возможные требования к разрабатываемой системе фиксируются на этом этапе и документируются в документе спецификации требований.

2) Проектирование системы. На этом этапе изучаются требования к требованиям первого этапа и готовится проектирование системы. Такая конструкция системы помогает определить требования к оборудованию и системе и помогает определить общую архитектуру системы.

3) Внедрение. С учетом исходных данных, полученных при проектировании системы, система сначала разрабатывается в виде небольших программ, называемых модулями, которые интегрируются на следующем этапе. Каждое устройство разработано и проверено на его функциональность, которая называется модульным тестированием.

4) Интеграция и тестирование – все модули, разработанные на этапе внедрения, интегрируются в систему после тестирования каждого модуля. После интеграции вся система проверяется на наличие ошибок и сбоев.

5) Развертывание системы – после завершения функционального и нефункционального тестирования; продукт развернут в среде клиента или выпущен на рынок.

6) Обслуживание – Есть некоторые проблемы, которые возникают в клиентской среде. Чтобы исправить эти проблемы, исправления выпущены. Также для улучшения продукта выпущены лучшие версии. Техническое обслуживание проводится для предоставления этих изменений в среде клиента.

**Waterfall model - Ситуации использования.**

1) Требования очень хорошо задокументированы, понятны и зафиксированы.

2) Определение продукта является стабильным.

3) Технология понятна и не динамична.

4) Здесь нет двусмысленных требований.

5) Достаточные ресурсы с необходимым опытом доступны для поддержки продукта.

6) Проект короткий.

**Waterfall model - Преимущества.**

1) Просто и легко понять и использовать

2) Простота в управлении благодаря жесткости модели. Каждый этап имеет конкретные результаты и процесс обзора.

3) Фазы обрабатываются и завершаются по одному.

4) Хорошо работает для небольших проектов, где требования очень хорошо поняты.

5) Четко определенные этапы.

6) Хорошо понятные вехи.

7) Легко организовать задачи.

8) Процесс и результаты хорошо документированы.

**Waterfall model - Недостатки.**

1) Никакое рабочее программное обеспечение не производится до конца жизненного цикла.

2) Большое количество риска и неопределенности.

3) Не очень хорошая модель для сложных и объектно-ориентированных проектов.

4) Плохая модель для длительных и текущих проектов.

5) Не подходит для проектов, где требования изменяются от умеренного до высокого риска. Таким образом, риск и неопределенность высоки с этой моделью процесса.

6) Трудно измерить прогресс в несколько этапов.

7) Невозможно учесть меняющиеся требования

8) Регулировка объема в течение жизненного цикла может завершить проект.

9) Интеграция осуществляется как «большой взрыв» в самом конце, что не позволяет выявить какие-либо технологические или бизнес-узкие места или проблемы на ранних этапах.

**2. Iteration model.**

1) В итерационной модели итеративный процесс начинается с простой реализации небольшого набора требований к программному обеспечению и итеративно расширяет развивающиеся версии, пока вся система не будет внедрена и готова к развертыванию.

2) Модель итеративного жизненного цикла не пытается начать с полной спецификации требований.

3) Разработка начинается с определения и внедрения только части программного обеспечения, которое затем анализируется для определения дальнейших требований. Затем этот процесс повторяется, создавая новую версию программного обеспечения в конце каждой итерации модели.

**Build1**

**Design&Development --- Testing --- Implementation**

**Requirements Build2**

**Design&Development --- Testing --- Implementation**

**Build3**

**Design&Development --- Testing --- Implementation**

**Iterative model - Ситуации использования.**

1) Требования к полной системе четко определены и понятны.

2) Основные требования должны быть определены; однако некоторые функции или запрошенные улучшения могут со временем развиваться.

3) Есть время для ограничения рынка.

4) Новая технология используется и изучается командой разработчиков во время работы над проектом.

5) Ресурсы с необходимыми наборами навыков недоступны и планируется использовать на контрактной основе для конкретных итераций.

6) Есть некоторые особенности и цели высокого риска, которые могут измениться в будущем.

**Iterative model - Преимущества.**

1) Некоторая рабочая функциональность может быть разработана быстро и в начале жизненного цикла.

2) Результаты получены рано и периодически.

3) Параллельное развитие может быть запланировано.

4) Прогресс можно измерить.

5) Менее затратно изменить сферу / требования.

6) Тестирование и отладка во время небольшой итерации очень просты.

7) Риски выявляются и устраняются в ходе итерации; и каждая итерация является легко управляемой вехой.

8) Проще управлять риском – сначала выполняется часть с высоким риском.

9) После каждой итерации поставляется новая версия продукта

10) Проблемы и риски, определенные для каждого приращения, могут быть использованы / применены к следующему приращению.

11) Лучше подходит для крупных и критически важных проектов.

**Iterative model - Недостатки.**

1) Может потребоваться больше ресурсов.

2) Требуется больше внимания руководства.

3) Могут возникнуть проблемы с архитектурой или дизайном системы, поскольку не все требования собраны в начале всего жизненного цикла.

4) Не подходит для небольших проектов.

5) Сложность управления больше.

6) Конец проекта может быть неизвестен, что является риском.

**3. V-model.**

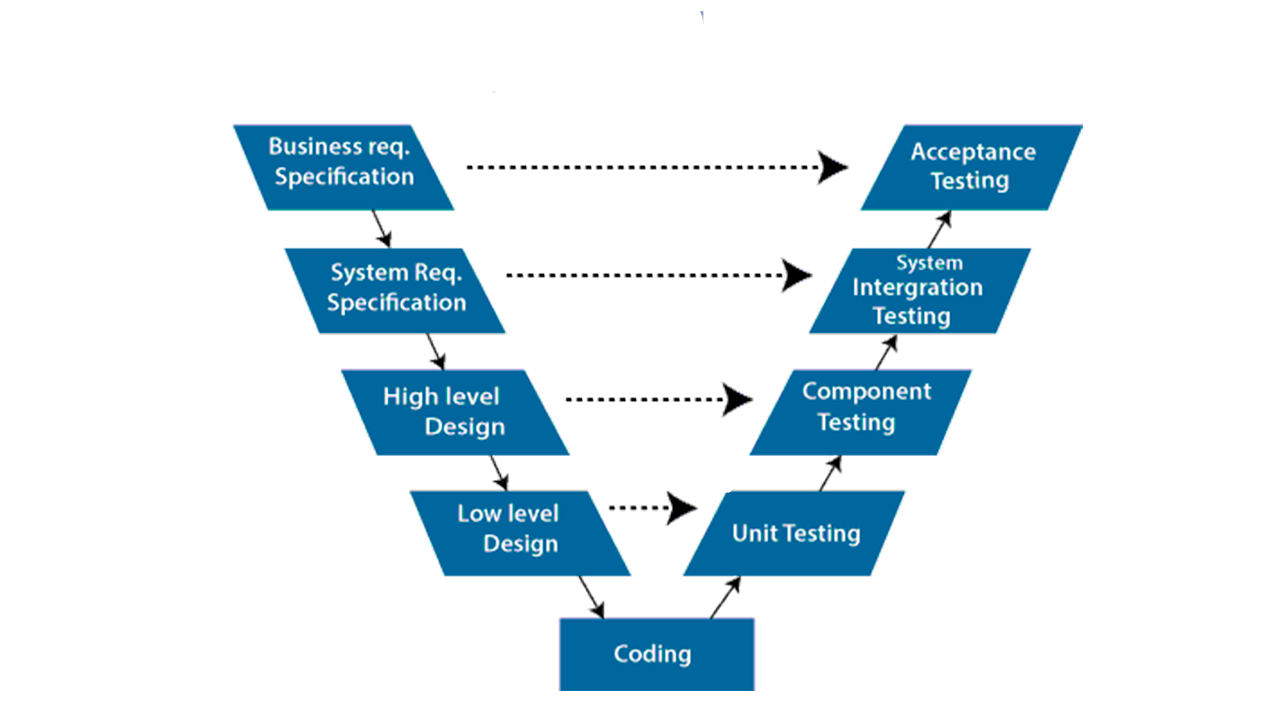
1) V-модель – это модель SDLC, в которой выполнение процессов происходит последовательно в форме буквы V. Он также известен как модель верификации и валидации.

2) V-модель является расширением модели водопада и основана на связи фазы тестирования для каждой соответствующей стадии разработки. Это означает, что для каждой отдельной фазы в цикле разработки существует непосредственно связанная фаза тестирования. Это очень дисциплинированная модель, и следующий этап начинается только после завершения предыдущего этапа.

3) В рамках V-модели соответствующая фаза тестирования фазы разработки планируется параллельно.

**Developer’s Lifecycle Tester’s Lifecycle**

**Verification Phase Validation Phase**



**V-model - Этапы верификаций**

**1) Анализ бизнес-требований.**

Это первая фаза в цикле разработки, когда требования к продукту понимаются с точки зрения клиента. Этот этап включает в себя подробное общение с клиентом, чтобы понять его ожидания и точные требования. Это очень важный вид деятельности, которым нужно хорошо управлять, так как большинство клиентов не уверены, что именно им нужно. Планирование проекта приемочных испытаний выполняется на этом этапе, поскольку бизнес-требования могут использоваться в качестве входных данных для приемочных испытаний.

**2) Системный дизайн**

Когда у вас есть четкие и подробные требования к продукту, пришло время разработать полную систему. Проект системы будет иметь понимание и детализацию полной аппаратной и коммуникационной настройки для разрабатываемого продукта. План тестирования системы разрабатывается на основе проектирования системы. Выполнение этого на более ранней стадии оставляет больше времени для фактического выполнения теста позже.

**3) Архитектурный дизайн - HLD**

Архитектурные спецификации поняты и разработаны на этом этапе. Обычно предлагается более одного технического подхода, и на основе технической и финансовой осуществимости принимается окончательное решение. Проект системы далее разбит на модули, выполняющие различные функции. Это также называется дизайном высокого уровня (HLD)

Передача данных и связь между внутренними модулями и внешним миром (другими системами) четко поняты и определены на этом этапе. С этой информацией интеграционные тесты могут быть разработаны и задокументированы на этом этапе.

**4) Модульный дизайн - LLD**

На этом этапе указывается подробный внутренний дизайн для всех системных модулей, называемый Низкоуровневым проектированием (LLD) . Важно, чтобы проект был совместим с другими модулями в архитектуре системы и другими внешними системами. Модульные тесты являются неотъемлемой частью любого процесса разработки и помогают устранить максимальные ошибки и ошибки на самой ранней стадии. Эти модульные тесты могут быть разработаны на этом этапе на основе внутренних конструкций модулей.

**5) Кодирование**

Фактическое кодирование системных модулей, разработанных на этапе проектирования, рассматривается на этапе кодирования. Выбор наиболее подходящего языка программирования определяется на основе системных и архитектурных требований.

Кодирование выполняется на основе руководящих принципов и стандартов кодирования. Код проходит многочисленные обзоры кода и оптимизируется для лучшей производительности, прежде чем окончательная сборка будет проверена в хранилище.

**V-model - Этапы валидации**

**1) Модульное тестирование**

Модульные тесты, разработанные на этапе проектирования модуля, выполняются в коде на этом этапе проверки. Модульное тестирование – это тестирование на уровне кода, которое помогает устранить ошибки на ранней стадии, хотя все дефекты не могут быть обнаружены модульным тестированием.

**2) Интеграционное тестирование**

Интеграционное тестирование связано с этапом архитектурного проектирования. Интеграционные тесты выполняются для проверки сосуществования и связи внутренних модулей в системе.

**3) Системное тестирование**

Тестирование системы напрямую связано с фазой проектирования системы. Системные тесты проверяют всю функциональность системы и связь разрабатываемой системы с внешними системами. Большинство проблем совместимости программного и аппаратного обеспечения могут быть обнаружены во время выполнения этого теста системы.

**4) Приемочное тестирование**

Приемочное тестирование связано с фазой анализа бизнес-требований и включает тестирование продукта в пользовательской среде. Приемочные тесты раскрывают проблемы совместимости с другими системами, доступными в пользовательской среде. Он также обнаруживает нефункциональные проблемы, такие как загрузка и дефекты производительности в реальной пользовательской среде.

**V-model – Применение**

Применение V-модели практически совпадает с моделью водопада, поскольку обе модели имеют последовательный тип. Требования должны быть очень четкими до начала проекта, потому что возвращение и внесение изменений обычно обходится дорого. Эта модель используется в области медицинского развития, поскольку является строго дисциплинированной областью.

Следующие указатели являются одними из наиболее подходящих сценариев для использования приложения V-Model.

1. Требования четко определены, четко задокументированы и зафиксированы.

2. Определение продукта является стабильным.

3. Технология не динамична и хорошо понимается командой проекта.

4. Здесь нет двусмысленных или неопределенных требований.

5. Проект короткий.

**Преимущество** метода V-Model заключается в том, что его очень легко понять и применить. Простота этой модели также облегчает управление. **Недостатком** является то, что модель не является гибкой к изменениям, и на случай изменения требований, которое очень распространено в современном динамичном мире, внесение изменений становится очень дорогим.

**V-model - Преимущества**

1. Это очень дисциплинированная модель, и Фазы завершаются по одному.

2. Хорошо работает для небольших проектов, где требования очень хорошо поняты.

3. Просто и легко понять и использовать.

4. Простота в управлении благодаря жесткости модели. Каждый этап имеет конкретные результаты и процесс обзора.

**V-model - Недостатки**

1. Высокий риск и неопределенность.

2. Не очень хорошая модель для сложных и объектно-ориентированных проектов.

3. Плохая модель для длительных и текущих проектов.

4. Не подходит для проектов, где требования изменяются от умеренного до высокого риска.

5. Как только приложение находится в стадии тестирования, трудно вернуться назад и изменить функциональность.

6. Никакое рабочее программное обеспечение не производится до конца жизненного цикла.

**4. Модель большого взрыва.**

Модель Большого взрыва – это модель SDLC, в которой мы не следуем никаким конкретным процессам. Разработка только начинается с необходимых денег и усилий в качестве входных данных, а результатом является разработанное программное обеспечение, которое может соответствовать или не соответствовать требованиям заказчика. Эта модель Большого взрыва не соответствует процессу / процедуре, и требуется очень мало планирования. Даже заказчик не уверен, что именно он хочет, и требования выполняются на лету без особого анализа.

Эта модель идеально подходит для небольших проектов с одним или двумя разработчиками, работающими вместе, и также полезна для академических или практических проектов. Это идеальная модель для продукта, где требования недостаточно понятны, а дата окончательного выпуска не указана.

**Time**

**Efforts BIG BANG SOFTWARE**

**Resources**

**Модель большого взрыва - Применение**

1. Модель Большого взрыва включает в себя сосредоточение всех возможных ресурсов на разработке и кодировании программного обеспечения, практически без планирования.

2. Требования поняты и реализованы по мере их поступления.

3. Любые необходимые изменения могут или должны обновлять программное обеспечение полностью.

**Модель большого взрыва - Преимущества**

1. Это очень простая модель.

2. Мало или не требуется планирование.

3. Прост в управлении.

4. Требуется очень мало ресурсов.

5. Предоставляет разработчикам гибкость

6. Это хорошее учебное пособие для новичков или студентов.

**Модель большого взрыва – Недостатки**

1. Очень Высокий риск и неопределенность.

2. Не очень хорошая модель для сложных и объектно-ориентированных проектов.

3. Плохая модель для длительных и текущих проектов.

4. Может оказаться очень дорогим, если требования неправильно поняты.

**5. Спиральная модель.**

«Спиральная модель» похожа на инкрементную, но с акцентом на анализ рисков. Она хорошо работает для решения критически важных бизнес-задач, когда неудача несовместима с деятельностью компании, в условиях выпуска новых продуктовых линеек, при необходимости научных исследований и практической апробации.

Каждый виток спирали соответствует созданию фрагмента или версии программного обеспечения, на нём уточняются цели и характеристики проекта, определяется его качество и планируются работы следующего витка спирали. Таким образом углубляются и последовательно конкретизируются детали проекта и в результате выбирается обоснованный вариант, который доводится до реализации.

**Спиральная модель предполагает 4 этапа для каждого витка:**

Планирование;

Анализ рисков;

Конструирование;

Оценка результата и при удовлетворительном качестве переход к новому витку.

На каждом витке спирали могут применяться разные модели процесса разработки ПО. В конечном итоге на выходе получается готовый продукт.

Разработка итерациями отражает объективно существующий спиральный цикл создания системы. Неполное завершение работ на каждом этапе позволяет переходить на следующий этап, не дожидаясь полного завершения работы на текущем. При итеративном способе разработки недостающую работу можно будет выполнить на следующей итерации.

**Предусматриваемые риски.**

1. Дефицит специалистов.

2. Нереалистичные сроки и бюджет.

3. Реализация несоответствующей функциональности.

4. Разработка неправильного пользовательского интерфейса.

5. «Золотая сервировка», перфекционизм, ненужная оптимизация и оттачивание деталей.

6. Непрекращающийся поток изменений.

7. Нехватка информации о внешних компонентах, определяющих окружение системы или вовлечённых в интеграцию.

8. Недостатки в работах, выполняемых внешними (по отношению к проекту) ресурсами.

9. Недостаточная производительность получаемой системы.

10. Разрыв между квалификацией специалистов и требованиями проекта

**Главная** **задача** — как можно быстрее показать пользователям системы работоспособный продукт, тем самым активизируя процесс уточнения и дополнения требований. Основная проблема спирального цикла — определение момента перехода на следующий этап. Для её решения необходимо ввести временные ограничения на каждый из этапов жизненного цикла. Переход осуществляется в соответствии с планом, даже если не вся запланированная работа закончена. План составляется на основе статистических данных, полученных в предыдущих проектах, и личного опыта разработчиков.

**6. Модель RAD - Rapid Application Development**

Быстрая разработка приложений – это методология разработки программного обеспечения, которая использует минимальное планирование в пользу быстрого прототипирования.

**Прототип** – это рабочая модель, функционально эквивалентная компоненту продукта.

**Decide Build Evaluate with Prototype good**

**Requirements Prototype Customers Enough?**

**NO YES**

**Improve Prototype Prototype Becomes**

**Final Product**

В модели RAD функциональные модули разрабатываются параллельно как прототипы и объединяются, чтобы сделать полный продукт для более быстрой доставки продукта. Поскольку детального предварительного планирования нет, это облегчает включение изменений в процесс разработки.

**Применение.**

1. RAD следует использовать только в том случае, если система может быть модульной, чтобы доставлять ее постепенно.

2. Его следует использовать, если существует высокая доступность дизайнеров для моделирования.

3. Его следует использовать только в том случае, если бюджет допускает использование инструментов автоматической генерации кода.

4. Модель RAD SDLC следует выбирать только в том случае, если имеются специалисты в предметной области, обладающие соответствующими деловыми знаниями.

5. Должен использоваться, когда требования изменяются во время проекта, а рабочие прототипы должны быть представлены заказчику небольшими итерациями в течение 2-3 месяцев.

**Преимущества**.

1. Изменение требований могут быть учтены.

2. Прогресс можно измерить.

3. Время итерации может быть коротким с использованием мощных инструментов RAD.

4. Производительность с меньшим количеством людей за короткое время.

5. Сокращение времени разработки.

6. Увеличивает возможность повторного использования компонентов.

7. Быстрые начальные обзоры происходят.

8. Поощряет обратную связь с клиентами.

9. Интеграция с самого начала решает множество проблем интеграции.

**Недостатки.**

1. Зависимость от технически сильных членов команды для определения бизнес-требований.

2. Только система, которая может быть модульной, может быть построена с использованием RAD.

3. Требуются высококвалифицированные разработчики / дизайнеры.

4. Высокая зависимость от навыков моделирования.

5. Неприменимо к более дешевым проектам, так как стоимость моделирования и автоматической генерации кода очень высока.

6. Сложность управления больше.

7. Подходит для систем, основанных на компонентах и ​​масштабируемых.

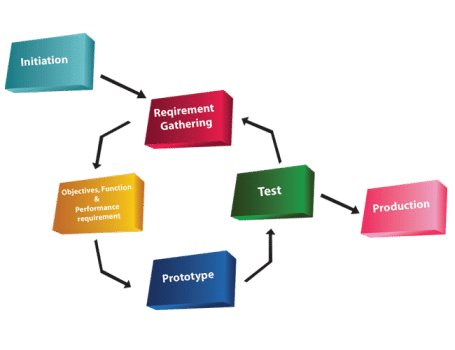
8. Требует участия пользователя на протяжении всего жизненного цикла.

9. Подходит для проекта, требующего более коротких сроков разработки.

**7. Модель прототипного обеспечения.**

**Prototype** – это рабочая модель программного обеспечения с некоторыми ограниченными функциональными возможностями. Прототип не всегда содержит точную логику, используемую в реальном программном приложении, и требует дополнительных усилий при оценке усилий.

Прототипирование используется для того, чтобы пользователи могли оценить предложения разработчиков и опробовать их перед внедрением. Это также помогает понять требования, которые специфичны для пользователя и могут не учитываться разработчиком при разработке продукта.



**1. Тип - Throwaway / быстрое прототипирование**

Одноразовое прототипирование также называется быстрым или близким прототипированием. Этот тип прототипирования требует очень мало усилий с минимальным анализом требований для создания прототипа. Как только фактические требования понятны, прототип отбрасывается, и фактическая система разрабатывается с очень четким пониманием требований пользователя.

**2. Тип - Эволюционное прототипирование**

Эволюционное прототипирование, также называемое **макетом**, основано на создании реальных функциональных прототипов с минимальными функциональными возможностями в начале. Разработанный прототип является сердцем будущих прототипов, на основе которых построена вся система. Используя эволюционное прототипирование, хорошо понятные требования включаются в прототип, а требования добавляются по мере их понимания.

**3. Тип - Инкрементальное прототипирование**

Инкрементное прототипирование относится к созданию нескольких функциональных прототипов различных подсистем и последующей интеграции всех доступных прототипов в единую систему.

**4. Тип - Экстремальное прототипирование**

Экстремальное прототипирование используется в области веб-разработки. Он состоит из трех последовательных фаз. Во-первых, базовый прототип со всеми существующими страницами представлен в формате HTML. Затем обработка данных моделируется с использованием уровня прототипов сервисов. Наконец, службы реализованы и интегрированы в окончательный прототип.

**Преимущества.**

1. Повышенная вовлеченность пользователей в продукт еще до его внедрения.

2. Поскольку отображается рабочая модель системы, пользователи получают лучшее представление о разрабатываемой системе.

3. Сокращает время и затраты, поскольку дефекты могут быть обнаружены намного раньше.

4. Доступна более быстрая обратная связь с пользователем, что приводит к лучшим решениям.

5. Отсутствующие функциональные возможности могут быть легко идентифицированы.

6. Запутанные или сложные функции могут быть определены.

**Недостатки**.

1. Риск недостаточного анализа требований из-за слишком большой зависимости от прототипа.

2. Пользователи могут запутаться в прототипах и реальных системах.

3. Практически, эта методология может увеличить сложность системы, поскольку область действия системы может выйти за рамки первоначальных планов.

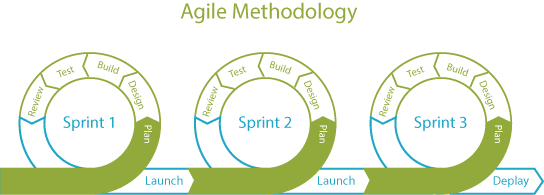
4. Разработчики могут попытаться повторно использовать существующие прототипы для создания реальной системы, даже если это технически неосуществимо.

5. Усилия, вложенные в создание прототипов, могут быть слишком большими, если они не контролируются должным образом.

1. **Модель Agile**
2. Модель Agile SDLC представляет собой комбинацию итеративных и инкрементальных моделей процессов с акцентом на адаптивность процессов и удовлетворенность клиентов за счет быстрой доставки работающего программного продукта.
3. Agile Methods разбивает продукт на небольшие инкрементальные сборки.
4. Сборки предоставляются в итерациях.
5. Каждая итерация обычно длится от одной до трех недель.

Agile основан на адаптивных методах разработки ПО, в то время как традиционные модели SDLC, такие как модель водопада, основаны на прогнозном подходе. Прогнозирующие группы в традиционных моделях SDLC обычно работают с подробным планированием и имеют полный прогноз точных задач и функций, которые должны быть выполнены в течение следующих нескольких месяцев или в течение жизненного цикла продукта. Методы прогнозирования полностью зависят от анализа требований и планирования, выполненного в начале цикла. Любые изменения, подлежащие включению, проходят строгий контроль и управление изменениями.

Agile использует адаптивный подход, когда нет детального планирования и ясность будущих задач только в отношении того, какие функции необходимо разработать. Существует функционально-ориентированная разработка, и команда динамично адаптируется к изменяющимся требованиям к продукту. Продукт тестируется очень часто с помощью итераций выпуска, что сводит к минимуму риск возникновения серьезных сбоев в будущем.



Взаимодействие с клиентами является основой этой методологии Agile, а открытое общение с минимальной документацией – типичные особенности среды разработки Agile. Agile команды работают в тесном сотрудничестве друг с другом и чаще всего расположены в одном географическом месте.

**Модель Agile - преимущества.**

1. Реалистичный подход к разработке программного обеспечения.

2. Способствует командной работе.

3. Функциональность может быстро развиваться и демонстрироваться.

4. Требования к ресурсам минимальны.

5. Подходит для фиксированных или меняющихся требований.

6. Поставляет ранние частичные рабочие решения.

7. Хорошая модель для сред, которые постоянно меняются.

8. Минимальные правила, документация легко используется.

9. Обеспечивает одновременную разработку и доставку в общем запланированном контексте.

10. Мало или нет планирования не требуется.

11. Прост в управлении.

12. Дает гибкость разработчикам.

**Модель Agile - недостатки.**

1. Не подходит для обработки сложных зависимостей.

2. Общий план, проворный лидер и проворная практика PM – необходимость, без которой это не будет работать.

3. Строгое управление доставкой диктует объем, функциональность, которую необходимо доставить, и корректировки для соблюдения сроков.

4. В значительной степени зависит от взаимодействия с клиентами, поэтому, если клиент не ясно, команда может двигаться в неправильном направлении.

5. Существует очень высокая индивидуальная зависимость, так как генерируется минимум документации.

6. Передача технологии новым членам команды может быть довольно сложной из-за отсутствия документации.

Эта модель предполагает, что при реализации проекта не нужно опираться только на заранее созданные планы. Важно ориентироваться на меняющиеся условия внешней и внутренней среды и учитывать обратную связь от заказчиков и пользователей. Это поощряет разработчиков экспериментировать и искать новые решения, не ограничивать себя жесткими рамками и стандартами.

Каждый участник знает, что он сегодня делает и что уже сделал, какие были проблемы (standup meeting).

Эта модель максимально ориентирована на людей, адаптируется к изменениям проектной ситуации, оказывается от долгосрочного планирования, максимально тесно взаимодействует с заказчиком.

**Agile-подходы**

**SCRUM**

Подход «структуры»; цель – закончить спринт; рабочий процесс делится на равные **спринты** – периоды от недели до месяца в зависимости от проекта и команды; над каждым проектом работает **универсальная** **команда** специалистов, к которым присоединяются владелец продукта (**product** **owner**) и **scrum**-**мастер** (душа команды).

Scrum – это автобус, который останавливается на определенных остановках, где люди выходят группами.

**KANBAN**

Подход «баланса»; цель – закончить задачу; рабочий процесс делится на стадии выполнения **конкретных** **задач**: планируется, разрабатывается, тестируется, завершено и т.д.; вся **команда** **едина**; **нет** **ролей** владельца продукта и scrum-мастера.

Kanban – это маршрутка, где пассажир выходит там, где ему нужно.