**Основы инженерии программных систем. Вопросы к тесту. 2017-18**

1. **Понятие программной инженерии.**

“Software engineering”

Приложение систематического, дисциплинированного и измеримого подхода к развитию, функционированию и сопровождению программного обеспечения, а также исследованию этих подходов.

Приложение дисциплины «инженерия» к программному обеспечению.

Деятельность связанная с производством и поддержанием программного обеспечения.

1. **Понятие программного продукта (ПП).**

Комплекс взаимосвязанных программ для решения определенной проблемы массового спроса, подготовленный к реализации, как любой вид промышленной продукции. Продукт предназначен для массового использования;

1. **Понятие технологии программирования.**

Совокупность идей, технологий, регламентов и инструментов, которая обеспечивает промышленное производство программного обеспечения.

Технологии базируюся на идеях, закреплены в регламентах и представлены инструментами.

1. **Требования к промышленному производству программ: требования к производственному процессу.**

- Сроки;

- Бюджет;

- Оценка трудозатрат;

- Необходимые ресурсы;

- Необходимый персонал;

- З/п;

- Отчуждение кода от программиста;

- Изменения в коллективе («текучка» кадров, увеличение и уменьшение количества работников);

1. **Требования к промышленному производству программ: требования к продукту**.

- Полная функциональность;

- Нефункциональные требования (интерфейс, надежность, производительность и т.д.);

- Стабильность работы;

- Изменения в задаче походу разработки;

- Документация;

- Максимальное использование существующих компонент;

- Использование унаследованного программного обеспечения (Legacy);

1. **Требования к промышленному производству программ: требования к послепроизводственной стадии.**

- Механизм установки продукта;  
- Механизм защиты от нелегального копирования

- Механизм обучения пользователей;  
- Гарантийное обслуживание;

- Возможность исправления ошибок

- Возможность развития программы;

1. **Legacy software.**

Устаревшее программное обеспечение.

Программное обеспечение «доставшееся в наследство» от сторонних разработчиков;

1. **Соотношение успешных и неуспешных проектов в области разработки ПП.**

Нормальное распределение. Около 20% успешное завершение, около 30% полный провал, около 50 % завершение с превышением сроков, бюджета, неполной функциональностью.

1. **Жизненный цикл ПП.**

- Выявление проблемы. Проблема — несоответствие желаемого и действительного;

- Постановка цели;

- Нахождение идеи;

- Технико-экономическое обоснование. Идеи должны быть технически осуществимы и экономически обоснованы;

- Техническое задание;

- Проектирование;

- Реализация (разработка);

- Стабилизация (проверка, тестирование);

- Развертывание (внедрение);

- Эксплуатация(сопровождение);

1. **Основные подходы к разработке ПП.**

- Code&fix;

- Каскадный;

- Agile;

1. **Каскадный подход к разработке ПП.**

Последовательно проходящие фазы жизненного цикла продукта:

- Анализа требований;

- Проектирования;

- Реализации;

- Тестирования;

- Интеграции;

- Поддержки;

Переход к следующему этапу после полного завершения текущего. Возвраты не предусмотрены.

Такой стиль подходит для проектов, в которых проектные требования четко определяются заранее и с большой вероятностью не будут корректироваться потом. Данная схема организации разработки удобна с точки зрения управления проектом, так как позволяет четко сформулировать состав и обязанности его участников и контролировать графики выполнения проекта.

Может быть использована, если система должна заработать сразу (без права на ошибку) или если требуется жесткое планирование, например при работе по гос.заказам.

Основные недостатки: «длинная» обратная связь и высокая цена изменений и исправлений.

1. **Спиральный подход к разработке ПП.**

Сочетает итеративность и этапность;

На каждой итерации проходят этапы для фрагмента или версии программного продукта:

- Определение целей

- Оценка и разрешение рисков;

- Разработка и тестирование;

- Планирование следующей итерации;

Когда требования и параметры проекта непрерывно корректируются, а новые требования формулируются лишь по мере необходимости выполнения конкретных работ.

Исполнитель и заказчик работают в постоянном тесном сотрудничестве, клиент привлекается на каждом этапе, формулируя свои соображения по поводу созданных компонентов. Однако при такой организации очень велик риск, что процесс разработки выйдет из-под контроля, поэтому данная модель обычно используется в относительно небольших проектах.

1. **Каркасный подход к разработке ПП.**

Сборочный подход. Программный продукт «собирается» из заранее определенного набора компонентов. Применяется для задач, в которых программы имеют одинаковую структуру, но разную реализацию отдельных частей. Различные реализации имеют разные характеристики, например может быть быстрая, но не очень точная реализация или более точная, но медленная.

Есть один «каркас» программы и множество вариантов реализации каждого шага.

1. **Монументальные технологии: область применения.**

Хорошо формализованные задачи (моделирование физических процессов, математические расчеты), работа с гос.органами, создание ответственных систем без права на ошибку.

1. **Облегченные технологии: область применения.**

Проекты с сжатыми сроками и отсутствием хорошо формализованных требований (стартапы).

1. **Технология макетирования. Виды прототипов.**

Появилась на стыке Waterfall и Agile технологий. Программа разрабатывается как последовательность прототипов (макетов), каждый из которых все более полно удовлетворяет интересам заказчика. Например расширение функциональности, изменение интерфейса, ускорение работы и т.д.

Количество макетов зависит от задачи и фирмы, например 6 недель на первую версию и по 2 недели на последующие.

Опорные версии:

* 1. Демонстрационный макет – убедиться и показать заказчику, что задача разрешима в принципе.
  2. Исследовательский макет – исследуется до каких пор задача разрешима.
  3. Рабочая версия – программа с полной функциональностью, но например без базы данных.
  4. Промышленная версия – программа с полной функциональностью, базами данных и всеми настройками.
  5. Коммерческая версия – имеется локализация, система обновления, служба сопровождения, реклама

1. **Закон Конвея**.

Структура продукта, производимого организацией, определяет структуру организации;

1. **Линейная организация. Достоинства.**

1. Понятность и простота;

2. Единоначалие;

3. Масштабируемость;  
4. Устойчивость;

5. Строго определенная зона ответственности;

6. Индивидуальная ответственность

7. Соответствие ответственности и полномочий;

8. Простота распределения работы;

9. Простота оценки работы;

10. Простые правила поощрения и продвижения по службе;

11. Оперативность управления (для небольших систем);

12. Начальник отвечает за все проблемы подчиненного;

1. **Линейная организация. Недостатки.**

1. Высокие требования к квалификации руководителя;

2. Высокие требования к обработке информации руководителем;

3. Руководитель – узкое место системы, выход его из строя – выход из строя всей системы;

4. Малое количество связей, они перегружены, связи нельзя убирать;

5. Поощрение продвижением, может способствовать потере хороших специалистов

6. Проблемы связи в больших системах;

7. Отсутствие горизонтальных связей — низкая самоорганизация;

1. **Линейно-штабная организация. Достоинства и недостатки.**

Достоинства:

- Единоначалие (решения принимает руководитель);

- Наличие в штабе узких специалистов (готовят варианты решения) способствует более глубокой проработке вопросов;

- Оперативность решений;

- Эффективное использование управленческого и производственного потенциала для решения экстремальных задач;

Недостатки:

- Аналогично недостаткам линейной структуры;

Недостаточно четкое распределение ответственности (лица готовящие решение, не участвуют в его выполнении);

- Тенденции к чрезмерной централизации;

1. **Линейно-функциональная организация. Достоинства и недостатки.**

Достоинства:

- Функциональные руководители освобождают руководителя от решения специальных вопросов;

- Выше скорость передачи информации (за счет горизонтальных связей);

- Специализированная информация и решения функциональных руководителей передаются на прямую, по функциональной иерархии;

- Эффективны в стабильной среде;

Недостатки:

- Усложнение системы

- Многоначалие, противоречивые указания;

- Трудности с принятием и реализации согласованных управленческих решений;

- При появлении новых, нестандартных или сложных задач, возникает необходимость в частых согласований решений на высшем уровне управления.

1. **Бригада равных. Достоинства и недостатки.**

Достоинства:

- Сокращение управленческого аппарата;

- Гибкое использование кадров;

Недостатки:

- При росте бригады, количество связей между ее членами растет экспоненциально;

- Высокие требования к коммуникациям;

- Усложнение взаимосвязи между бригадами;

- Сложность координации;

1. **Матричная организация. Достоинства и недостатки.**

Достоинства:

- Гибкое использование кадров;

Недостатки:

- Отсутствие единоначалия;

- Проблема баланса соответствия потребностей проекта ресурсам линейных отделов;

1. **Бригада главного программиста.**

- Ролевая бригада;

- Организована по принципу хирургической бригады (один занимается основной работой, остальные оказывают поддержку);

- Рассчитана на 10 человек:

\*Главный программист;

\*Дублер

\*Инструментальщик;

\*Языковед(ЯП);

\*Архивариус(хранение, сборка версий);  
 \*Тестер;

\*Администратор (менеджер) + Секретарь;

\*Редактор (тех.писатель) + Секретарь;

- Возможно совмещение ролей одним человеком;

- Главный программист становится «Узким местом»;

1. **Kanban - доска.**

Инструмент для планирования и контроля работы команды разработчиков ПО. Должна состоять как минимум из трех колонок: «сделать» (англ. to-do), «в процессе» (in progress), «сделано»(done). При разработке ПО SCRUM канбан-доска обычно включает следующие колонки в соответствии со статусом задач: обсуждается (backlog), согласовано (ready), кодируется (coding), тестируется (testing), подтверждается (approval) и сделано (done). На доску в соответствующий столбец прикрепляются канбан-карточки (стикеры, магниты). Каждой из этих карточек соответствует задача, которая движется по доске, по мере ее выполнения. Такое движение соответствует движению SCRUM-процесса производства по Burndown Chart сверху вниз. Часто используется электронная канбан-доска.

1. **Scrum. Sprint.**

Временной промежуток от 1 до 4 недель, за который команда создает часть продукта, готовую к демонстрации и ценную для заказчика - инкремент бизнес-продукта. Оптимальная продолжительность спринта 2 недели. 1 неделя – мало, а месяц – сложно планировать. Чем короче спринт, тем более гибким является процесс разработки, релизы выходят чаще, быстрее поступают отзывы от потребителя, меньше времени тратится на работу в неправильном направлении. С другой стороны, при более длительных спринтах скрам-команда уменьшает издержки на совещания, демонстрации продукта и т. п. Разные команды подбирают длину спринта согласно специфике своей работы, кросс-функциональности команд и требований, часто методом проб и ошибок.

1. **Scrum. Цель спринта.**

Формулирует Product Owner в бизнес терминах, т. е. что в результате должен получить владелец бизнеса.

Это краткое описание бизнес-цели, ради которой выполняется данный спринт. Цель на спринт помогает команде принимать бизнес-обоснованные решения. Этот артефакт необходим для того, чтобы команда проекта могла самостоятельно принимать решение в случае появления альтернативных путей решения бизнес-задачи.

1. **Scrum. Инкремент спринта.**

Какое приращение возможностей продукта получит владелец бизнеса в бизнес-терминах. Перед началом спринта планируется ожидаемый инкремент, по окончанию спринта оценивается фактический инкремент.

1. **Scrum. User Story.**

Задача, поставленная неформально.

Требуемую бизнес-функциональность, которую добавляют в бэклог, часто называют историей. Зачастую история имеет следующую структуру: «Будучи пользователем <тип пользователя> я хочу сделать <действие>, чтобы получить <результат>». Такая структура удобна тем, что понятна как разработчикам, так и заказчикам. User Story разбиваются по важности и функциональности и далее разбиваются на задачи в бэклоге.

1. **Scrum. Story point.**

Оценка в часах, как правило, необъективна и не учитывает **эффективные часы работы**.  
Story Points — эфемерная оценка (оценка в “попугаях”), которая не имеет физических единиц измерения. 1 SP — это оценка потраченных усилий всей команды (разработчики, верстальщики, QA) на реализацию самой простой задачи (выход на live). Оценивается не длительность задачи, а ее размер относительно другой.

Оценка в Story Points будет эффективной, если будут выполнены следующие условия:

* состав команды не меняется;
* один и тот же продукт;
* задачи выполняются только в спринте;
* крупные задачи декомпозируются на мелкие;
* оценивает вся команда и все задачи;
* задачи сравниваются между собой.

Чаще всего для оценки в Story Points используют **шкалу Фибоначчи**.

Вторая по популярности — **оценка задач в трех осях** (объем, риски, сложность). Объем задачи —  время, которое уйдет на ее реализацию. Иногда бывает что задача достаточно объемная, но по сложности очень легкая и тривиальная. Бывает наоборот — задача небольшая, но сложная, требующая анализа и несущая определенные риски.

Соответственно, по этим критериям формируется табличка по задачи для каждого отдела (разработчики, дизайнеры, QA). Оценивать объем, сложность и риски можно от 1 до 3, либо от 1 до 5. Команды голосуют по этим критерии, суммируют и определяют “вес” задачи в Story Points.

Некоторые команды используют другой подход — **оценку задач в футболках**. Самая простая и маленькая задача команды обозначается размером XS. Относительно этой задачи сравниваются все последующие (S, M, L, XL, XXL). Соответственно, так команда может планировать, сколько больших, средних и крупных задач может сделать за спринт.

Аналогичная оценка — **оценка в буквах** (A, B, C, D, E), где A — самая мелкая задача, E — самая крупная.

1. **Scrum. Списки сложности задач.**

Задачи, для которых определена сложность в SP через покер планирования, помещаются в лист сложности задач, согласно определенной оценке, что упрощает процесс планирования, т.к. задача размещается в листе с задачами сходной сложности.

1. **Scrum. Журнал (backlog) продукта.**

Product Backlog - Формируется при общей встрече или индивидуальных интервью со всеми заинтересованными лицами (стэкхолдерами, пользователями). Записываются User Story, требования и пожелания.

 Project backlog — это список требований к функциональности, упорядоченный по их степени важности, подлежащих реализации. Элементы этого списка называются [пользовательскими историями](https://ru.wikipedia.org/wiki/Пользовательские_истории) (user story) или элементами беклога (backlog items). Журнал пожеланий проекта открыт для редактирования для всех участников скрам-процесса. Project backlog ведется SCRUM Product Owner.

1. **Scrum. Журнал (backlog) спринта.**

Бэклог спринта — это обещание команды, что будет добавлено в обновлении продукта по окончании спринта.

Из бэклога продукта в бэклог спринта попадают несколько требований. Их число зависит от опытности команды и сложности задач, а содержательная часть определяется целью спринта. Бэклог спринта - зона ответственности команды. После согласования и принятия обязательств, бэклог спринта обычно не изменяется. Все функции разбиты по задачам, каждая из которых оценивается скрам-командой. На Sprint Planning Meeting команда оценивает объем работы, который нужно проделать для завершения спринта методом Planning Poker.

1. **Scrum. Порядок заполнения бэклога спринта.**

Источник работы в новый спринт, «какие функции будут реализованы дальше»;

Он заполняется командой во время планирования спринта, в него берутся самые приоритетные истории из продуктового бэклога, которые соответствуют критериям Definition of Read и поставленной владельцем продукта цели;

Истории делятся на более мелкие задачи, оцениваются по трудозатратам — им присваиваются Story Point — и постепенно уходят в разработку.

1. **Scrum. Диаграмма сгорания.**

График для отслеживания прогресса и темпа работы, заполняется по мере выполнения задач.

Burndown Chart (бёрдаун чарт) – диаграмма сгорания задач. Направление графика сверху вниз. Предназначен для отслеживания оставшегося объема работ, где по оси Х кол-во дней спринта, а по оси Y кол-во Story Points. Первому дню спринта соответствует максимальное кол-во Story Points.   
  
Burnup Chart (бёрнап чарт) – диаграмма сгорания задач. Направление графика снизу вверх. Предназначен для отслеживания объема работ, где по оси Х кол-во дней спринта, а по оси Y кол-во Story Points. Последнему дню спринта соответствует максимальное кол-во Story Points.

1. **Scrum. Product owner.**

Владелец продукта, связующее звено между заказчиком и командой разработки. Представитель владельца бизнеса. Ответственный за развите продукта. Направляет работу команды в нужном направлении.

Самая главная ответственность Product Owner – это создание и контроль Product Backlog:

* определение элементов бэклога продукта;
* правильное расположение элементов для оптимизации достижения цели;
* обеспечение понятности и прозрачности Product Backlog;
* обеспечение прозрачности и понятности требований, над которыми предстоит работать всей Scrum Team;

1. **Scrum. Scrum master.**

Арбитр, который организует и проводит совещания, следит за соблюдением всех принципов скрама, разрешает противоречия и защищает команду от отвлекающих факторов, проводит фасилитацию митингов, отвечает за учет, хранение и выдачу SCRUM-инвентаря. Данная роль не предполагает ничего иного, кроме корректного ведения скрам-процесса.  
  
Scrum Master не дает заданий, а устраняет проблемы, появляющиеся внутри команды.  
Кросс-функциональная команда разработчиков проекта, состоящая из специалистов разных профилей: программистов, тестировщиков, аналитиков, архитекторов и т. д.

1. **Scrum. Команда разработчиков.**

Development Team – команда разработки, кросс-функциональная команда разработчиков проекта, состоящая из специалистов разных профилей: программистов, тестировщиков, аналитиков, архитекторов и т.д. Размер команды составляет от 5 до 9 человек (5 оптимально). Команда является единственным полностью вовлеченным участником разработки и отвечает за результат как единое целое. Данная рабочая единица является самодостаточной, самоуправляемой и самоорганизующейся. Это как некий единый организм, состоящий из отдельных элементов.

1. **Scrum. T-shape.**

Кросс-функциональная команда разработчиков проекта, состоящая из специалистов разных профилей: программистов, тестировщиков, аналитиков, архитекторов и т. д.

При этом, каждый член команды должен уметь делать все, но что-то он должен делать лучше (в отдельном направлении должен быть очень продвинут). Правило практики: нечего делать — иди тестируй.

1. **Scrum. Сессия планирования.**

Событие перед началом спринта, на котором планируется работа. Задачи нужно рассмотреть заранее, разделить на таски (более мелкие части), чтобы представлять объем и способы работы. На этом же мероприятии владелец продукта определяет цель спринта, в ней отражено, что изменится в новой версии продукта и зачем. Команда разработки наиболее активно участвует в планировании спринта. Скрам-мастер организует встречу, а владелец продукта консультирует команду как представитель бизнеса. Выбираются наиболее приоритеные задания из Бэклога, которые возможно выполнить за спринт и соотвествующие цели спринта и критериям Definition of Ready. Задачи оцениваются в SP методом покера планирования.

1. **Scrum. Definition of Ready.**

В очередной спринт берутся высокоприоритетные истории из бэклога продукта. Истории реализовываются, и задачи превращаются в инкремент продукта.

Но для того, чтобы взять элементы бэклога в текущий спринт, важно, чтобы самые приоритетные пользовательские истории были «подготовлены». Это и есть **Definition of Ready**.

Если DoR отсутствует, то есть критерии не установлены для истории, то такие истории вызовут проблемы: могут возникнуть дополнения на ходу, неучтённый объём работы, а что хуже, непродуманная задача окажется бесполезной.

Если разработчики забирают в работу недостаточно подробные или определённые User Stories, они рискуют.

Определение истории как READY основано на трёх принципах. История должна быть:

* ясной,
* выполнимой,
* проверяемой.

Пользовательская история ясна, если все члены scrum-команды понимают, что она значит. Совместное придумывание User Stories на груминге и их обсуждение повышают ясность.

История считается выполнимой, если команда может полностью реализовать её за один спринт. Если это невозможно, историю нужно разбить на более мелкие.

История проверяема, если есть способ определить, работает ли функция, как от неё ожидалось. Это гарантируют критерии приёмки (**Acceptance criteria**).

**Definition of Ready** находится на уровне бэклога и устанавливается до начала планирования спринта.

Примеры DoR:

* пользовательская история ясна,
* ее можно проверить,
* ее можно реализовать,
* определены критерии приёмки,
* учтены зависимости от других историй,
* есть критерии эффективности, масштабируемости, безопасности, если это необходимо,
* размер истории подходит scrum-команде,
* известен тот, кто принимает сделанную историю, и он знает об этом,
* разработчикам понятно, как продемонстрировать законченную историю.

1. **Scrum. Definition of Done.**

**Definition of Done определяется** на уровне спринта или релиза. DoD составляется самой командой на планировании. В списке требований зафиксировано, что нужно выполнить для определённой истории, чтобы она могла попасть в графу DONE на доске.

Definition of Done — это соглашение между командой разработки и владельцем продукта о том, что считать за минимальные критерии, когда история считается «Сделанной».

DoD варьируется от спринта к спринту. Промежуточные спринты могут иметь менее строгие критерии, тогда как DoD крупного релиза будет более серьёзным.

Примеры DoD:

* код написан,
* код прокомментирован, проверен, запущен,
* код проверил наставник или коллега (если это заведено),
* код запускается без ошибок,
* модульные тесты написаны и выполняются без ошибок,
* системные тесты выполняются,
* новая сборка задокументирована и т. д.

1. **Scrum. Daily Scrum.**

Ежедневное собрание не более 15 минут, проводимое в одно и то же время. Участвует скрам тим, наблюдать могут все. Проводит скрам-мастер. Цель митинга – оперативный обмен информацией, все в курсе происходящего, нет коммуникационных разрывов. Задаются три вопроса: что сделал вчера? что будешь делать сегодня? какие препятствия встают на пути к цели?

1. **Scrum. Обзор спринта.**

**Sprint Review Meeting** Участвуют все. Знаменуется значительным приростом функционала продукта. Демонстрация работы готового продукта или функционала.   
  
Длительность митинга: по одному часу на каждую неделю спринта (2 часа Sprint Review = 2-х недельному спринту).Подготовка к данной встрече не должна превышать 2-х часов.

Команда демонстрирует, что сделано: показывает тот самый **инкремент** продукта. Владелец соглашается и принимает его или отправляет на доработку. На мероприятии могут присутствовать и заказчики, и другие заинтересованные стороны.

1. **Scrum. Retro.**

событие после закрытия спринта, на котором команда делится опытом, обсуждает трудности и анализирует свою работу. Эта встреча проводится scrum-мастером для продуктовой команды, поэтому присутствие владельца необязательно.

Sprint Retrospective Meeting.Ретроспектива. Проводится в последний день спринта.  
Призвана оценить результат команды. Задаются вопросы: что можно улучшить? как? как повысить эффективность команды?

Время на ретроспективу для 2-х недельного спринта не более 2-х часов.  
Понятие Кайдзен и счастье. Кайдзен – непрерывное совершенствование. Счастливые люди = высокая производительность команды.   
Можно задать вопросы: Что может сделать вас счастливее в следующем спринте? Что сделает вас счастливее вообще?

1. **Эффект привязки.**

Возникает при планировании и оценке сложности задач,, когда команда открыто обсуждает оценки выражается в том, что озвученное мнение заставляет других участников корректировать свои представления (под давлением авторитета, опыта или других факторов). Для сглаживания этого эффекта применяется «Покер планирования» где все оценки даются одновременно и независимо друг от друга.

Покер планирования выявляет потенциально влиятельного участника команды, изолируя его мнение от других участников группы. Затем необходимо, чтобы участник аргументировал свой выбор, если он не совпадает с превалирующим мнением. Если участники группы могут выражать свою сплочённость таким образом, они более склонны верить в свои первоначальные оценки.

Если у влиятельного участника есть хорошие аргументы для спора, все остальные будут видеть смысл и прислушиваться, но, по крайней мере, остальные участники не будут подвержены эффекту привязки; вместо этого они должны будут исходить только из разумных соображений.

1. **Покер планирования.**

Техника оценки, основанная на достижении договорённости, используемая для оценки сложности предстоящей работы или относительного объёма решаемых задач. Минимизирует эффект привязки путём опроса каждого из участников команды таким образом, что никто не знает чужого решения до одновременного оглашения выбора каждого из участников.

Для проведения покера планирования необходимо подготовить список обсуждаемых функций и несколько колод пронумерованных карт. Карты в колодах должны быть пронумерованы. Обычно колода содержит карты, содержащие [числа Фибоначчи](https://ru.wikipedia.org/wiki/Числа_Фибоначчи), включая ноль: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89; другие разновидности колод могут использовать аналогичные последовательности.

В колоде могут быть также специальные карты:

* *знак вопроса* (?), означающий неуверенность;
* *бесконечность* (∞), означающая, что обсуждаемая функция или принципиально не может быть реализована, или слишком велика, чтобы присваивать ей число;
* *чашка кофе* (☕), означающая требование перерыва.

Аргумент в пользу использования последовательности Фибоначчи — отражение возрастающей [неопределённости](https://en.wikipedia.org/wiki/Software_development_effort_estimation" \l "Uncertainty_assessment_approaches) с ростом сложности оцениваемых функций или задач.

Задачи с высокой оценкой сложности необходимо дробить на более мелкие.

1. Scrum Master ведет собрание;
2. Product Owner представляет краткие обзоры каждой задачи;
3. Происходит обсуждение, задаются вопросы;
4. Участники Developer Team выбирают по одной карте, потом переворачивают;
5. Если в результате голосования есть большой разброс в очках, то выслушивают двоих, перевернувших карты с минимальным и максимальным значением;
6. Затем голосуют вновь и присваивают задаче Story Points.
7. **MSF. Базовые принципы.**

- Единое видение;

- Концентрация на бизнес-приоритетах;

- Проявляйте гибкость — будьте готовы к переменам;

- Поощряйте свободное общение;

- Извлекайте из всего уроки;

- Наделяйте участников полномочиями (в соответствии с ролью);

- Распределение ответственности при фиксации отчетности;

1. **MSF. Решения vs Продукты. Состав Решения.**

Продукт (product) предназначен для массового использования через распространение дистрибутивов, поставляется AS IS, а решение (solution) для решения задач конкретного заказчика, поставляется через развертывание(ответственность поставщика).

Состав решения:

1. Программно-технические средства

2. Программный код

3. Документация

4. Процесс внедрения

5. Обучение

6. Поддержка

7. Коммуникации – информирование внешних заинтересованных лиц;

1. **MSF. Треугольник компромиссов.**

- Управление компромиссами подразумевает достижение баланса всех элементов проекта;

- Возможны ситуации, когда за выделенное время невозможно решить все поставленные задачи;

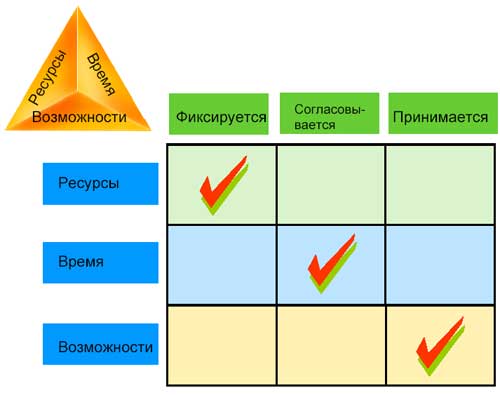
- Необходим постоянный анализ и нахождение компромиссных решений о размере бюджета, сроках и полнотой функциональности продукта;

- Треугольник компромиссов наглядно показывает, что любое изменение одного из компонентов требует корректировки других;



1. **MSF. Матрица компромиссов.**

- В матрице компромиссов компоненты (ресурсы, сроки, функционал) делятся по приоритету на обязательные (параметр фиксируется), желательные (параметр согласовывается) и факультативные (параметр принимается как есть).



1. **MSF. Организация бригады.**

- Основана на концепции бригады равных (небольшие многопрофильные команды, роли равноравны);

- При этом участники разделены по ролевым группам, каждая из которых имеет зону ответственности и защищает интересы определенных лиц, роль имеет решающий голос в зоне своей ответственности;

- Баланс ответственности и полномочий;

- Каждая роль отчитывается за результат деятельности по достижению своих качественных целей;

- Планирование «снизу»;

- Модель бригады может масштабироваться в зависимости от числа участников;

1. **MSF. Концепция Advocacy group.**

Участники разделены по ролевым группам, каждая из которых имеет зону ответственности и защищает интересы определенных лиц, роль имеет решающий голос в зоне своей ответственности;

1. **MSF. Роль Product manager.**

- Представитель заказчика в бригаде;

- Работа с требованиями заказчика (выявление stakeholders и их проблем);

- Формирование ожиданий заказчика;

- Формирование общего видения и рамок проекта;

- Определение компромиссов (время, ресурсы, функциональность);

- Маркетинг

- План коммуникаций (Связь с клиентом);

Обязанности по этапам:

- Envisioning (Выявление нужды требований заказчика, общее видение и рамки проекта)

- Planning (Концептуальный дизайн, анализ бизнес-требований, коммуникационный план)

- Developing (Формирование ожиданий заказчика)

- Stabilizing(Выполнение коммуникационного плана, планирование премьеры продукта)

- Deploying(Получение отзывов и оценок заказчика, акт о приеме работы).

1. **MSF. Роль Program manager.**

- Управляет разработкой ПО и несет ответственность за его поставку;

- Время, деньги, возможности;

- Планирование и распоряжение бюджетом;

- Планирование итераций;

- Управление рисками

Обязанности по этапам:

- Envisioning (Концепция решения, структура проекта, риски)

- Planning (Концептуальный и логический дизайн, функциональная спецификация, сводный план и сводный календарный график, бюджет)

- Developing (Управление функциональной сппецификацией, мониторинг проекта, доработка планов)

- Stabilizing(Мониторинг проекта, приоретизация ошибок)

- Deploying(Сопоставление рамок проекта с поставленным решением).

1. **MSF. Роль Architecture.**

- Определение архитектуры системы и стэка используемых технологий;

1. **MSF. Роль Developer.**

- Разработка (проектирование и реализация) ПО в соответствии с заданными спецификациями;

- Консультирование бригады по технологическим вопросам

Обязанности по этапам:

- Envisioning (Прототипирование, анализ технологических возможностей, анализ осуществимости)

- Planning (Оценка технологий, логический и физический дизайн, план и календарный график разработки, смета разработки)

- Developing (Разработка программного кода и инфраструктуры, документирование конфигураций)

- Stabilizing(Устранение ошибок, оптимизация программного кода)

- Deploying(Разрешение проблем, поддержка эксплуатации).

1. **MSF. Роль Tester.**

- Выявление неполадок;

- Дает окончательное разрешение на выпуск;

- Проверка соответствия реализованного функционала продукта общей концепции;

Обязанности по этапам:

- Envisioning (Стратегии тестирования, критерии приемлимости, их влияние на разработку)

- Planning (Оценка дизайна, план и календарный график тестирования)

- Developing (Функциональное тестирование, тестирование документации, доработка плана тестирования)

- Stabilizing(Тестирование, сообщение об ошибках и их статусе)

- Deploying(Тестирование производительности).

1. **MSF. Роль Release manager.**

- Развертывание и последующая эксплуатация продукта;

- Организация обучения персонала заказчика;

- Проверка соответствия инфраструктуры заказчика требованиям ПО;

Обязанности по этапам:

- Envisioning (Требования внедрения, их влияние на разработку, требования сопровождения)

- Planning (Оценка дизайна, план и календарный график пилотного и окончательного внедрения)

- Developing (Чеклисты развертывания, чеклисты подготовки к внедрению, доработка планов внедрения)

- Stabilizing(Развертывание и поддержка пилотного внедрения, планирование внедрения, обучение персонала сопровождения)

- Deploying(Управление внедрением, одобрение изменений).

1. **MSF. Роль User experience.**

- Представляет интересы конечных пользователей;

- Занимается изучением и решением проблем пользователей;

- Оценивает продукт на предмет соответствия потребностей пользователей;

Обязанности по этапам:

- Envisioning (Необходимые эксплуатационные характеристики и их влияние на разработку)

- Planning (Сценарии/примеры использования, пользовательские требования, требования локализации и общедоступности, пользовательская документация, план обучения, график тестирования удобства эксплуатации)

- Developing (Доработка плана обучения, тестирование удобства эксплуатации, графический дизайн)

- Stabilizing(Доработка эксплуатационных руководств, учебные материалы)

- Deploying(Обучение управление календарным графиком обучения).

1. **MSF. Изменения в бригаде в зависимости от масштаба проекта.**

- В зависимости от масштаба проекта, некоторые роли могут выполняться одним участником бригады (для небольших проектов) или несколько человек могут представлять одну роль (для крупных проектов), т. е. объединяться в подкоманды, каждая со своим лидером;

- для особо сложных проектов Program manager может быть представлен несколькими функциональными группами.

- При этом некоторые роли хорошо совмещаются с другими (User experience), другие не совмещаются (Developer);

- Необходимо избегать сочетания ролей, имеющих конфликт интересов;

1. **MSF. Матрица совмещения ролей.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Архитектура продукта | Управление продуктом | Управление программой | Разработка | Тестирование | Удовлетворение потребителя | Управление выпуском |
| Архитектура продукта |  | Нет | Да | Да | Не желательно | Не желательно | Не желательно |
| Управление продуктом | Нет |  | Нет | Нет | Да | Да | Не желательно |
| Управление программой | Да | Нет |  | Нет | Не желательно | Не желательно | Да |
| Разработка | Да | Нет | Нет |  | Нет | Нет | Нет |
| Тестирование | Не желательно | Да | Не желательно | Нет |  | Да | Да |
| Удовлетворение потребителя | Не желательно | Да | Не желательно | Нет | Да |  | Не желательно |
| Управление выпуском | Не желательно | Не желательно | Да | Нет | Да | Не желательно |  |

1. **MSF. Организация процесса разработки.**

* Итеративный подход (рекомендовано первая версия-базовая функциональность, затем ее наращивание);
* Подход, основанный на фазах и вехах;
* Вехи — опорные точки для планирования и мониторинга хода проекта;
* Процесс разработки разбит на стадии:

- Envisioning (формирование бригады, общее видение и рамки проекта, риски)

- Planning (Составление планов проекта)

- Developing (Создание компонентов решения)

- Stabilizing(Тестирование)

- Deploying(Внедрение).

1. **MSF. Стадия Envisioning.**

Выработка концепции.

Задачи:

- Создание и сплочение проектной группы;

- Формирование общего видения(цели, представления и ожидания заказчика каким должно быть решение) и рамок(что из предложенного будет реализовано в соответствии с ограничениями) проекта;

- Выявление главных рисков;

- Выявление и анализ бизнес-требований;

Веха «Концепция утверждена»

Результаты:

-Общее видение и рамки проекта;

- Документ оценки рисков;

- Описание структуры проекта;

Ведущая роль — Product manager;

1. **MSF. Стадия Planning.**

Составление планов проекта;

Анализ и документирование требований проекта:

- Бизнес требования;

- Потребительские требования;

- Эксплуатационные требования;

- Системные требования;

Проектирование решения (концептуальный дизайн, логический, физический) и создание функциональной спецификации(детально описывают вид поведение составляющих решения , их архитектуру и дизайн);

Цели функциональной спецификации:

- Инструкции для разработчиков;

- Основа для оценка объема работы;

- Согласование с заказчиком;

- Синхронизация работы проектной команды;

Веха «Планы проекта утверждены»

Результаты — базовая версия проекта, включает:

- Функциональная спецификация;

- План управления рисками;

- Сводный план и сводный календарный график проекта;

Ведущая роль — Program manager;

1. **MSF. Стадия Developing.**

Создание компонентов решения (код и документация).

Веха «Разработка завершена» Создание всех составляющих завершено, решение готово к тестированию.

Результаты:

- Исходный и исполнимый код;

- Стратегия развертывания;

- Окончательная функциональная спецификация;

- Материалы поддержки решения;

- Спецификации и сценарии тестов;

Ведущая роль — Development, User experience;

1. **MSF. Стадия Stabilizing.**

Тестирование решения, приоретизация и устранение ошибок. Подготовка решения к выпуску.

При появлении достаточно стабильной версии производится пилотное внедрение (внедрение в часть производственной среды или для части пользователей).

Веха «Готовность решения утверждена»

Результаты:

- Окончательный продукт;

- Документация выпуска;

- Материалы поддержки решения;

- Результаты тестирования;

- Исходный и исполнимый код;

- Проектная документация;

Ведущая роль — Tester, Realise manager;

1. Улучшение качества решения для выполнения условий к развертыванию в производственной среде.

2. Проверка соответствия требованиям и ожиданиям заинтересованных лиц.

3. Проверка удобства решения с точки зрения пользователя.

4. Оптимизация и снижение рисков, связанных с развертыванием и использованием решения в целевых средах.

1. **MSF. Стадия Deploying.**

1. Успешная интеграция решения в производственных средах.

2. Перенос ответственности за оставшуюся часть реализации решения с проектной группы на операционные группы и группы поддержки.

Веха «Внедрение завершено».

Результаты:

- Информационные системы эксплуатации и поддержки;

- Процедуры и процессы;

- Базы знаний, отчеты;

- Версии проектных документов, код созданные в ходе проекта;

- Отчет о завершении проекта;

- Окончательные версии проектных документов;

- Показатели удовлетворенности заказчика и потребителей;

Ведущая роль — Realise manager;

1. **MSF. «Вскрытие».**

Документ обобщающий взгляды участников бригады на весь процесс создания решения:

- Достижения;

- Проблемы и ошибки;

- Уроки;

- Выводы;

Post Mortal, по окончании проекта делают постпроектный анализ для того, чтобы узнать, что не получилось, на каком этапе допущена ошибка, почему она допущена, что нужно сделать для того, чтобы ее избежать.

1. **MSF. Сборки ПП.**

Проведение ежедневных сборок всех модулей в целостную программу с целью тестирования и сохранения удачной версии.

1. **MSF. Релизы. Уровни версий.**

Релизы (версионность - регулярность). Положительные аспекты:

1. Обратная связь для разработчика (ошибки)

2. Обратная связь для заказчика (продвижение)

3. Конкретные цели, небольшие задачи

4. Последовательное расширение версий

5. Версии планируются

3 уровня версий:

1. разработчика (личная)  общая сборка

2. тестера

3. презентации

1. **MSF. Базовые версии.**

Зафиксированное состояние любого артефакта или проекта в целом.

Помогают управлять изменениями.

Фиксация состояний всего проекта – точки отсчета

Три базовых версии – собственная, ежедневный билд, версия для демонстрации заказчику.

В рамках MSF программный код, документация, дизайн, планы и другие рабочие материалы создаются, как правило, итеративными методами. MSF рекомендует начинать разработку решения с построения, тестирования и внедрения его базовой функциональности.

Затем к решению добавляются все новые и новые возможности. Такая стратегия именуется стратегией версионирования. Несмотря на то, что для малых проектов может быть достаточным выпуск одной версии, рекомендуется не упускать возможности создания для одного решения ряда версий. С созданием новых версий эволюционирует функциональность решения.

1. **MSF. «Живые документы».**

Документация изменяется по мере эволюции проекта;

Разработка документации идет параллельно с программированием. Требуется постоянный пересмотр. Документация должен соответствовать текущему состоянию разрабатывемой системы.

1. **MSF. Особенности планирования.**

Планирование с учетом рисков

Планирование снизу вверх

Планирование с резервным временем (резервы в распоряжении PgM)

1. MSF: Стремление к совершенству.

Постэтапный анализ — собрание бригады, анализ и обсуждение достижений и просчетов, как закрепить достижения и недопускать ошибок (чтобы ошибки не повторялись).

1. **Zero-defect mindset.**

Одна из глобальных идей управления качеством. Автор Ф.Кросби. Ошибки возможны, но необходимо примимать такие меры что бы ошибку невозможно было повторить.

*Программа нулевых дефектов {программа ZD)* базируется на следующих положениях:

• упор на предупреждение появления дефектов, а не на их обнаружение и исправление;

• направленность усилий на сокращение уровня дефектности в производстве;

• осознание факта, что потребитель нуждается именно в бездефектной продукции и производитель может и должен именно такую продукцию поставить2;

• необходимость для руководства предприятия четко сформулировать цели в области качества на длительный период3;

• понимание того, что качество работы компании определяется не только качеством производственных процессов, но и качеством деятельности непроизводственных подразделений (деятельность таких подразделений рассматривается как оказание внутренних услуг[[2]](https://studref.com/311659/menedzhment/krosbi_kontseptsiya_nulevyh_defektov" \l "gads_btm));

• признание необходимости финансового анализа деятельности в области обеспечения качества.

Основой успеха программы *ZD* стал принцип недопустимости изначального установления какого-либо приемлемого уровня дефектности, кроме нулевого2. Фундамент процветающей организации, по мнению Кросби, закладывается ясным пониманием четырех принципов управления качеством:

• качество означает соответствие требованиям;

• качество достигается предупреждением дефектов, а не их оценкой;

• стандарт качества — это отсутствие дефектов, или нулевые дефекты;

• измерителем качества является *цена несоответствия* (потери от несоответствия требованиям, связанные с тем, что та или иная работа не сделана правильно с первого раза), а не какие-либо искусственные надуманные показатели. Именно это понятие он призывает сопоставлять с затратами на обеспечение качества вместо расчетной дополнительной прибыли за счет повышения качества.

**54-87.. MSF. Действия каждой роли на каждой стадии проекта. (Cм. файл MSF.doc).**

**Фаза выработки концепции**

Основная роль Program manager

|  |  |  |  |
| --- | --- | --- | --- |
| 54 | Product manager | Управление продуктом | Общие цели проекта; выявление нужд и требований заказчика; документ общего описания и рамок проекта. |
| 55 | Program manager | Управление программой | Цели дизайна; концепция решения; структура проекта. |
| 56 | Developer | Разработка | Прототипирование; анализ технологических возможностей; анализ осуществимости. |
| 57 | User experience | Удовлетворение потребителя | Необходимые эксплуатационные характеристики решения и их влияние на его разработку. |
| 58 | Tester | Тестирование | Стратегии тестирования; критерии приемлемости, их влияние на разработку решения. |
| 59 | Release manager | Управление выпуском | Требования внедрения и их влияние на разработку решения; требования сопровождения. |

**Фаза планирования**

|  |  |  |  |
| --- | --- | --- | --- |
| 60 | Product manager | Управление продуктом | Концептуальный дизайн; анализ бизнес-требований; коммуникационный план. |
| 61 | Program manager | Управление программой | Концептуальный и логический дизайн; функциональная спецификация; сводный план и сводный календарный график проекта; бюджет. |
| 62 | **Developer** | Разработка | Оценка технологий; логический и физический дизайн; план и календарный график разработки; смета разработки (development estimates). |
| 63 | User experience | Удовлетворение потребителя | Сценарии/примеры использования, пользовательские требования, требования локализации и общедоступности (accessibility); пользовательская документация/план обучения/график тестирования удобства эксплуатации; обучение. |
| 64 | Tester | Тестирование | Оценка дизайна; требования тестирования; план и календарный график тестирования. |
| 65 | Release manager | Управление выпуском | Оценка дизайна; эксплуатационные требования; план и календарный график пилотного и окончательного внедрения. |

**Фаза разработки**

|  |  |  |  |
| --- | --- | --- | --- |
| 66 | Product manager | Управление продуктом | Ожидания заказчика. |
| 67 | Program manager | Управление программой | Управление функциональной спецификацией; мониторинг проекта; доработка планов. |
| 68 | **Developer** | Разработка | Разработка программного кода и инфраструктуры; документирование конфигураций. |
| 69 | User experience | Удовлетворение потребителя | Обучение; доработка плана обучения; тестирование удобства эксплуатации (usability testing); графический дизайн. |
| 70 | Tester | Тестирование | Функциональное тестирование; выявление проблем; тестирование документации; доработка плана тестирования. |
| 71 | Release manager | Управление выпуском | Чеклисты развертывания (rollout checklists); доработка планов внедрения (включая пилотное внедрение); чеклисты подготовки к внедрению (site preparation checklists). |

**Фаза стабилизации**

|  |  |  |  |
| --- | --- | --- | --- |
| 72 | Product manager | Управление продуктом | Исполнение коммуникационного плана; планирование премьеры продукта. |
| 73 | Program manager | Управление программой | Мониторинг проекта; приоритезация ошибок. |
| 74 | **Developer** | Разработка | Устранение ошибок; оптимизация программного кода. |
| 75 | User experience | Удовлетворение потребителя | Доработка эксплуатационных руководств; учебные материалы. |
| 76 | Tester | Тестирование | Тестирование; сообщение об ошибках и их статусе; тестирование конфигурации. |
| 77 | Release manager | Управление выпуском | Развертывание и поддержка пилотного внедрения; планирование внедрения; обучение персонала сопровождения. |

**Фаза внедрения**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Роль |  |  |
| 78 | Product manager | Управление продуктом | Получение отзывов и оценок заказчика; акт о приеме выполненной работы. |
| 79 | Program manager | Управление программой | Сопоставление рамок проекта с поставленным решением; управление стабилизацией. |
| 80 | **Developer** | Разработка | Разрешение проблем; поддержка эскалации. |
| 81 | User experience | Удовлетворение потребителя | Обучение; управление календарным графиком обучения. |
| 82 | Tester | Тестирование | Тестирование производительности. |
| 83 | Release manager | Управление выпуском | Управление внедрением; одобрение изменений. |