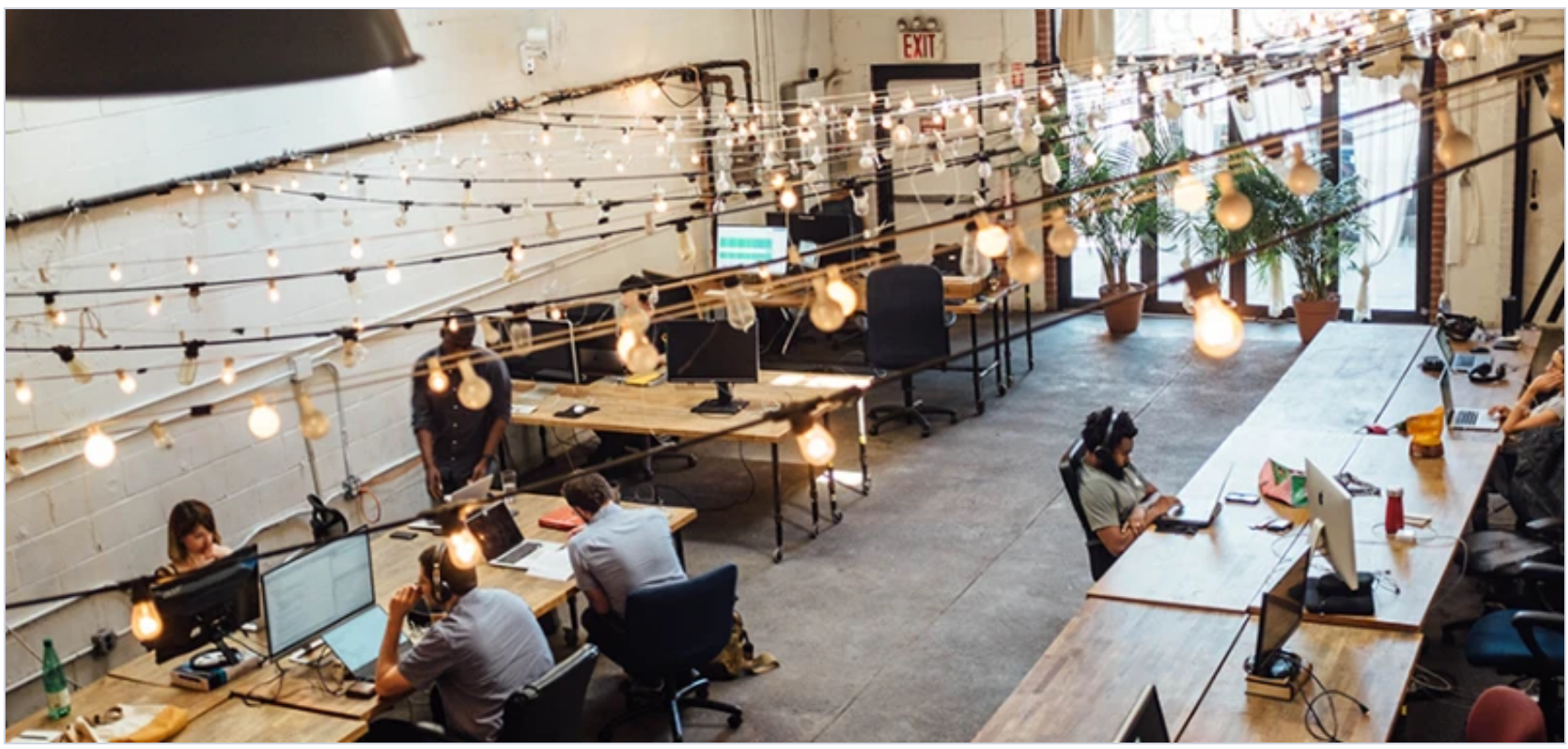
lichMax

40 уровень

Санкт-Петербург

20.07.2017 9440 4

Уровень 38. Ответы на вопросы к собеседованию по теме уровня



Здравствуйте. Опять-таки, похоже нет ответов на вопросы из этого уровня. Поэтому, как обычно, выкладываю свои. Вдруг, кому помогут (или кто-то что-то дополнит или ответит лучше).

Итак, были такие вопросы:

1. Что такое Agile?
2. Что такое Scrum?
3. Какие роли Scrum вы знаете?
4. Что такое спринт? Расскажите с подробностями
5. Кто такие QA?
6. Кто такой product owner?
7. Расскажите об иерархии исключений
8. Что делать если JVM выкинула Error?
9. Какие нововведения в области исключений из Java 7 вы знаете?
10. Зачем нужны аннотации? Как ими пользоваться?

А теперь мои ответы:

1. **Agile** — это серия подходов к разработке программного обеспечения, ориентированных на использование итеративной

самооранизующихся рабочих групп, состоящих из специалистов разных профилей. Существует несколько методик, относящих к гибкой методологии разработки, в частности экстремальное программирование, DSDM, Scrum, FDD.

Основные концепции agile-методов:

- люди и взаимодействие важнее процессов и инструментов;
- работающий продукт важнее исчерпывающей документации;
- сотрудничество с заказчиком важнее согласование условий контракта;
- готовность к изменениям важнее следованию первоначального плана.

Также были сформулированы основные принципы такого подхода:

- удовлетвореие клиента за счёт ранней и бесперебойной поставки ценного программного обеспечения;
- приветствие изменений требований даже в конце разработки (это может повысить конкурентноспособность продукта на рынке);
- частая поставка рабочего программного обеспечения (каждый месяц или каждую неделю, или даже ещё чаще);
- тесное, ежедневно общение заказчика с разработчика на протяжении всего проекта;
- проектом занимаются мотивированные личности, обеспеченые нужные условиями работы, поддержкой и доверием;
- рекомендуемый метода передачи информации - личный разговор (лицом к лицу);
- работающее программное обеспечение - лучший измеритель прогресса;
- спонсоры, разработчики и пользователи должны поддерживать постоянный темп на неопределённый срок;
- постоянно внимание улучшению технического мастерства и удобному дизайну;
- простота - искусство не делать лишней работы;
- лучшие технические требования, архитектура и дизайн полуются у самороорганизованной команды;
- постоянная адаптация к изменяющимся условиям.

Основной проблемой разработки было признано то, что никто из участников ни на одном этапе не обладает всей полнотой информации о том, что делать.

2. **Scrum** — это одна из методик гибкой разработки. Она делает акцент на качественном контроле процесса разработки. Основная особенность скрама — это разбиение процесса разработки на итерации имеющий чёткую протажённость по времени (обычно 2-6 недель; их называют "спринтами").

   В начале спринта проводится "планирование спринта" — совещание на 3-4 часа, где обсуждаются основные задачи, которые будут решаться на протяжении спринта. В конце спринта проводится "демо" – демонтрация результатов работы команды за этот спринт.

   Перед самым первым спринтом заказчик (или его представитель) формирует список требований, предъявляемых к будущему продукту. Такие требования называются "user story", а самого заказчика называют "product owner". Также в этом списке заказчик (PO) указывает приоритет для каждой задачи. Сначала будут реализовываться задачи с более высоким приоритетом. Весь список называется "product backlog", или "резерв продукта".

   Кроме product owner, среди участников ещё выделяют scrum-мастера: он проводит все совещания, следит за соблюдением всех принципов скрама, разрешает противоречия и защищает команду от отвлекающих факторов. Обычно в качестве скрам-мастера выступает кто-то из команды.

   На первом совещании в начале спринта каждой задаче, кроме того, что назначается приоритет, даётся ещё приблизительная оценка в "абстрактных человеко-днях", которые ещё называют "story point". Затем команда решает, сколько она успеет выполнить задач за время спринта. Например, заказчик отобрал 7 задач, а команда сможет сделать только 5. Тогда остальные две задачи пойдут на следующий спринт. Если заказчику это не нравится, он может повысить их приоритет, но тогда другие задачи выпадут из спринта. Кроме того, скрам-мастер может разбить некоторые задачи на более мелкие и расставить им различные приоритеты, чтобы заказчик остался доволен.

   Список отобранных задач на спринт называются "sprint backlog", или "резерв спринта". Для лучшей наглядности обычно составляется таблица, в которой указываются задачи, которые нужно реализовать, которые находятся в процессе реализации, и те которые уже выполнены. Также строится диаграмма "сгорания задач". Она графически показывает, сколько ещё задач осталось выполнить. В идеале график сгорания задач к концу спринта должен опустится до нуля.

   Также в течение спринта каждый день или через день проводятся небольшие совещания внутри команды на 5-15 минут,

Когда спринт завершается, scrum-master проводит demo, на котором демонстрируется список всего, что полностью сделано. Затем проводится «разбор полетов» — совещание тоже на пару часов. На нем обычно пытаются выяснить, что было сделано хорошо, а что (и как) можно было сделать лучше.

Обычно за 2-3 спринта можно выявить основные проблемы, которые мешают команде работать эффективнее, и устранить их. Это приводит к большей продуктивности, не увеличивая нагрузку на команду. Такое было невозможным до эры гибких методологий.

Также в середине спринта, иногда еще проводят «вычесывание» — совещание посвященное планированию следующего спринта. На нем обычно уточняют приоритеты задач, а так же могут разбить некоторые задачи на части и/или добавить новые задачи в product backlog – резерв продукта.

3. В скрам участников делят на "свиней" и "кур". "Свиньи" полностью задействованы в процессе, "куры" — только частично. К категории "свиней" относят следующие роли:

   - Скрам-мастер (проводит совещания, следит за выполнением принципов скрама и пр.; это кто-то из команды, produst owner не может быть скрам-мастером);
   - Владетелей продукта (Product Owner) (представляет интересы конечных пользователей и других заинтересованных в продукте сторон);
   - Команда разработки (Development Team) (кросс-функциональная команда, состоящая из специалистов разных профилей: разработчиков, тестировщиков, дизайнеров, архитекторов и т.д. Размер команды в идеале составляет от 3 до 9 человек. Команда является единственные полностью вовлечённым участником разработки и отвечает за результат как единое целое. Никто, кроме команды, не может вмешиваться в процесс разработки на протяжении спринта).

   К "курам" относятся следующие роли:

   - Пользователи (Users)
   - Клиенты, продавцы (Stakeholders) (лица, которые инициируют проект и для кого проект будет приносить выгоду. Они вовлечены в скрам только во время обзорного совещания по спринту (Sprint Review));
   - Управляющие (Managers) (люди, которые управляют персоналом);
   - Эксперты-консультанты (Consulting Experts).

4. **Спринт** — это одна итерация цикла разработки программного обеспечения в Скраме. Обычно спринт жёстко фиксирован по времени. Продолжительность спринта выбирается на основании размера команды, специфики работы, требований, размера проекта. Чаще всего это подбирается опытным путём, на основании проб и ошибок. В среднем спринт может быть продолжительность от 2 недель до 4 (хотя в некоторых компания его продолжительность бывает равна и 6 неделям). Для оценка объёма работ в спринте может быть использована предварительная оценка, измеряемая в очках истории. Предварительная оценка фиксируется в беклоге проекта.

5. **QA** расшифровывается как Quality Assuarance (то есть "Гарантия качества", если перевести дословно). К этой категории относятся тестировщики. Это люди, которые выявляют в программе баги и ошибки. Для этого они используют разные средства, как ручные, так и автоматические. После выявления багов и ошибок они сообщают об этом разработчикам, и те исправляют эти ошибки.

6. **Product Owner** (владелец проекта) — это заказчик или представитель заказчика. Это либо кто-то со стороны клиента (сторонняя фирма или физ.лицо), либо какой-то другой сотрудник фирмы, определяющий требования к конечному продукту и следящий за их выполнением (например, бизнес-аналитик). Как уже было сказано, он определяет требования к продукту и задачи, которые должна решить команда разработки, чтобы получить конечный продукт. Список требований (или задач) он оформляет в определённым порядке и с заданным приоритетом для каждого требования (задачи). Обычно он участвует в планировании спринта (перед началом спринта) и в демонстрации результатов спринта (после окончания спринта).

7. Во главе иерархии исключений в Java стоит класс `Throwable`. От не наследуются два класса: `Error` и `Exception`. Объекты первого класс выбрасываются, когда возникают какие-то серьёзные ошибки в работе программы и java-машины в целом (а также в работе ОС и аппаратной части компьютера), так, что программа больше не может продолжать работать и закрывается. Это может быть недостаток памяти (OutOfMemoryError) или переполнение стека (StackOverflowError).

   Экземпляры второго класса могут выбраться при самых различных исключительных ситуаций. Собственно от этого класса наследуется класс `RuntimeException` и все остальные классы. Первый класс так выделяют, потому что, во-первых

исключительные ситуации обычно являются следствием непредвиденного стечения обстоятельств, например, ошибки ввода-вывода.

Также исключения делятся на проверяемые (checked) и непроверяемые (unchecked). Первые необходимо либо отлавливать в блоке `try-catch`, либо указывать в качестве выбрасываемых в сигнатуре метода (используя слово *throws*). Непроверяемые этого не требуют, и выбор остаётся за программистом. К проверяемым относятся исключения типа `Throwable` (а также наследуемые от него) и типа `Exception` (и наследуемые от него, кроме исключений типа `RuntimeException`). К непроверяемым относятся исключения типа `Error` (а также наследуемые от него) и исключения типа Runtime (и наследуемые от него).

8. Ничего, программа просто закроется. Обычно это критические ошибки, не всегда возникшие по вине программиста (например, выход из строя жёсткого диска или сбой операционной системе). Правда, среди них есть две ошибки, которые могу следствием несовершенства кода — это `StackOverflowError` и `OutOfMemoryError`. Первый тип ошибок возник, при переполнении стека вызовов — то есть, когда вызывается слишком много методов и стек вызовов становится слишком больший (типичный пример — бесконечная рекурсия или рекурсия с большим количеством итераций). Вторая ошибка — `OutOfMemoryError` — возникает, когда переполняет память отведённая под объекты, то есть когда создаётся слишком много тяжеловесных объектов, и они все держатся в памяти (или когда сборщик мусора не успевает их уничтожать).

9. В Java 7 в плане исключение были введены три новых "вещи":

   - `Multicatching`. Это когда в одном `catch`-блоке сразу обрабатывается несколько исключений:

   ```
   try {
    } catch (Exception1|Exception2|Exception3| ... | ExceptionN ex) {}
   ```

   В таком случае параметр `ex` будет неявно иметь модификатор `final`. Кроме того, байт-код, сгенерированный компиляцией такого кода (с единым `catch`-блоком) будет короче, чем байт-код, сгенерированный компиляцией кода с несколькими `catch`-блоками.

   - `Try-with-resources`. Эта конструкция позволяет после слова `try` в скобках указать открываемые ресурсы, которые сами закроются после окончания конструкции `try () {} catch{}`. В качестве ресурса в данной конструкции может быть использован любой объект, реализующий интерфейс `AutoCloseable`.

   Во время закрытия ресурсов тоже может быть брошено исключение. В `try-with-resources` добавлена возможность хранения "подавленных" исключений, и брошенное try-блоком исключение имеет больший приоритет, чем исключения, получившиеся во время закрытия. Получить последние можно вызовом метода `getSuppressed()` от исключения, брошенного `try`-блоком.

   - `Rethrowing`. Перебрасывание исключений с улучшенной проверкой соответствия типов.

   Компилятор Java SE 7 тщательнее анализирует перебрасываемые исключения. Рассмотрим следующий пример:

```
static class FirstException extends Exception { }
static class SecondException extends Exception { }

public void rethrowException(String exceptionName) throws Exception {
    try {
        if ("First".equals(exceptionName)) {
            throw new FirstException();
        } else {
            throw new SecondException();
        }
    } catch (Exception ex) {
        throw e;
    }
}
```

В примере `try`-блок может бросить либо `FirstException`, либо `SecondException`. В версиях до Java SE 7

В Java SE 7 вы можете указать, что метод `rethrowException` бросает только `FirstException` и `SecondException`. Компилятор определит, что исключение `Exception` ex могло возникнуть только в `try`-блоке, в котором может быть брошено `FirstException` или `SecondException`. Даже если тип параметра `catch` — `Exception`, компилятор определит, что это экземпляр либо `FirstException`, либо `SecondException`:

```
1  public void rethrowException(String exceptionName) throws FirstException, SecondException {
2       try {
3           // ...
4       } catch (Exception e) {
5           throw e;
6       }
7   }
```

Если `FirstException` и `SecondException` не являются наследниками `Exception`, то необходимо указать и `Exception` в объявлении метода.

- Аннотации используются для размещения рядом с классом, полем, методом или переменой какой-то дополнительной, служебной информации (метаданных), относящейся к ней. Чтобы указать аннотацию, надо над заголовком класса или метода, либо надо объявлением поля или переменной, написать после @ название аннотации (класса аннотации), например так:

```
1  ...
2  @Override
3  public void doSomeThing() {
4  }
5  ...
```

Кроме того, у аннотации могут быть свойства, они указывают в скобках через запятую в виде пар "ключ=значение", где в качестве ключа выступает название свойства, а качестве значения — собственно, значение, которое должно принять это свойство. Выглядит это так:

```
1  @CatInfo(manager=Catmanager.class, unique=true)
2  class Cat {}
```

Если указывается указывает значение только для одного свойства, то имя свойства и знак равно можно не писать: `@SuppressWarnings("unchecked")`.

Также можно вообще не ставить скобки, если никаким свойствам не присваиваются никакие значения.

Чтобы создать свою аннотацию, надо указать модификатор доступа, потом после пробела ключевое слово "*@interface*" и дальше опять после пробела имя аннотации (принять начинать его с большой буква). Дальше идёт тело аннотации в фигурных скобках. Внутри тела указываются свойства в виде объявлений методов (как в интерфейсах). Также у свойств можно указать значения по умолчанию (они их будут принимать, если при указании аннотации в нужном месте, этому свойству не будет присвоено какое-то значение). Всё вместе это выглядит так:

```
1  @interface CatManager
2  {
3   Class manager();
4   boolean unique();
5   String name() default "Unknown Cat";
6  }
```

Аннотации выполняют следующие функции:

a. дают необходимую информацию для компилятора;
b. дают информацию различным инструментам для генерации другого кода, конфигураций и т. д.;
c. могут использоваться во время работы кода;
d. повышают читабельность кода и его понимание программистами.

их поводу показывает предупреждение). Вторая аннтоциая ставится над переопределёнными методами класса-наследника (это позволяет контролировать связь между исходным методом и переопределённым). Третья аннотация позволяет подавлять (не выводить) некоторые исключительные предупреждения, обычно выводимые компилятором в связи с данным методом или классом (например, что он уже устарел и нерекомендован к использованию).

lichMax

+25

Комментарии (4) популярные новые старые

JavaCoder

Введите текст комментария

Сергей Уровень 31, Киев, Украина 25 ноября 2019, 09:56

Согласен с MrArtemAA по поводу product owner. Мало того, product owner могут быть два. Как со стороны заказчика, так и со стороны разработки

Ответить 0

hidden #1281202 Уровень 41 14 августа 2019, 15:58

"тесное, ежедневно общение заказчика с разработчика на протяжении всего проекта;" Product owner не общается с разработчиками, а с бизнес-аналитиками, которые в свою очередь разговаривают с менеджерами или напрямую с разработчиками.

Ответить 0

Roman Уровень 32, Германия 12 марта 2020, 15:32

То, что вы описали, это Скрамфол (попытка выдать вотерфол за скрам) :)
Если PO не будет общаться напрямую с разработчиками, то как она сможет доносить до них видение того, каким должен быть конечный результат?
Если PO для работы нужны данные от аналитиков - это ее право, она может с ними общаться, но с программистами - только напрямую. Иначе они будут реализовывать то, что им расскажут аналитики, а не то, что замыслила PO.

Ответить +2

MrArtemAA Уровень 40, Москва, Россия 14 декабря 2017, 15:37

По вопросам 2, 6 — product owner — это не заказчик. Это именно владелец продукта. Он знает все о продукте, имеет его видение и постоянно общается в заказчиком (Stakeholder) и командой. Хорошее видео по роли PO

Ответить +3

RUSH

JavaRush — это интерактивный онлайн-курс по изучению Java-программирования с нуля. Он содержит 1200 практических задач с проверкой решения в один клик, необходимый минимум теории по основам Java и мотивирующие фишки, которые помогут пройти курс до конца: игры, опросы, интересные проекты и статьи об эффективном обучении и карьере Java-девелопера.

ПОДПИСЫВАЙТЕСЬ

ЯЗЫК ИНТЕРФЕЙСА

Русский