МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЕНИЯ В ТЕХНИЧЕСКИХ СИСТЕМАХ

Кафедра «Информационных системы»

Реферат
по дисциплине ТСПП
на тему:
«*Crystal*»

Выполнил:
ст. гр. ИС/б-21-о
Куркчи А. Э.
Проверил:
Строганов В. А.

Севастополь
2016

Содержание

ВВЕДЕНИЕ

Целью работы является детально изучить выданную по варианту методологию, узнать об её основных приёмах, определить основные достоинства и недостатки, а также её сферу применения.

В наши дни разработка качественного программного продукта требует работы нескольких разработчиков, а всякая работа команды должна быть согласована. Для этого внутри команды применяются различные методологии разработки. Согласно выбранной методологии каждый этап логически планируется и все члены команды четко придерживаются этому плану.

Под методологией следует понимать набор правил, методов, способов и идей, определяющих процесс разработки программного продукта.

Так как методология определяет как будет выполняться разработка программного продукта, в разное время были сформулированы множество успешных методологий, позволяющих создавать качественный продукт. При этом выбор конкретной методологии для проекта зависит от различных критериев, так, например, от размера команды, сложности самого проекта, его критичности, а также, зачастую, от личных качеств сотрудников.

В данном реферате будет рассмотрено семейство гибких методологий *Crystal*.

ИСТОРИЯ ВОЗНИКНОВЕНИЯ МЕТОДОЛОГИИ

Автором данного семейства методологий является Алистер Кокберн (*Alistair Cockburn*). В 1991 году компания *IBM Consulting Group* попросила его создать методологию для объектно-технических проектов. Однако, не обладая достаточным опытом в сфере методологий он обратился за помощью к своей начальнице Кэти Улисс (*Kathy Ulisee*). По её совету Алистер начал проводить интервью с проектными командами. Их рассказы сильно отличались от того, что он ранее читал в книгах по методологиям. В частности, команды подчеркнули аспекты, не фигурирующие в текстах методологий: тесную связь, моральный дух, доступ к экспертам по предметной области и так далее.

Как ведущий консультант он опробовал полученные идеи на проекте из 45 человек с фиксированным бюджетом 15 000 000$. Эти идеи в сочетании с большим количеством творчества сработали как планировалось и показали себя главными факторами успеха. В работе «*Surviving Object-Oriented Projects*» он описал полученные им уроки в ходе проведения множество интервью и работы над этим проектом.

Проверив эти идеи на других проектах, он получил триаду основных требований, необходимых для продуктивной работы команды. В отличие от авторов других гибких методологий, Алистер Кокберн пришел к принципам гибкости в погоне за эффективностью, а не необходимостью адаптироваться к быстроизменяющимся требованиями.

В 2003 году описал свою методологию в книге «*Crystal Clear*».

СОДЕРЖАНИЕ МЕТОДОЛОГИИ

Crystal – это семейство методологий с общим генетическим кодом, который подчёркивает частые релизы, тесное общение и рефлективное совершенствование. Не существует единой *Crystal* методологии, для каждого типа проектов своя методология. В сущности, каждый проект, основываясь на генетическом коде *Crystal*, генерирует нового члена семейства.

Название *Crystal* происходит от классификации проектов в двух измерениях: размер и критичность – по аналогии с минералами, имеющими цвет и прочность. [1, с.4]

Цвета в *Crystal*, определяются соответственно количеству человек в команде:

1. *Clear* – 1-6 человек
2. *Yellow* – 7-20 человек
3. *Orange* – 21-40 человек
4. *Red* – 41-80 человек

И так далее для *Maroon*, *Blue* и *Violet*. Большие проекты требуют большего общения и координации действий в команде, и ссылаются на цвета темнее (*clear*, *yellow*, *orange*, *red* и так далее).

Прочность в зависимости от критичности проекта делится на 4 типа:

1. *Comfort* – в случае отказа теряется удобство
2. *Discretionary money* – теряется потенциальный доход
3. *Essential money* – теряется основной доход
4. *Life* – от работы зависит жизнь

Проекты с большей критичностью увеличивают показатель прочности методологии, требуя большей проверки и верификации правил.

Так на рисунке 1 показана сетка выбора нужного набора правил методологии в зависимости от условий. Содержимое каждой из ячеек обозначает конкретную методологию, так, например, значение *L6* подходит для команды из не более чем 6 членов, занимающихся жизненно важным проектом.

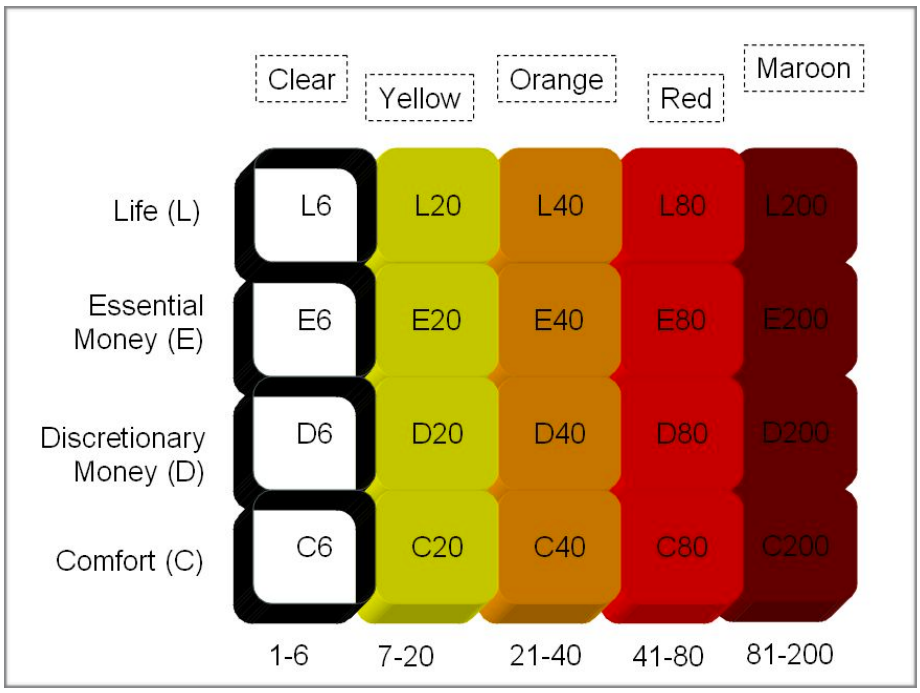

Рисунок 1 – Сетка выбора *Crystal*

Crystal, являясь итеративной методологией рекомендует адаптироваться к изменениям требований путём их фиксации на период одной итерации. [1, с.35] Этот подход позволяет сосредоточиться на конкретном задании и получить к концу каждой итерации некоторое состояние всего программного продукта. Разработчики при этом не остаются в подвешенном состоянии.

Генетический код Crystal состоит из:

- Экономико-кооперативная игровая модель
- Выбранных приоритетов
- Выбранных свойств
- Выбранных принципов
- Выбранных шаблонных техник и примеров проекта

Экономико-кооперативная игровая модель [1, с.5] считает разработку программного обеспечения серией «игр», чьи ходы состоят только из изобретения и общения. Каждая игра в серии имеет две цели, которые соревнуются за ресурсы: предоставить ПО в этой игре и подготовиться к следующим играм серии. Игра никогда не повторяется. Так каждый проект использует стратегию, отличающуюся от всех предыдущих игр. Экономико-

кооперативная игровая модель заставляет людей в проекте думать об их работе специфичным, сфокусированным и эффективным способом.

Общими для семейства *Crystal* являются следующие приоритеты:

- Безопасность доходов
- Эффективность разработки

Crystal направляет команду проекта следовать семи свойствам, первые три из которых являются основными. Остальные могут быть добавлены в любом порядке для увеличения безопасности.

- Частые релизы
- Тесное общение
- Рефлективное совершенствование
- Чувство безопасности
- Сконцентрированность
- Доступ к экспертам по предметной области
- Техническая среда с автоматическим тестированием, управлением конфигурациями и частой интеграцией.

Рассмотрим каждое из свойств детальнее:

1. Частые релизы

Самое важное свойство любого проекта, маленького или большого, гибкого или нет, это тестирование конечными пользователями, желательно, каждые несколько месяцев. Преимущества такого подхода заключаются в следующем:

+ Заказчик получает обратную связь в процессе разработки

+ Конечные пользователи получают возможность уточнить изначальные требования и предоставить их в качестве обратной связи

+ Разработчики остаются сфокусированными, преодолевая проблемы

+ Команда доходит до отладки и релиза, не теряя интерес к проекту

Релизы должны происходить не реже чем раз в 4 месяца, оптимальным вариантом является раз в 2 месяца. Релизы в сеть можно производить еженедельно.

Также постоянная интеграция должна стать нормой и производиться каждый час, день или в худшем случае неделю. Наилучшей практикой является автоматический цикл сборки-тестирования на автоматических тестах не менее раза в пол часа.

2. Тесное общение

Под тесным общением понимается, что вся информация протекает на фоне для каждого из членов команды таком образом, что они подбирают нужную информацию. Обычно это реализуется путём расположения команды в одном помещении. Таким образом, когда один человек задаёт вопрос, остальные в помещении могут вступить в обсуждение или подчерпнуть из него нужную информацию, продолжая работу.

Когда к тесному общению привыкают, вопросы и ответы протекают естественно, минимально мешая команде.

Тесное общение в совокупности с частыми релизами привносит быструю и качественную обратную связь. Именно поэтому эти два свойства указаны первыми.

3. Рефлективное совершенствование

Суть такого совершенствования в том, что команда должна составлять список того, что работает, а что нет и обсуждать что могло бы работать лучше, а главное – производить соответствующие изменения в следующей итерации. Не стоит тратить на это много времени, раза в пару недель – месяц будет достаточно. Сам факт отстранения от рутинной разработки и обсуждения того, что может работать лучше, уже эффективен.

В результате такой рефлексии начавшийся плохо или даже катастрофично проект может в итоге даже обогнать предполагаемый график.

4. Чувство безопасности

Это свойство подразумевает возможность членам команды говорить о том, что их беспокоит без негативных последствий для себя. Например, сказать

менеджеру, что сроки нереальны. а коллеге, что её разработка требует улучшений или даже дать ей понять, что стоит чаще принимать душ. [1, с.46] Чувство безопасности важно в первую очередь из-за того, что благодаря ей команда выявляет и устраняет свои слабости. Без неё люди не будут говорить, а слабость и дальше будут вредить команде.

Чувство безопасности – это начальный этап формирования доверия. Доверие в свою очередь позволяет команде свободно общаться, что значительно ускоряет проект. Исходя из этих факторов, чувство безопасности - критическое свойство, которого стоит придерживаться.

5. Сконцентрированность

Заключается в том, что бы понять, что необходимо сделать, а затем спокойно и с достаточным временем работать над этим. Понимание, что необходимо сделать приходит от расстановки целей и приоритетов лидером или заказчиком. Под спокойствием и достаточным временем понимается возможность людей работать, не будучи прерваны на другие задания.

6. Доступ к экспертам по предметной области

Непрерывный доступ к экспертам по предметной области предоставляет команде среду для интеграции и тестирования в рамках «частых релизов», быструю обратную связь по качеству их продукта, конструктивных решений и актуальные требования.

Даже один час в неделю прямого доступа к экспертам очень ценен. Чем больше часов каждую неделю эксперт по предметной области доступен команде, тем больше пользы они могут получить. Тем не менее, первый час является ключевым.

Другим важным фактором является скорость ответа на вопросы. Если ответ не был получен в течении 3-х дней, разработчик, скорее всего, реализует наилучшую свою догадку и в итоге может забыть перепроверить её в следующий раз. Для решения этой проблемы у разработчиков должна быть по крайней мере телефонная связь с экспертами на протяжении недели.

7. Техническая среда с автоматическим тестированием, управлением конфигурациями и частой интеграцией

Несмотря на то, что все составляющие данного свойства широко известны и применяются в наше время, стоит рассмотреть каждый из них по отдельности.

- Автоматическое тестирование. Для команд, использующих ручные тесты, этот фактор не так критичен. Однако, любой программист, однажды перешедший на автоматическое тестирования, никогда не вернётся к ручным тестам. Этот фактор помогает улучшить «качество жизни» разработчика и сократить его рутинную работу.
- Управление конфигурациями. Позволяет людям проверять их наработки асинхронно. Сохранять изменения, переключаться к нужной конфигурации в релизе, откатываться к другой в случае ошибки. Это помогает разрабатывать раздельно и, одновременно, вместе.
- Частая интеграция. Многие команды интегрируют систему несколько раз в день. Если они не могут с этим справится – они делают это раз в день или, в худшем случае, каждый следующий день. Чем чаще происходит интеграция, тем быстрее обнаруживаются ошибки и несоответствия, тем свежее взгляды команды.

Наиболее успешным решением будет объединить эти три составляющие в одну систему непрерывной интеграции с тестами. Такая система позволяет отлавливать ошибки на уровне интеграции в считаные минуты.

ИЗВЕСТНЫЕ ПРИМЕНЕНИЯ

В силу того, что оригинально методология была описана только в варианте *Cystal Clear*, подходящем для маленьких команд разработчиков, а для *Crystal Orange* был описан только набросок большинство историй применения не выходят за рамки команды в силу малой популярности самих продуктов.

Crystal Clear чаще всего применяется в условиях стартапов в которых команда разработчиков подходит под рамки этой методологии. А её применение оправданно её простотой и удобством, отсутствием строгих и сложных в реализации требований.

ЗАКЛЮЧЕНИЕ

Семейство методологий *Crystal* обладает достаточным выбором под каждый тип проектов. Предлагая более мягкий подход к работе разработчиков как личностей он является менее строгим, по отношению к экстремальному программированию. Однако, в силу своей «дружелюбности» принципы *Crystal* соблюдаются чаще.

Методология подходит для практически любой команды и задачи, но при этом требует грамотной расстановки приоритетов и выбора свойств. Основные принципы ставят разработчика как личность в команде в центре внимания, предоставляя ему возможность творческого развития. Командная же составляющая позволяет упростить и соответственно ускорить разработку проекта за счет тесного общения всех членов команды.

В сочетании с необязательными свойствами *Crystal* позволяет сократить затраты на рутинные занятия, улучшить отношения в команде, устранить её слабости и придерживаться актуальных требований заказчика, не изменяя их в процессе итерации.

СПИСОК ИСПОЛЬЗОВАНОЙ ЛИТЕРАТУРЫ

1. *Alistair Cockburn. Crystal Clear* [Электронный ресурс]. – http://users.dcc.uchile.cl/~nbaloian/cc1001-03/ejercicios/crystalclearV5d.pdf
2. *Alistair Cockburn. Cristal light methods* [Электронный ресурс]. – http://alistair.cockburn.us/Crystal+light+methods