Лекция 14. Некоторые UML дефициты.

Управление риском

Влияние риска вычисляют по выражению:

RE = P(UO) * Z(UO),

где: RE — показатель риска (Risk Exposure — подверженность риску);

P(UO) — вероятность неудовлетворительного результата (Unsatisfactory Outcome);

Z(UO) — потеря при неудовлетворительном результате.

При разработке программного продукта неудовлетворительным результатом может быть:

превышение бюджета;

низкая надежность;

неправильное функционирование и т. д.

Управление риском включает шесть действий:

Идентификация риска — выявление элементов риска в проекте.

Анализ риска — оценка вероятности и величины потери по каждому элементу риска.

Ранжирование риска — упорядочение элементов риска по степени их влияния.

Планирование управления риском — подготовка к работе с каждым элементом риска.

Разрешение риска — устранение или разрешение элементов риска.

Наблюдение риска — отслеживание динамики элементов риска, выполнение корректирующих действий.

Идентификация риска

Три категории источников риска:

проектный риск;

технический риск;

коммерческий риск.

Источники проектного риска:

выбор бюджета, плана, человеческих ресурсов программного проекта;

формирование требований к программному продукту;

сложность, размер и структура программного проекта;

методика взаимодействия с заказчиком.

К источникам технического риска относят:

трудности проектирования, реализации, формирования интерфейса, тестирования и сопровождения;

неточность спецификаций;

техническая неопределенность или отсталость принятого решения.

Главная причина технического риска — реальная сложность проблем выше предполагаемой сложности.

Источники коммерческого риска:

создание продукта, не требующегося на рынке;

создание продукта, опережающего требования рынка (отстающего от них);

потеря финансирования.

Лучший способ идентификации — использование проверочных списков риска, которые помогают выявить возможный риск.

Например, проверочный список десяти главных элементов программного риска:

Дефицит персонала.

Нереальные расписание и бюджет.

Разработка неправильных функций и характеристик.

Разработка неправильного пользовательского интерфейса.

Слишком дорогое обрамление.

Интенсивный поток изменения требований.

Дефицит поставляемых компонентов.

Недостатки в задачах, разрабатываемых смежниками.

Дефицит производительности при работе в реальном времени.

Деформирование научных возможностей.

На практике каждый элемент списка снабжается комментарием — набором методик для предотвращения источника риска. После идентификации элементов риска следует количественно оценить их влияние на программный проект, решить вопросы о возможных потерях. Эти вопросы решаются на шаге анализа риска.

Анализ риска

В ходе анализа оценивается вероятность возникновения P, и величина потери L, для каждого выявленного i -го элемента риска. В результате вычисляется влияние RE, i -го элемента риска на проект. Вероятности определяются с помощью экспертных оценок или на основе статистики, накопленной за предыдущие разработки.

Итоги анализа сводятся в таблицу 14:

Таблица. 14.

Итоги анализа

№	Элемент риска	Вероятность, %	Потери	Влияние риска
1.	Критическая программная ошибка	3-5	10	30-50
2.	Ошибка потери ключевых данных	3-5	8	24-40
3.	Отказоустойчивость недопустимо снижает производительность	4-8	7	28-56

4.	Отслеживание опасного условия как безопасного	5	9	45
5.	Отслеживание безопасного условия как опасного	5	3	15
6.	Аппаратные задержки срывают планирование	6	4	24
7.	Ошибки преобразования данных приводят к избыточным вычислениям	8	1	8
8.	Слабый интерфейс пользователя снижает эффективность работы	6	5	30
9.	Дефицит процессной памяти	1	7	7
10.	СУБД теряет данные	8	8	4

Ранжирование риска — назначение каждому элементу риска приоритета, который пропорционален влиянию элемента на проект.

Для больших проектов количество элементов риска велико (30-40). В этом случае к элементам риска применяют принцип Парето 80/20. В ходе ранжирования определяют 20% элементов риска (существенные элементы). В дальнейшем учитывается влияние только существенных элементов риска.

Планирование управления риском

Цель — сформировать набор функций управления каждым элементом риска.

При этом используют понятие эталонного уровня риска:

превышение стоимости;

срыв планирования;

упадок производительности.

Эталонные уровни риска могут быть причиной прекращения проекта. Если комбинация проблем, создающих риск, станет причиной превышения любого из этих уровней, работа будет остановлена. В фазовом пространстве риска эталонному уровню риска соответствует эталонная точка. В эталонной точке решения "продолжать проект" и "прекратить проект" имеют одинаковую силу.

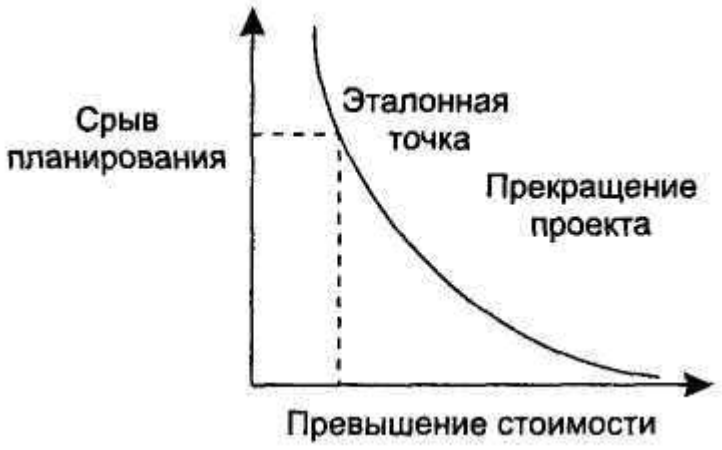

Рис. 64. Управление риском

Ниже кривой - рабочая область проекта, выше кривой — запретная область (при попадании в эту область проект должен быть прекращен).

Последовательность шагов планирования:

Исходные данные — набор четверок [R,P,L,RE].

Определяются эталонные уровни риска в проекте.

Разрабатываются зависимости между каждой четверкой [R,P,L,RE] и каждым эталонным уровнем.

Формируется набор эталонных точек, образующих сферу останова. В сфере останова предсказываются области неопределенности.

Для каждого элемента риска разрабатывается план управления. Предложения плана составляются в виде ответов на вопросы "зачем, что, когда, кто, где, как и сколько".

План управления каждым элементом риска интегрируется в общий план программного проекта.

Разрешение и наблюдение риска

Основанием для разрешения и наблюдения является план управления риском. Работы по разрешению и наблюдению производятся с начала и до конца процесса разработки. Разрешение риска состоит в плановом применении действий по уменьшению риска.

Наблюдение риска гарантирует:

цикличность процесса слежения за риском;

вызов необходимых корректирующих воздействий.

Для управления риском используется эффективная методика "Отслеживание 10 верхних элементов риска". Эта методика концентрирует внимание на факторах повышенного риска, экономит много времени, минимизирует "сюрпризы" разработки.

Рассмотрим шаги методики "Отслеживания 10 верхних элементов риска":

Выполняется выделение и ранжирование наиболее существенных элементов риска в проекте.

Производится планирование регулярных просмотров (проверок) процесса разработки. В больших проектах (в группе больше 20 человек) просмотр должен проводиться ежемесячно, в остальных проектах — чаще.

Каждый просмотр начинается с обсуждения изменений в 10 верхних элементах риска (их количество может изменяться от 7 до 12).

В обсуждении фиксируется текущий приоритет каждого из 10 верхних элементов риска, его приоритет в предыдущем просмотре, частота попадания элемента в список верхних элементов.

Если элемент в списке опустился, он по-прежнему нуждается в наблюдении, но не требует управляющего воздействия.

Если элемент поднялся в списке, или только появился в нем, то элемент требует повышенного внимания. Кроме того, в обзоре обсуждается прогресс в разрешении элемента риска (по сравнению с предыдущим просмотром).

Внимание участников просмотра концентрируется на любых проблемах в разрешении элементов риска.

3.3.3. Разработка в стиле экстремального программирования

Основная область применения XP — небольшие проекты с постоянно изменяющимися требованиями заказчика. Заказчик может не иметь точного представления о том, что должно быть сделано. Основным структурным элементом XP-процесса является XP-реализация. Структура XP-реализации показана на Рис. 65. Исходные требования к продукту фиксируются с помощью пользовательских историй.

Истории позволяют оценить время, необходимое для разработки продукта. Они записываются заказчиком и задают действия, которые должна выполнять для него программная система. Каждая история занимает три-четыре текстовых предложения в терминах заказчика. Кроме того, истории служат для создания тестов приемки. Тесты приемки используют для проверки правильности реализации пользовательских историй.

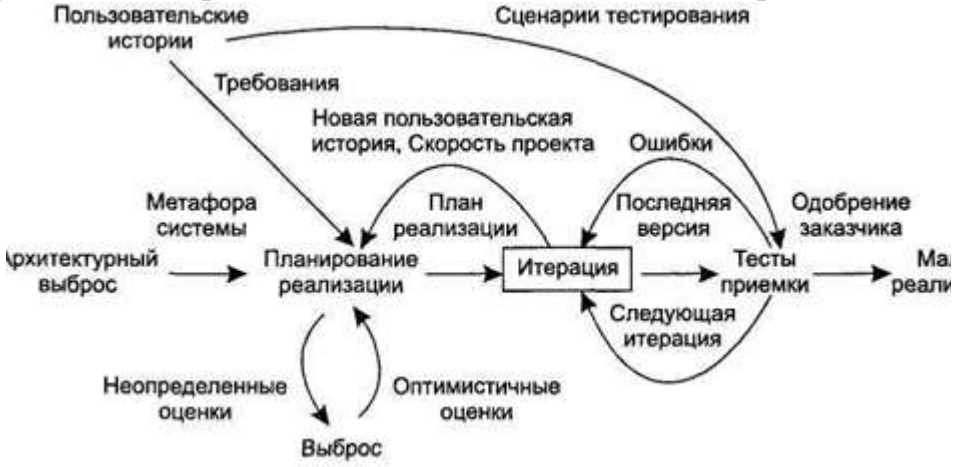

Рис. 65. Структура XP-реализации

Привыкнув не добавлять функциональность заранее и использовать краткосрочное планирование, разработчики смогут легко приспосабливаться к изменению требований заказчика.

Цель планирования, с которого начинается итерация, — выработать план решения программных задач.

Каждая итерация должна длиться от одной до трех недель. Пользовательские истории внутри итерации сортируются в порядке их значимости для заказчика. Кроме того, добавляются задачи, которые не смогли пройти предыдущие тесты приемки и требуют доработки.Пользовательским историям и тестам с отказом в приемке сопоставляются задачи программирования.

Задачи записываются на карточках, совокупность карточек образует детальный план итерации. Для решения каждой из задач требуется от одного до трех дней. Задачи, для которых нужно менее одного дня, группируются вместе, а большие задачи разбивают на более мелкие задачи.

Разработчики оценивают количество и длительность задач. Для определения фиксированной длительности итерации используют метрику

"скорость проекта", вычисленную по предыдущей итерации (количество завершенных в ней задач/дней программирования).

Если предполагаемая скорость превышает предыдущую скорость, заказчик выбирает истории, которые следует отложить до более поздней итерации.

Если итерация слишком мала, к разработке принимается дополнительная история. Вполне допустимая практика — переоценка историй и пересмотр плана реализации после каждых трех или пяти итераций.

Первоочередная реализация наиболее важных историй — гарантия того, что для клиента делается максимум возможного. Стиль разработки, основанный на последовательности итераций, улучшает подвижность процесса разработки.

В каждую XP-итерацию многократно вкладывается строительный элемент — элемент XP-разработки. Рассмотрим организацию элемента XP-разработки

Элемент XP-разработки

Структура элемента XP-разработки показана на Рис. 66. День XP-разработчика начинается с установочной встречи. Ее цели: обсуждение проблем, нахождение решений и определение точки приложения усилий всей команды.

Участники утренней встречи стоят и располагаются по кругу, так можно избежать длинных дискуссий. Все остальные встречи проходят на рабочих местах, за компьютером, где можно просматривать код и обсуждать новые идеи.

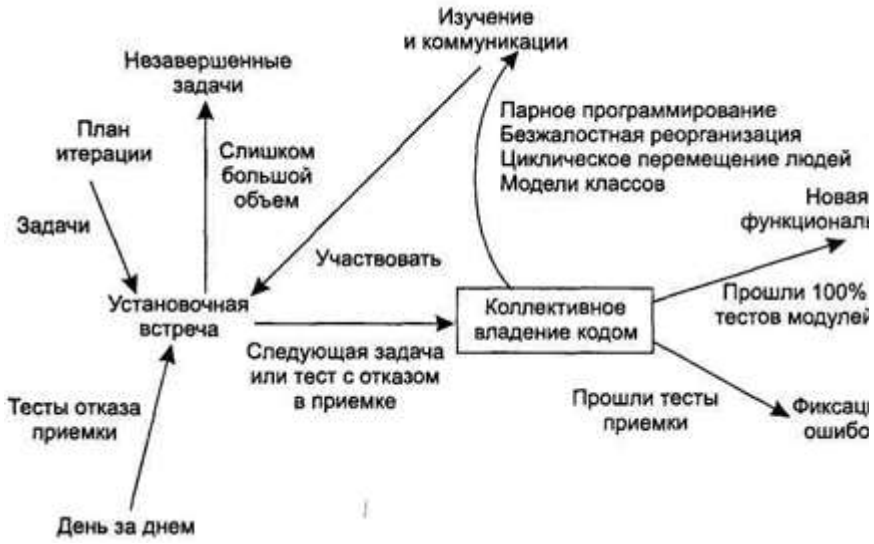

Рис. 66. Структура элемента XP-разработки

Весь день XP-разработчика проходит под лозунгом коллективного владения кодом программной системы. В результате этого происходит фиксация ошибок и добавление новой функциональности в систему. Следует удерживаться от соблазна добавлять в продукт функциональность, которая будет востребована позже. Полагают, что только 10% такой функциональности будет когда-либо использовано, а потери составят 90%

времени разработчика. В XP считают, что дополнительная функциональность только замедляет разработку и исчерпывает ресурсы.

Коллективное владение кодом

Организацию коллективного владения кодом иллюстрирует Рис. 67.

Коллективное владение кодом позволяет каждому разработчику выдвигать новые идеи в любой части проекта, изменять любую строку программы, добавлять функциональность, фиксировать ошибку и проводить реорганизацию.

Один человек не в состоянии удержать в голове проект нетривиальной системы. Благодаря коллективному владению кодом снижается риск принятия неверного решения (главным разработчиком) и устраняется нежелательная зависимость проекта от одного человека. Работа начинается с создания тестов модуля, она должна предшествовать программированию модуля. Тесты необходимо помещать в библиотеку кодов вместе с кодом, который они тестируют. Тесты делают возможным коллективное создание кода и защищают код от неожиданных изменений. В случае обнаружения ошибки также создается тест, чтобы предотвратить ее повторное появление. Кроме тестов модулей, создаются тесты приемки, они основываются на пользовательских историях. Эти тесты испытывают систему как "черный ящик" и ориентированы на требуемое поведение системы.

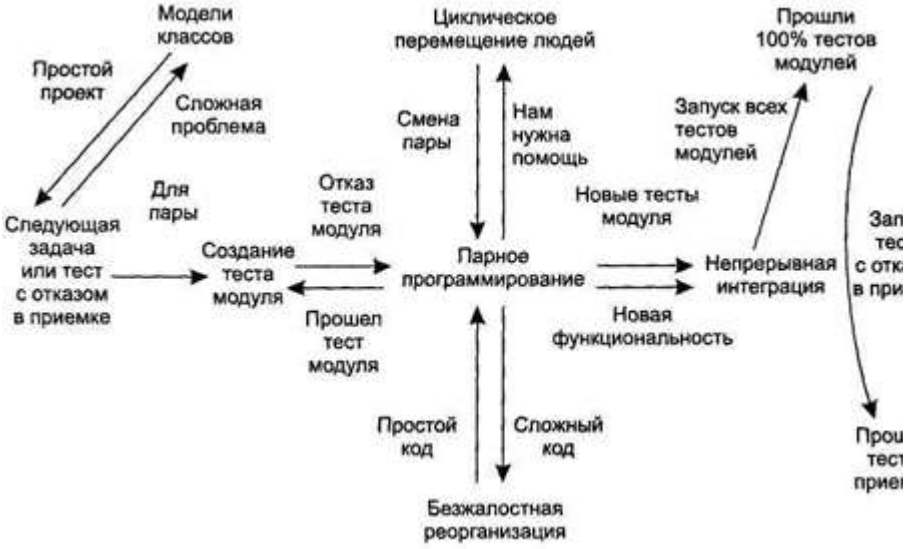

Рис. 67. Организацию коллективного владения кодом

На основе результатов тестирования разработчики включают в очередную итерацию работу над ошибками. Тестирование — один из краеугольных камней XP. Все коды в проекте создаются парами программистов, работающими за одним компьютером. Парное программирование приводит к повышению качества без дополнительных затрат времени. А это, в свою очередь, уменьшает расходы на будущее сопровождение программной системы.

Оптимальный вариант для парной работы — одновременно сидеть за компьютером, передавая друг другу клавиатуру и мышь. Пока один человек

набирает текст и думает (тактически) о создаваемом методе, второй думает (стратегически) о размещении метода в классе.

Во время очередной итерации всех сотрудников перемещают на новые участки работы. Такие перемещения помогают устранить изоляцию знаний и "узкие места". Особенно полезна смена одного из разработчиков при парном программировании.

В XP считают, что код нужно постоянно обновлять — удалять лишние части, убирать ненужную функциональность. Этот процесс называют реорганизацией кода (refactoring). Поощряется безжалостная реорганизация, сохраняющая простоту проектных решений. Реорганизация поддерживает прозрачность и целостность кода, обеспечивает его легкое понимание, исправление и расширение. На реорганизацию уходит значительно меньше времени, чем на сопровождение устаревшего кода. Увы, нет ничего вечного — когда-то отличный модуль теперь может быть совершенно не нужен.

Еще одна составляющая коллективного владения кодом — непрерывная интеграция. Без последовательной и частой интеграции результатов в систему разработчики не могут быть уверены в правильности своих действий. Кроме того, трудно вовремя оценить качество выполненных фрагментов проекта и внести необходимые коррективы.

По возможности XP-разработчики должны интегрировать и публично отображать, демонстрировать код каждые несколько часов. Интеграция позволяет объединить усилия отдельных пар и стимулирует повторное использование кода.

Взаимодействие с заказчиком

Одно из требований XP — постоянное участие заказчика в проведении разработки. По сути, заказчик является одним из разработчиков. Все этапы XP требуют непосредственного присутствия заказчика в команде разработчиков. Причем разработчикам нужен заказчик-эксперт. Он создает пользовательские истории, на основе которых оценивается время и назначаются приоритеты работ.

В ходе планирования реализации заказчик указывает, какие истории следует включить в план реализации. Активное участие он принимает и при планировании итерации.

Заказчик должен как можно раньше увидеть программную систему в работе. Это позволит как можно раньше испытать систему и дать отзыв о ее работе.

Поскольку при укрупненном планировании заказчик остается в стороне, разработчикам нужно постоянно общаться с заказчиком, чтобы получать как можно больше сведений при реализации задач программирования. Нужен заказчик и на этапе функционального тестирования, при проведении тестов приемки.

Таким образом, активное участие заказчика не только предотвращает появление некачественной системы, но и является непременным условием выполнения разработки.