Текст 1.

Что такое программная инженерия?

«Для оптимиста стакан наполовину полон.

Для пессимиста стакан наполовину пуст.

Для инженера стакан в два

раза больше, чем должен быть.»

Анонимный

«Информатика связана с компьютерами не больше, чем астрономия

- с телескопами.»

Эдсгер У. Дейкстра.

Большинство книг по программной инженерии оставляют у вас впечатление, что разработка программного обеспечения - это прежде всего управление проектами. Другими словами, речь идет не об инженерии как таковой, а скорее о том, как вы подходите к разработке программного обеспечения, в частности, о знании того, какие шаги следует предпринять и как вы их сочетаете. Управление, безусловно, важно, особенно потому, что большинство программных проектов выполняются командами, но инженерным аспектом этого в конечном итоге пренебрегают. Здесь мы больше сосредоточимся на инженерных аспектах разработки программного обеспечения. Мы стараемся показать, что существуют интересные интеллектуальные головоломки и захватывающие решения, и что разработка программного обеспечения - это не только наставление за наставлением или кулинарная книга по логистике и менеджменту.

Вычислительная система (далее называемая system) - это компьютерная система, целью которой является ответ на вопросы о действиях человека в некоторой области и/или поддержка их. Мы говорим, что система относится к своей предметной области. Она включает в себя внутренние структуры, представляющие предметную область. Эти структуры включают данные, представляющие сущности и отношения в предметной области, и программу, предписывающую, как можно манипулировать этими данными. Вычисление фактически происходит, когда процессор (интерпретатор или CPU) выполняет эту программу или ее часть.

Мы надеемся донести в этом тексте, что программное обеспечение состоит из множества частей, каждая из которых по отдельности проста, но проблема в том, что их слишком много. Дело не в сложности отдельных компонентов; вас ошеломляет их множество - вы просто теряете представление о деталях. Давайте проиллюстрируем это на простом примере. Предположим, кто-то хочет построить забор вокруг дома. Строительство включает в себя четыре задачи: установку столбов, распиловку дерева, покраску и забивание гвоздей. Установка стоек должна предшествовать покраске и прибиванию гвоздей, а обрезка должна предшествовать прибиванию гвоздей. Предположим, что установка стоек занимает 3 единицы времени, распиловка древесины занимает 2 единицы времени, покраска занимает 5 единиц времени для необрезанной древесины и 4 единицы времени в противном случае, а забивание гвоздей занимает 2 единицы времени для неокрашенной древесины и 3 единицы времени в противном случае. В каком порядке следует выполнять эти задачи, чтобы завершить проект в кратчайшие сроки?

Трудно прийти к правильному решению (или решениям), не записав возможные варианты и не рассмотрев их один за другим (если только они не поставляются со встроенным компьютерным чипом). Трудно сказать, почему эта проблема сложна, потому что ни один отдельный шаг не кажется сложным.

В конце концов, самая сложная операция включает в себя сложение небольших целых чисел. Разработка программного обеспечения полна проблем, подобных этой: все отдельные шаги просты, но общая проблема непреодолима.

Проблема в том, что для сдержанной логики близость к правильности неприемлема; один перевернутый бит может изменить весь смысл программы.

Инженерам-программистам нравится думать, что их проблемы уникальны для всей инженерии или даже для более широких проблем человечества. Но классический детский стишок напоминает нам, что опасность не замечать деталей была известна, по крайней мере, еще со времен битвы при Босворт-Филд и времен Ричарда III (1452-1485).:

Из-за отсутствия гвоздя была потеряна подкова.

Из-за отсутствия подковы была потеряна лошадь.

Из-за отсутствия лошади всадник пропал.

Разработчики программного обеспечения еще не нашли адекватных методов для решения такой сложности, и этот текст в основном посвящен представлению текущего состояния знаний о том, как справиться со сложностью разработки программного обеспечения. Разработку программного обеспечения часто путают с программированием. Роль программной инженерии заключается в том, чтобы уловить бизнес-потребности заказчика и определить «чертежи» системы, чтобы программисты могли ее реализовать.

Разработка программного обеспечения опирается на нашу способность мыслить о пространстве и времени, процессах и взаимодействиях между процессами и структурами.

Рассмотрим пример разработки программной системы для управления банковским автоматом, известным как банкомат (ATM).

Большинство из нас не знают, что на самом деле происходит внутри банкомата; тем не менее, мы могли бы предложить наивное объяснение того, как работают банкоматы. Мы знаем, что банкомат позволяет нам вносить или снимать деньги, и мы можем представить, как разделить эти действия на более простые, которые будут выполняться воображаемыми маленькими «агентами», работающими внутри автомата.

Мы знаем, что банкомат играет роль банковского служащего в окне (кассира). Читатель может задаться вопросом, почему мы должны представлять себе множество виртуальных агентов, выполняющих работу одного кассира. Почему бы просто не представить себе одного виртуального агента, выполняющего работу кассира?! Причина в том, что это не сильно помогло бы, потому что все, чего мы достигли бы, — это преобразовали бы один сложный и непостижимый объект (банкомат) в другой сложный и непостижимый объект (виртуальный кассир). Чтобы понять сложную вещь, нужно развить представления о взаимосвязях между внутренними частями. Разделяя сложную работу на более простые задачи и описывая, как они взаимодействуют, мы упрощаем проблему и облегчаем ее понимание и решение. Вот почему воображение имеет решающее значение для разработки программного обеспечения (как и для любой другой деятельности по решению проблем!).

Конечно, недостаточно раскрыть статическую структуру планируемой системы. Нам также необходимо описать, как элементы системы ("работники" и "вещи") взаимодействуют во время выполнения задачи.

В некотором смысле разработка программного обеспечения параллельна стратегиям решения проблем в искусственном интеллекте (ИИ): представляет цель; представляет текущее состояние; и сводит к минимуму различия с помощью анализа средств и целей. Как и в случае с любым дизайном, разработку программного обеспечения можно рассматривать как деятельность по уменьшению различий, сформулированную в терминах символического описания различий. Наконец, при автономных вычислениях цели явно представлены в программе, реализующей систему.

Язык программирования, как и любой другой формальный язык, представляет собой набор символов и правил для манипулирования ими. Именно тогда, когда им нужно познакомиться с реальным миром, вы обнаруживаете, что ассоциации могут создаваться по-разному, и некоторые правила не были указаны. Новичок часто видит только преимущества создания программного продукта и игнорирует риски.

Эксперт видит более широкую картину и предвидит риски. В конце концов, разделение проблемы на подзадачи и решение их по частям не гарантирует логической строгости и строгой согласованности между частями. Риски обычно включают в себя такие условия, как то, что программа может делать то, что от нее ожидается, а затем еще некоторые неожиданные возможности.

Другой риск заключается в том, что не все состояния окружающей среды заносятся в каталог до начала разработки программы. В зависимости от того, как вы формулируете свои предположения, вы можете найти решение. Проблемы возникают, если предположения оказываются неточными, неправильными или изменяются в связи с меняющимся миром.

У нас всегда есть: ПОЛЬЗОВАТЕЛЬ <" КОМПЬЮТЕРНАЯ СИСТЕМА < ОКРУЖАЮЩАЯ СРЕДА.

При разработке системы мы моделируем пользователя и среду в компьютерной системе.

Моделирование обязательно предполагает упрощение и абстрагирование. Цель упрощения - сделать разработку управляемой: система должна решать одну проблему, а не все проблемы.

Текст 2.

Почему разработка программного обеспечения сложна

«Программное обеспечение подобно энтропии. Его трудно ухватить, он ничего не весит и подчиняется второму закону термодинамики, то есть всегда увеличивается.>

Норман Р. Огастин

Если вы инженер-строитель, строящий мосты, то все, что вам нужно знать, - это о мостах. В отличие от этого, если вы разрабатываете программное обеспечение syftware, вам нужно знать о предметной области программного обеспечения (потому что это то, что вы создаете), и вам нужно знать о проблемной области (потому что это то, для чего вы создаете решение). Некоторые проблемы требуют длительных периодов целенаправленных исследований (годы, десятилетия или даже дольше). Очевидно, что мы не можем рассматривать исследование таких проблем как часть разработки программного обеспечения. Мы будем предполагать, что проблема либо решена (теоретическое решение существует), либо она может быть решена за относительно короткий промежуток времени информированным неспециалистом.

Еще одна проблема заключается в том, что программное обеспечение является формальной областью, в то время как реальный мир неформален. Решение проблем в этих разных областях требует разных стилей, и в конечном итоге необходимо согласовать эти стили. Узкая интерпретация разработки программного обеспечения имеет дело только с разработкой самого программного обеспечения. Это означает, что при наличии точного определения того, что должно быть запрограммировано, узкоспециализированная программная инженерия занимается проектированием, внедрением и тестированием программы, представляющей собой решение заявленной проблемы.

Более широкая интерпретация разработки программного обеспечения включает в себя поиск решения реальной проблемы. Реальная проблема может не иметь ничего общего с программным обеспечением. Например, реальной проблемой может быть медицинская проблема наблюдения за пациентом или финансовая проблема разработки торговых стратегий. В широкомасштабной разработке программного обеспечения нет точного определения того, что должно быть запрограммировано.

Наша задача сводится ни много ни мало к разработке изменений в текущей деловой практике. Можно было бы пойти еще дальше и утверждать, что второй закон термодинамики работает против инженеров-программистов (или кого-либо еще, пытающегося строить модели мира). Разработка программного обеспечения — это в основном моделирование физического мира и поиск хороших абстракций. Если вы находите репрезентативный набор абстракций, разработка протекает естественным образом. Однако поиск абстракций в проблемной области (также известной как «прикладная область») требует определенного уровня «грубой детализации".» Это означает, что наши абстракции неизбежно являются всего лишь приближениями - мы не можем описать проблемную область в мельчайших деталях: в конце концов, для этого потребовалось бы работать на уровне атомных или даже субатомных частиц. Учитывая, что каждая физическая система состоит из очень многих частей, лучшее, что мы можем сделать, — это описать ее в терминах только некоторых из ее переменных. Работа с приближениями сама по себе не обязательно является проблемой, если структура мира никогда не меняется. Но второй закон термодинамики гласит, что Вселенная имеет тенденцию к возрастанию беспорядка. Мы живем в меняющемся мире: вещи изнашиваются и ломаются, организации банкротятся, приобретаются или реструктурируются, меняются методы ведения бизнеса и так далее. Какой бы порядок ни был зафиксирован в тех сравнительно немногих переменных, с которых мы начали, со временем он имеет тенденцию рассеиваться по другим переменным, где он больше не считается порядком. Наши (приблизительные) абстракции обязательно становятся недействительными с течением времени, и нам нужно начинать все сначала. Это требует времени и ресурсов, которых у нас может не быть в наличии.

Разработка программного обеспечения по-прежнему во многом зависит от героических усилий нескольких избранных разработчиков. Стандартизация продуктовой линейки и разработки по-прежнему в значительной степени отсутствует, но усилия в этом направлении предпринимаются. Инструменты и показатели для разработки продукта и управления проектами являются ключевыми.

Текст 3.

Жизненный цикл разработки программного обеспечения

Алгоритм решения задач Фейнмана предлагает следующий алгоритм:

(i) Запишите проблему, (ii) хорошенько подумайте и (iii) запишите ответ.

Можно ожидать, что разработка любого продукта будет проходить как организованный процесс, который обычно включает следующие этапы:

- Планирование/спецификация.

- Дизайн.

- Внедрение.

- Оценка.

То же самое относится и к разработке программного обеспечения. Общие этапы разработки программного обеспечения, следующие:

1. Спецификация/требования:

- Документирование сценариев использования и вывод статической модели предметной области.

2. Проектирование:

- Распределение обязанностей между объектами и определение динамики их взаимодействий при различных сценариях использования.

3. Реализация:

- Кодирование дизайна на языке программирования.

4. Тестирование:

- Отдельные классы/компоненты (модульное тестирование) и всю систему в целом (интеграционное тестирование).

5. Эксплуатация и техническое обслуживание:

- Запуск системы.

- Исправление ошибок и добавление новых функций.

Жизненный цикл обычно включает в себя множество других мероприятий, некоторые из которых предшествуют вышеуказанным, например маркетинговое исследование для определения потребности рынка в планируемом продукте.

Раннее вдохновение для разработки жизненного цикла программного обеспечения пришло из других инженерных дисциплин, где вышеуказанные действия обычно выполняются последовательно (или, по крайней мере, так считалось). Этот метод известен как водопадный процесс, потому что разработчики создают монолитные системы одним махом. Это требует завершения работы с артефактами текущего этапа, прежде чем переходить к следующему. В гражданском строительстве такой подход привел бы к следующему: аккуратно завершите все чертежи перед началом строительства; завершите конструкцию перед проверкой ее на прочность; и т.д.

Однако с годами разработчики поняли, что

разработка программного обеспечения в этих аспектах отличается от разработки любого другого продукта:

- В отличие от большинства других продуктов, программное обеспечение неосязаемо и его трудно визуализировать. Большинство людей знакомятся с программным обеспечением через то, что оно делает: какие входные данные оно принимает и что генерирует в качестве выходных данных.

- Программное обеспечение, вероятно, является самым сложным артефактом - большой программный продукт состоит из множества фрагментов, а также их взаимосвязей, каждый из которых играет важную роль - один перевернутый бит может изменить весь смысл программы.

- Программное обеспечение, вероятно, является самым гибким артефактом - оно может быть легко и радикально модифицировано на любом этапе процесса разработки (или, по крайней мере, так воспринимается). Эти идеи привели к внедрению инкрементальных и итеративных (или эволюционных) методов разработки.

Популярный инкрементальный и итеративный процесс называется унифицированным

Процесс.

Каждая итерация должна включать в себя более глубокое прохождение проекта. Инкрементный и итеративный процесс направлен на то, чтобы как можно скорее добраться до рабочего экземпляра. Наличие рабочего экземпляра позволяет заинтересованным сторонам иметь что-то осязаемое, с чем можно поиграть и навести справки. В ходе таких экспериментов (предпочтительно конечными пользователями) обнаруживаются неожиданные недостатки, которые запускают новый виток разработки, используя сбои и знания о вещах, которые не сработали бы, в качестве трамплина для новых подходов. Это в значительной степени способствует достижению консенсуса и формированию понимания всеми сторонами того, что необходимо разработать и чего ожидать по завершении.

Итак, ключом к инкрементным и итеративным методам является постепенное углубление понимания или «визуализации» целевого продукта, как продвигаясь вперед, так и возвращаясь к более ранним действиям, чтобы заново открыть больше его возможностей. Методы, которые являются еще более агрессивными с точки зрения коротких итераций и активного вовлечения пользователей, характеризуются как гибкие.

Все процессы жизненного цикла преследуют цель постепенного совершенствования дизайна продукта, но разные люди придерживаются разных взглядов на то, как этого достичь. Это было верно в прошлом и остается верным по сей день, и время от времени мы будем комментировать различные подходы.

Мы с энтузиазмом поддерживаем поэтапный и итеративный подход, и в этом духе изложение в этом тексте продвигается поэтапно и итеративно, последовательно разрабатывая этапы жизненного цикла программного обеспечения. По каждой новой теме мы будем касаться поверхности и двигаться дальше только для того, чтобы вернуться к ней позже и копнуть глубже.

Беглый обзор существующих учебников по программной инженерии показывает, что разработка программного обеспечения в значительной степени связана с управлением. Управление проектом требует организационных и управленческих навыков, таких как определение и организация множества задач, составляющих проект, распределение ресурсов для выполнения этих задач и отслеживание фактического и ожидаемого использования ресурсов.

Успешные программные проекты представляют собой сочетание тщательной объективной оценки,

адекватной подготовки, постоянного наблюдения и

оценки окружающей среды и прогресса, а также корректировки тактики.

Интересно сравнить проблемы, рассмотренные Бруксом [1975], и проблемы недавнего движения за гибкие методы - оба делают акцент на коммуникации членов команды разработчиков. Поэтому наша важная цель здесь - представить инструменты, облегчающие общение между разработчиками. Ключевыми такими инструментами являются:

- Модульное проектирование: Разбиение системы на модули помогает справиться со сложностью; однако модульизация может быть выполнена многими способами с различным качеством результатов.

- Язык символов: Унифицированный язык моделирования (UML) используется аналогично тому, как в математике используются такие символы, как o, J, d. Они сокращают изложение материала и облегчают читателю понимание материала.

- Показатели проекта и продукта: Показатели для планирования и измерения хода реализации проекта, а также показатели для измерения качества программных продуктов необходимы для инженерного подхода к разработке программного обеспечения.

- Эвристика проектирования: также известная как шаблоны, они передают лучшие практики, которые были зарекомендовали себя во многих контекстах и проектах.

Разложение проблемы на более простые, так называемый подход "разделяй и властвуй", распространен при решении сложных проблем.

В разработке программного обеспечения это воплощается в модульности: исходный код для модуля может быть написан и поддерживаться независимо от исходного кода для других модулей. Как и в любой другой деятельности, ценность структурированного подхода к разработке программного обеспечения становится очевидной только при решении сложных задач.

Текст 4.

Язык символов

«Без образов мы не можем ни мыслить, ни понимать что-либо.»

Мартин Лютер (1483-1546)

«В этом мире есть только 10 типов людей

Текст для чтения и перевода 4

Те, кто знает двоичный код, и те, кто его не знает.»

Неизвестный

Как часть процесса проектирования, важно донести свои идеи до общественности. Описывая процесс достижения определенной цели, человек на самом деле мыслит в терминах аббревиатур и символов, поскольку они описывают «детали» того, что он делает, и не смог бы действовать разумно, если бы он этого не делал. Джордж Миллер обнаружил в 1950-х годах, что кратковременная память человека может хранить около семи элементов одновременно. Кратковременная память — это то, что мы используем, например, для запоминания телефонного номера ровно настолько, чтобы оторвать взгляд от бумаги, на которой он написан, и набрать номер. Она также известна как рабочая память, потому что в ней предполагается, что информация обрабатывается при первом восприятии. Его сравнивают с оперативной памятью компьютера.

Вспомните, сколько раз вам приходилось оглядываться назад в процессе набора номера, особенно если вы не знакомы с кодом города, из-за чего номер становится сложной цифрой! Оказывается, гипотеза Миллера справедлива для любых семи «элементов», которые могут быть чем угодно, например числами, лицами, людьми или сообществами - поскольку мы организуем информацию на более высоких уровнях абстракции, мы все еще можем запомнить семь из того, что это такое.

Это называется разбиением на фрагменты.

Символы проще разбить на шаблоны, которые представлены новыми символами. Использование символов и иерархической абстракции облегчает людям представление о сложных системах.

Как можно заметить по всему этому тексту, базовые обозначения часто тривиальны и могут быть освоены относительно быстро. Ключ заключается в навыках создания различных моделей. Приобретение этого опыта может занять значительное количество времени.

Нашим основным языком символов является UML, но он не строго соблюдается во всем тексте. Мы будем использовать другие обозначения или текст, разработанный специально для рекламы, если сочтем, что это передает сообщение более элегантным способом.

Чтобы ознакомиться с UML, вы можете начать с http://www.uml.org, который является официальным веб-сайтом стандарта. Люди обычно используют разные символы для разных целей и на разных стадиях развития.

Во время разработки есть много способов обдумать ваш дизайн и много способов неформально описать его. Любая модель дизайна или язык моделирования имеют пределы того, что они могут выразить, и ни один взгляд на дизайн не говорит всего. Например, строгое следование стандарту может быть обременительным для первоначальных эскизов; напротив, всегда рекомендуется документировать завершенный дизайн в UML.

UML, вероятно, является наиболее широко принятой графической нотацией для проектирования программного обеспечения. Однако это не является общепринятым и часто подвергается критике (например, за непоследовательность), но все согласятся с тем, что символы полезны, поэтому разные авторы изобретают свои собственные любимые символы. Даже в математике, высшем языке символов, существуют разногласия по поводу символов даже для таких устоявшихся дисциплин, как математический анализ (ср. Ньютоновский против Символы Лейбница для математического анализа), чтобы не затрагивать более поздние темы. Подводя итог, вы можете изобрести свои собственные символы, если считаете это абсолютно необходимым, но прежде чем использовать их, объясните их значение / семантику и убедитесь, что всегда легко найти значения ваших символов.

Текст 5.

Анализ требований и спецификация системы

Мы начинаем с изложения требований заказчика, если проект спонсируется конкретным заказчиком, или с изложения видения, если у проекта нет спонсора. Заявление о видении аналогично заявлению о требованиях заказчика в том смысле, что в нем кратко описывается, что представляет собой предполагаемая система, за которым следует список функций/ услуг, которые она будет предоставлять, или задач/действий, которые она будет поддерживать.

Учитывая изложенные заказчиком требования, первый шаг в процессе разработки программного обеспечения называется анализом требований или системным анализом. Во время этого действия разработчик пытается расширить и дополнить формулировку требований и подготовить спецификацию системы - документ, который является точным описанием того, что должна делать планируемая система. Анализ требований разграничивает систему и определяет предлагаемые ею услуги, определяет типы пользователей, которые будут взаимодействовать с системой, и идентифицирует другие системы, которые взаимодействуют с нашей. Сначала система рассматривается как черный ящик, идентифицируются ее службы («кнопки») и детализируются типичные сценарии взаимодействия для каждой службы. Анализ требований включает в себя как установление фактов о том, как проблема решается в текущей практике, так и представление о том, как могла бы работать планируемая система.

Вспомним пример с банкоматом. Мы определили соответствующих игроков в нем. Однако это может оказаться слишком большим скачком для сложной системы. Более постепенный подход заключается в том, чтобы начать с выявления внешних игроков (называемых «акторами») и их пошагового взаимодействия. То, что происходит внутри системы, рассматривается отдельно на этапах проектирования и анализа.

Популярным методом анализа требований является моделирование вариантов использования.

Набор вариантов использования описывает элементарные задачи, которые должна выполнять система, и связь между этими задачами и внешним миром. Описание каждого варианта использования представляет собой диалог между пользователем и системой, направленный на то, чтобы помочь пользователю достичь бизнес-цели. В каждом диалоговом окне пользователь инициирует действия, а система отвечает реакцией. Варианты использования определяют, какая информация должна выходить за пределы системы в ходе диалога (без учета того, что происходит внутри системы).

Поскольку варианты использования представляют собой рецепты для достижения целей пользователем, название каждого варианта использования должно включать глагол, отражающий достижение цели. Ключевые действия обобщены в артефакте, называемом диаграммой вариантов использования.

Варианты использования — это только начало процесса разработки программного обеспечения.

Когда мы рисуем и разрабатываем схему вариантов использования системы, это означает, что мы знаем, чего система должна достичь, а не как; следовательно, это не просто «небольшой вопрос построения системы» (программирования), который остается после того, как мы определим варианты использования.