Лекция 1: Об архитектуре

Юрий Литвинов
y.litvinov@spbu.ru

1. Введение

Большая часть программ, за которые люди готовы платить деньги, довольно сложна. Даже небольшое по промышленным меркам приложение требует порядка двух-трёх человеко-лет на разработку и имеет размер в несколько десятков тысяч строк кода. Большие программные системы могут иметь несколько миллионов строк кода и занимать сотни и тысячи человеко-лет разработки. При этом каждый из вас понимает, что может писать гораздо больше 100 строчек кода в день, а "за выходные" вообще может сделать практически всё. Тут весь вопрос в том, что же по факту создаётся.

На картинке выше (книга Брукса хоть и устарела местами, но во многом не теряет своей актуальности) слева вверху показана программа – законченный кусок кода, который его автор может скомпилировать, запустить на своем устройстве и который выполняет нужную задачу. Это именно то, чем неопытные программисты привыкли оценивать свою производительность. Но за такой код мало кто готов платить или использовать его в своих проектах.

При перемещении по стрелке вниз программа превращается в программный продукт. Это программа, которую любой человек может запускать, тестировать, исправлять и развивать. Она может использоваться в различных операционных средах и со многими наборами данных. Чтобы стать общеупотребительным программным продуктом, программа должна быть написана в обобщенном стиле. В частности, диапазон и вид входных данных должны быть настолько обобщенными, насколько это допустимо. Затем программу нужно тщательно протестировать, чтобы быть уверенным в ее надежности. Для этого нужно подготовить достаточное количество контрольных примеров для проверки диапазона допустимых значений входных данных и определения его границ, обработать эти примеры и зафиксировать результаты. Приложению нужен хороший пользовательский интерфейс, который нужно будет ещё разработать. Наконец, развитие программы в программный продукт требует создания подробной документации, с помощью которой каждый мог бы использовать ее, делать исправления и расширять. Ну и сама программа должна быть написана так, чтобы её потенциально можно было адекватно расширять и сопровождать. По разным эмпирическим оценкам программный продукт стоит в 3-10 раз дороже, чем просто отлаженная программа с такой же функциональностью.

При пересечении вертикальной границы программа становится компонентом программного комплекса. Последний представляет собой программную систему, в состав которой входит набор взаимодействующих программ, согласованных по программным интерфейсам и форматам данных, и все вместе решающие какие-то более сложные задачи,

чем каждая по отдельности. Хорошим примером тут можно считать консольные утилиты *nix. Чтобы стать частью программного комплекса, синтаксис и семантика ввода и вывода программы должны удовлетворять точно определенным интерфейсам. Программа должна быть также спроектирована таким образом, чтобы использовать заранее оговоренный объём ресурсов — объем памяти, устройства ввода/вывода, процессорное время. Наконец, программу нужно протестировать вместе с прочими системными компонентами во всех сочетаниях, которые могут встретиться. Это тестирование может оказаться большим по объему, поскольку количество тестируемых случаев растет экспоненциально. Оно также занимает много времени, так как скрытые ошибки выявляются при неожиданных взаимодействиях отлаживаемых компонентов. Компонент программного комплекса стоит примерно так же в 3-10 раз дороже, чем автономная программа с теми же функциями. И этот порядок может увеличиться, если в системе много компонентов.

Так чем программирование отличается от разработки коммерческих проектов? Для начала, первое является некоторой абстрактной деятельностью и может происходить во многих различных контекстах. Можно программировать для того, чтобы собственно научиться программировать (например, на занятиях в университете), для удовольствия (например, сделать игру и играть в неё с друзьями), можно программировать в рамках научных разработок (например, в рамках исследований в биоинформатике или лингвистике). А вот промышленное программирование, как правило, происходит в команде (уж больно большие получаются проекты, чтобы делать их в одиночку), и совершенно точно – для заказчика, который платит за работу деньги. При этом необходимо точно понимать, что нужно заказчику, выполнить работу в определенные сроки и результат должен быть нужного качества – того, которое удовлетворит заказчика и за которое он заплатит. Чтобы удовлетворить этим дополнительным требованиям, программирование ”обрастает”различными дополнительными видами деятельности.

Разработка программного кода предваряется анализом и проектированием. Трудозатраты на анализ и проектирование, а также форма представления их результатов сильно варьируются от видов проектов и предпочтений разработчиков и заказчиков. А до этого нужно ещё собрать и проанализировать требования и пожелания к продукту. Требуются также специальные усилия по организации процесса разработки. Например, сейчас популярна итеративно-инкрементальная модель, когда требуемая функциональность создается порциями, которые менеджеры и заказчик могут оценить, и тем самым есть возможность управления ходом разработки. Однако эта общая модель имеет множество модификаций и вариантов. А ведь есть ещё и другие модели.

Разработку системы также необходимо выполнять с учетом удобств ее дальнейшего сопровождения, повторного использования и интеграции с другими системами. Это значит, что система разбивается на компоненты, удобные в разработке, годные для повторного использования и интеграции. А также имеющие необходимые характеристики по быстродействию. Для этих компонент тщательно прорабатываются интерфейсы. Сама же система документируется на многих уровнях, создаются правила оформления программного кода, то есть проводится дополнительная работа, помогающая создать и поддерживать единую, стройную архитектуру, единообразный стиль, порядок в коде.

Нужно выбрать, какие технологии мы будем использовать. Нужно спланировать, как будет организован процесс разработки: какие нам нужны люди, в какой момент подключать их к работе, какого уровня специалисты нам нужны, нужно ли покупать какой-то софт, нужна ли аренда офиса, какие в проекте будут команды, как они будут организованы, как

будет строиться их взаимодействие, как оценивать успешность хода работ, что делать, если возникли непредвиденные сложности, как работать с поступаемыми изменениями, как контролировать качество продукта, как наладить его выпуск и развёртывание, как будет организовано сопровождение и многое-многое другое.

Все эти и другие дополнительные виды деятельности, выполняемые в процессе промышленного программирования и необходимые для успешного выполнения заказов и будем называть программной инженерией (software engineering). Получается, что так мы обозначаем, во-первых, некоторую практическую деятельность, а во-вторых, специальную область знания. Или другими словами, научную дисциплину. Ведь для облегчения выполнения каждого отдельного проекта, для возможности использовать разнообразный положительный опыт, достигнутый другими командами и разработчиками, этот самый опыт подвергается осмыслению, обобщению и надлежащему оформлению. Так появляются различные методы и практики (best practices) – тестирования, проектирования, работы над требованиями, архитектурных шаблонов и пр. А также стандарты и методологии, касающиеся всего процесса в целом. Вот эти-то обобщения и входят в программную инженерию как в область знания.

Итого, в сферу программной инженерии попадают все вопросы и темы, связанные с организацией и улучшением процесса разработки ПО, управлением коллективом разработчиков, разработкой и внедрением программных средств поддержки жизненного цикла разработки ПО.

Необходимость в программной инженерии как в специальной области знаний была осознана мировым сообществом в конце 60-х годов прошлого века. Рождением программной инженерии является 1968 год – конференция NATO Software Engineering, г. Гармиш (ФРГ), которая целиком была посвящена рассмотрению этих вопросов, и которая была спровоцирована поиском решений для происходившего в то время «кризиса программного обеспечения». Причины кризиса программного обеспечения были связаны с общей сложностью аппаратного обеспечения и сложностью разработки программного обеспечения. Кризис проявляет себя самым различным образом: стоимость проектов превышает бюджет, в проектах превышаются сроки выполнения, программное обеспечение было слишком неэффективным, программное обеспечение имело слишком низкое качество, программное обеспечение зачастую не отвечало необходимым требованиям, проекты были неуправляемыми, и возникали трудности с поддержкой кода, программное обеспечение было непригодным для распространения. Колоссальные успехи в области развития средств вычислительной техники пришли в противоречие с низкой производительностью труда программистов и низкими темпами ее роста. В связи с усложнением бизнеса, усложнением программных систем стало очевидным, что их трудно проектировать, кодировать, тестировать и особенно трудно понимать, когда возникает необходимость их модификации в процессе сопровождения. Появилась жизненная потребность в создании технологии разработки программных средств и инженерных методов их проектирования для существенного улучшения производительности труда разработчиков. Впрочем, как показывает таблица ниже, люди до сих пор не имеют проверенного и надёжного способа создавать качественное ПО в срок и в рамках зафиксированного бюджета. И чем сложнее проект, тем сложнее довести его до конца.

Почему же до сих пор так всё плохо в индустрии разработки ПО? Одна из самых главных причин — сложность ПО. Сложность программных объектов зависит от их размеров в гораздо большей степени, чем для любых других создаваемых человеком объектов. Раз-

рабатывать сложное ПО невыразимо трудно. В сложных системах огромное количество компонент, которые могут сочетаться друг с другом неким нелинейным способом, генерируя ещё большее количество сочетаний, и сложность целого растет значительно быстрее, чем линейно. К тому же масштабирование программного объекта — это не просто увеличение в размере или количестве каких-то типовых элементов, это обязательно увеличение числа различных элементов. В итоге программные системы очень сложно разрабатывать, понимать, описывать, тестировать. Сложность программ является существенным, а не второстепенным свойством. Поэтому описания программных объектов, абстрагирующиеся от их сложности, часто абстрагируются от их сущности. Математика и физические науки за три столетия достигли больших успехов, создавая упрощенные модели сложных физических явлений, получая из этих моделей свойства и проверяя их опытным путем. Это удавалось благодаря тому, что сложности, игнорировавшиеся в моделях, не были существенными свойствами явлений. И это не действует, когда сложности являются сущностью. Многие классические трудности разработки программного обеспечения проистекают из этой сложности сущности и ее нелинейного роста при увеличении размера. Сложность служит причиной трудности процесса общения между участниками команды разработчиков, что ведет к ошибкам в продукте, превышению стоимости разработки, затягиванию выполнения графиков работ. Сложность служит причиной трудности перечисления, а тем более понимания, всех возможных состояний программы, а отсюда возникает ее ненадежность. Сложность функций служит причиной трудностей при их вызове, из-за чего программами трудно пользоваться. Сложность структуры служит причиной трудностей при развитии программ и добавлении новых функций так, чтобы не возникали побочные эффекты. Сложность структуры служит источником невизуализуемых состояний, в которых нарушается система защиты. Сложность служит причиной не только технических, но и административных проблем. Из-за сложности трудно осуществлять контроль, в результате страдает концептуальная целостность. Трудно найти и держать в голове все “движущиеся части” системы. Вторая причина — недостаток накопленного у человечество опыта. Программная инженерия — весьма молодая область знаний, и практически на наших глазах происходит формирование многих её частей. Возможно через несколько сотен лет создание ПО станет настолько же понятным и отработанным процессом, как строительство зданий, но сейчас эта деятельность всё ещё больше похожа на творчество, чем на ремесло, и довольно плохо прогнозируется. В результате предсказать результат часто бывает крайне непросто. Третий момент состоит в том, что программные объекты постоянно подвержены изменениям, в том числе (в отличие от большинства объектов физического мира) и уже после их изготовления. Программы часто меняются, потому что 1) это становится необходимо (исправляются ошибки или меняется понимание их предназначения или окружающей инфраструктуры) и 2) это в принципе возможно за невысокую цену (тут как повезёт, на самом деле, часто заказчику бывает тяжело объяснить, что для реализации такой-то фичи придётся половину кода переписывать). Программы — это продукт мысли, воплощённый в программном коде. Поменяв несколько символов текста и осуществив процедуру сборки (и, возможно, размещения) программы, мы можем получить абсолютно другое поведение, чем у этой программы было раньше. И всё это чаще всего в полностью автоматическом режиме и практически без каких-либо затрат. Со многими физическими объектами такого в принципе осуществить невозможно, а если и возможно, это может быть очень долго и дорого. Да, здания тоже перестраиваются, но если бы можно было бы поменять внешний вид или добавить своему дому пару лишних этажей простым нажатием клавиш, вид наших

улиц бы сильно преобразился. Ещё один момент, который стоит отметить в разработке ПО — сильная социальная составляющая проектов. Программные продукты разрабатываются людьми (разработчиками) для людей (заказчиков). Должно быть много социального взаимодействия, как внутри команды, так и с людьми снаружи. В состав команды входят разные люди, которые выполняют разные работы, и данный курс будет частично направлен на то, чтобы показать, чем же ещё нужно заниматься, чтобы уверенно вести программные проекты к завершению.

Успех программного проекта зачастую определяется социальными факторами, а роль технологии второстепенна. Есть немало примеров технически отсталых систем, которые работают и полезны заказчикам. Обратное, как правило, неверно. От системы, не приносящей пользы (ожидаемой или реальной), заказчик рано или поздно откажется независимо от того, насколько блестящей она является в техническом отношении. В связи со всем этим на рынке труда востребованы программисты, умеющие работать в команде. Проводя интервью, кандидатов оценивают не только на наличие тех или иных знаний и навыков (разумеется, программист должен уметь писать хороший код), но оценивается также, насколько хорошо данный человек вольётся в существующих коллектив, сможет ли он строить эффективное взаимодействие с остальными участниками проекта. Для многих компаний это даже важнее: при должном уровне человека можно довольно быстро научить пользоваться практически любыми языками и технологиями, а вот психологическую несовместимость исправить практически невозможно. Имеющиеся промышленные стандарты для профессии “Программист” (более старый и более новый) подтверждают эти мысли:

Возрастает значение профессиональных компетенций коллективной разработки программного обеспечения, знание современных направлений, методов и технологий разработки программного обеспечения: понимание обязанностей различных участников команды по разработке программного обеспечения: руководитель разработки программного обеспечения, руководитель технической группы (team leader), архитектор, программист, тестировщик, дизайнер, верстальщик, аналитик; владение современными стратегиями и технологиями организации коллективной разработки программного обеспечения, включая системы управления версиями, процессы непрерывной интеграции, стандарты оформления кода и методы инспекции кода; понимание основных направлений развития методов коллективной разработки, их отличий и целесообразности применения в зависимости от типа решаемых задач и требований организации; владение гибкими (Agile) методологиями разработки программных продуктов.

Стандарт 2009 года описывает четыре квалификационных уровня профессиональных программистов. Программист первого уровня “создает код модулей или тестовых наборов для модулей системы или небольших приложений низкого уровня сложности по готовым спецификациям под руководством специалиста более высокого квалификационного уровня”. Обычно это называется “стажер” или “младший программист”. Программист второго уровня “самостоятельно создает спецификации, код модулей или тестовых наборов для компонент и подсистем; интегрирует модули в подсистемы, обеспечивая согласованное функционирование и требуемый уровень качества; руководит работой младших программистов; ответственность в пределах своего рабочего задания”. Это “программист”, “инженер-программист”, “разработчик” или что-то аналогичное. Программист третьего уровня “разрабатывает и согласует спецификации, код и тесты на уровне системы; тестирует и оптимизирует код приложений на системном уровне; руководит группой разработчиков; возложена определенная ответственность, имеет некоторую автономность в приня-

тии решений”. Это “старший разработчик”, “старший программист” и т.п. Программист четвёртого уровня в свою очередь “руководит разработкой сложного программного проекта; разрабатывает и адаптирует технологию и процесс разработки; планирует материальные ресурсы необходимые для выполнения проекта; подбирает состав, планирует задачи и руководит проектной группой; управляет качеством программного продукта; возложена определенная ответственность, автономность в принятии решений”. Это “старший инженер”, “ведущий программист” или что-то такое. Более подробно об этом всём, включая список компетенций для каждого уровня, можно прочитать собственно в самих стандартах. А ещё есть Guide to the Software Engineering Body of Knowledge (SWEBOK Guide), который по сути кратко описывает весь накопленный в программной инженерии свод знаний. Крайне рекомендуется к ознакомлению.