О квалификации

О квалификации IT- специалистов

Есть разные мнения на счет классификации уровней квалификации. Наиболее объективно, пожалуй, нижеописываемое мнение. Детальные требования к уровням квалификации безусловно зависят факторов, специфичных конкретному рабочему месту; поэтому, требования к специалистам той или иной квалификации будут изложены обобщенно.

Trayny

строго говоря, это уровень квалификации не профессионала, а стажера;
квалификация — заведомо не профессионал;
владение информационными технологиями — заведомо на допрофессиональном уровне;
опыт професиональной деятельности — заведомо отсутствует (вся его деятельность, какой бы она ни была, заведомо на допрофесс. Уровне);
должность — это даже не должность, а прохождение стажировки;

Junior developer

это уровень квалификации профессионала;
квалификация — Junior (начинающий профессионал);
владение информационными технологиями — заведомо на уровне профессионала;
опыт профессиональной деятельности — либо отсутствует, либо не значительный;
должность — developer;

Middle developer

это уровень квалификации профессионала;
квалификация — Middle (собственно профессионал);

владение информационными технологиями — заведомо на уровне профессионала;
опыт профессиональной деятельности — имеется, и около 1 г.;
должность — developer;

Senior developer

это уровень квалификации профессионала;
квалификация — Senior (высококвалифицированный профессионал);
владение информационными технологиями — заведомо на уровне профессионала;
опыт профессиональной деятельности — имеется, и значительный (в IT это около 3 г.);
Team lead иногда делегирует ему (обычно технические) обязанности;
должность — developer;

Team Lead

это квалификация профессионала (и руководителя — часть его обязанностей это управляющее воздействие);
квалификация — Team Lead (высококвалифицированный профессионал, и руководитель команды разработчиков);
владение информационными технологиями — заведомо на уровне профессионала (и навыки управления коллективом — заведомо на уровне профессионала (по крайней мере, для руководителя данного уровня));
опыт профессиональной деятельности — имеется, и значительный (в IT это около 3 г.);
Team Lead иногда делегирует Senior developer’у (обычно технические) обязанности;
владение информационными технологиями (очень часто, просто закономерно) оказывается даже выше, чем у Senior developer’а;
исполняет как технические, так и руководящие обязанности (такой спектр обязанностей закреплен за ним официально, и напрактике действительно необходим в его работе);
должность — Team Lead;

это квалификации профессионалов (технических и гуманитарных специалистов соответственно);
квалификация — Software Architect/ Product Manager (об их квалификации имеет смысл говорить только всвязи с их должностью (эти должности подразумевают квалификацию в разных областях));
владение информационными технологиями — в рамках своей квалификации (об их владении информационными технологиями имеет смысл говорить только всвязи с их квалификацией (соответственно, и должностью));
опыт профессиональной деятельности — имеется, и значительный (в IT это около 3 г.), а зачастую очень значительный (3 г. * N);
должность — Software Architect/ Product Manager (штабной или руководящий специалист с техническими обязанностями/ руководящий специалист);

Все hard- и soft- skills, какие бы то ни было, зависят от квалификации. Владение skills на том или ином уровне должно быть ясно из здравого смысла (а конкретный набор обязанностей, как и говорилось выше, зависит от конкретного рабочего места). При этом важно помнить — одни знания могут приобретаться, другие же терять актуальность — всвязи с ростом опыта работы (действительно, роль профессионала в компании зачастую меняется со временем). При всей этой динамичности знаний профессионала само понятие квалификации ни коим образом не является выхолощенным. Это потому, что квалификация неразрывно связана с полезностью профессионала. Если же оценка полезности профессионала представляется чем- то неясным, то следует учесть следующее. Такой критерий полезности, как производительность труда, безусловно зависит от квалификации. Производительность труда безусловно характеризует профессионала.
Об уровне квалификации следует помнить и следующее. С ростом квалификации профессионала растет и возлагаемая на него ответственность. Также необратимо возрастает и косность мышления. Эти факты заставляют обратить внимание на следующее. Более ответственный сотрудник более ценен для компании. При этом, профессионально развивающийся сотрудник ценнее инертного. Однако, высококвалифицированный сотрудник всегда ценнее, так как его производительность всегда выше (а поиск такого сотрудника всегда

дольше). При этом, необходимо помнить — случай, когда сотрудник overqualified для своего рабочего места, является исключением из этого правила. Overqualified (для своего рабочего места) сотрудник (зачастую, просто закономерно) менее ценен, чем nonqualified (разумеется, если речь не идет о вопиющей некомпетентности). Разумеется, есть различные доводы — pro и contra данного мнения. Однако, при всех этих доводах, нельзя забыть о следующем доводе — overqualified содрудник расценивается как «qualified soldier, half-qualified sergeant», именно так оцениваются его профессиональные и личные качества.

О производительности

О производительности труда IT- специалистов

Профессиональная деятельность IT- специалистов — это работа над проектами (производимого ими продукта), по крайней мере, с технической стороны их профессиональной деятельности. Успешность профессиональной деятельности IT- специалистов зависит от завершения проекта в срок (и в надлежащем качестве).

Срыв срока — это, собственно, факт срыва формально заданного срока. Наиболее формально заданный срок наиболее часто оказывается сорван. Действительно, выполнение проекта осуществляется человеком, а не техникой, и поэтому формальное определение срока крайне затруднительно.

Проект (не важно, сколь нагруженный) нельзя завершить быстрее, чем за ¾ срока. Попытки завершить проект быстрее (зачастую, просто закономерно) приводят к фиаско. Речь идет о ускорении завершения проекта сугубо усилиями сотрудников. Ускорение уже начатого проекта прочими методами (например, включением в команду новых сотрудников, внедрением новых технологий) крайне проблематично, и зачастую приводит к срыву. Действительно, прочие методы применимы до начала проекта, и никак не после. Если же они применены до начала проекта, то это может существенно снизить срок его завершения. Применение перспективных технологий может ускорить разработку на 2 … 10 %, при этом бюджет проекта изменяется неоднозначно; применение высококвалифицированного труда может ускорить разработку в 3 … 10 раз (и эта разница — только между одной ступенью квалификации), при этом бюджет проекта увеличивается до 2 раз. Привлечение дополнительных сотрудников не имеет однозначного влияния на срок проекта; это может и вовсе понизить производительность команды. Так же следует иметь ввиду, что отдача от каждого дополнительного сотрудника снижается (так как с каждым новым сотрудником повышается сложность координирования действий в команде), но бюджет проекта возрастает на величину его зарплаты.

Производительность и качество труда сотрудников зависит от координирования действий в команде. Выполнение проекта в срок и в надлежащем качестве зависит от производительности и качества труда сотрудников. Производительность и качество труда сотрудников зависят от слаженности действий в команде гораздо больше, чем от формальных (бюрократических) мер, направленных на регулирование процесса разработки. Производительность зависит так же от профессионального стимулирования сотрудников (материального и нематериального). Пренебрежение этими направлениями профессионального стимулирования препятствут достижению возможной производительности труда.

Производительность и качество труда в отрасли IT гораздо важнее, чем в ряде других отраслей. Действительно, от производительности и качества труда зависит выполнение проектов. Лица, являющиеся разработчиками в отрасли IT, получают свои доходы за счет выполнения проектов (по крайней мере, если говорить о доходах в реальном секторе экономики). Для этих лиц срыв проектов равносилен срыву их доходов.

Пренебрежение производительностью и качеством труда по своей сути является пренебрежением профессионализмом (что, в свою очередь, является вопиющей некомпетентностью).

Что касается метрик производительности труда IT- специалистов, то ее целесообразно измерять в способности выполнить проект в срок и в надлежащем качестве. Эта метрика — логическое значение (либо эта способность есть, либо ее нет). Прочие метрики (часы, строки кода, etc.) не говорят о возможности выполнения проекта (это потому, что целью разработки является выполнение проекта, а не затраченные часы и написанные строки кода, как и затраченные бюджеты).

О разработке ПО

О процессе разработки ПО

Процесс разработки ПО — это, собственно, процесс разработки ПО. Построение процесса разработки ПО — безусловно, прерогатива того, кто этот процесс и осуществляет. Однако, было бы ошибочно считать, что изучение проблематики (и наработанных решений) процесса разработки ПО представляет лишь академический интерес — действительно, в ходе исторического процесса подход к разработке ПО развивался. Отрицать полезность доводов, достигнутых путем долгой эволюции, было бы не рационально.
Всевозможные процессы можно классифицировать по критерию соответствия типовым моделям процесса разработки. Различные типовые модели (аппроксимативно) описывают процесс разработки. Как и говорилось выше, эти модели появились в процессе эволюции подхода к разработки ПО (и описываются в порядке их появления).
Разумеется, речь идет о типовых процессах разработки ПО — реальный процесс вряд ли будет в точности соответствовать типовому.
Типовые модели процесса разработки ПО описаны ниже.

Водопадная модель

Это первая типовая модель процесса разработки ПО.
Эта модель вменяет процессу следующее: процесс разработки состоит из фиксированного набора этапов, этапы процесса разработки выполняются в фиксированном порядке (один этап строго следует за другим — этой особенности модель и обязана своим названием). Этой модели присущи также следующие особенности: время выполнения проекта регламентируется наиболее формально, как и техническое задание (которое фиксируется на все время проекта, и ставится наиболее формально).
Эта модель наиболее строго регламентирует процесс разработки, и поэтому поэтому представляет лишь академические интерес (даже по мнению, приводимому в любом учебном пособии). Действительно, даже на ранних этапах развития IT не приходилось говорить о более-менее формальном соответствии какого- либо процесса этой модели.

Секвенция этапов процесса разработки (и уровень их детализации) должна быть описана описана по той причине, что без этих сведений описание процессов разработки ПО будет излишне пространным. Эта модель подразуемевает секвенцию следующих (сменяющих друг друга в указанном порядке) этапов:
постановка требований к продукту,
проектирование продукта,
реализация проекта продукта,
тестирование продукта,
поставка продукта (и сопутствующие операции),
поддержка продукта (в рамках соглашений);

конечно, различные тематические источники (описывающие данный предмет с различной степенью детализации) могут приводить различные списки этапов водопадного процесса, но целесообразно остановиться на списке приведенном выше.

Итеративная модель

Эта модель вменяет процессу следующее: процесс разработки состоит из набора этапов, полученных разбиением этапов классической (водопадной) модели. При этом, результирующая секвенция рассматривается как линейная (не иерархическая) секвенция. Такое разбиение на этапы преследует следующую цель — на каждом этапе решать ограниченную (и более просто разрешимую) задачу. Конкретный способ разбиения на этапы остается на откуп тому, кто осуществяет процесс разработки. Эта модель разработки, по сути, представляет собой водопадную модель, несколько упрощенную за счет того, что на каждом этапе разрешается более мелкая задача.
Этой модели присущи также следующие особенности: по сути, эта модель реализует особенности классического процесса разработки; однако, ей легче следовать за счет того, что на каждом этапе решается более простая задача.
Эта модель разработки имеет специфическую вариацию — инкрементальная модель. Особенность этой вариации в том, что процесс разработки предстает в виде (длинной) секвенции элементарных задач. Предполагается, что если на каждом этапе будет решаться элементарная задача (такую задачу трудно считать не разрешимой), то процесс разработки будет гарантированно успешным.

Эта разновидность данной модели также называется «эволюционной» - действительно, эволюционные процессы представляют собой ряд операций инкремента.

Спиральная модель

Эта модель вменяет процессу разработки следующее: этапы этого процесса повторяются во времени до их логического завершения. Иначе говоря, этот процесс есть очередь этапов, завернутая в кольцо. На схеме же он изображается как спираль — отсюда и название. После каждого этапа этого процесса принимается решение о деталях следующего этапа. Поэтому, этот процесс представляет собой, по сути, эволюционный процесс, в котором допускается доработка ранее выполненных этапов. Именно эволюционный — иначе этот процесс (и так сложный) обречен на фиаско (в силу сложности каждого этапа).
Этому процессу присуща также следующая особенность: он наиболее формально определяет саму очередь этапов, но наименее формально — сколько раз она будет завернута в ходе исполнения в кольцо. Сама очередь этапов определяется наиболее формально в силу эволюционной природы этого процесса. Количество прохождений этой очереди по кольцу определяется наименее формально — по той простой причине, что столь сложный эволюционный процесс не может быть расписан во всех деталях до самого конца. Если же такой процесс может быть расписан во всех деталях до самого конца — то он (несомненно) может быть заменен более простым процессом.

Специфические модели

Множество моделей разработки включает в себя также некоторые специфические модели. Здесь и далее под специфическими моделями имеются ввиду такие модели, которые кардинально отличаются от вышеописанных. Специфические модели можно обнаружить даже среди наиболее давно известных моделей. Все специфические модели кардинально отличаются от тех моделей, что находятся на прямом пути их исторического развития (этот путь описан выше). Поэтому, все специфические модели представляют ограниченный интерес. Среди старых моделей специфические модели не будут рассмотрены. На этом обзор старых моделей завершен. Далее будут рассмотрены

современные модели процесса разработки ПО. Если среди них будут специфические, это будет оговорено явно.

Современные модели

Современные модели представляют собой видоизмененные старые модели (с поправкой на различия в методологии).
Методология разработки ПО во времена старых моделей подразумевала наиболее формальное следование предписаниям моделей. Методология разработки ПО в настоящее время подразумевает необязательно формальное следование предписаниям моделей, но обязательно формальное следование самой цели разработки ПО (поставка продукта в срок и в надлежащем качестве). В связи с этим, старую методологию принято называть строгой, а новую — гибкой.
В настоящее время методологией могут называть не методологию как таковую, а следование той- или иной концепции в рамках методологии. Безусловно, это не вполне правильно, но это уже вошло в традицию. Сама современная методология способствует тому, чтобы подход к разработке был ориентирован не на различные доводы (имеющие прежде всего академическое значение), а на сам процесс разработки (с его конечной целью).
Что касается конкретных концепций, практикуемых в рамках современной методологии, то они могут быть интересны лишь с поправкой на их реализацию в конкретном процессе разработки ПО. Сама современная методология вменяет такой подход. Действительно, при гибкой методологии разработки какие- либо концепции, будучи оторванными от конкретного процесса разработки ПО, выхолащиваются до полной потери своего содержания.

О конкретных концепциях

Как и говорилось выше, все эти концепции, в отрыве от конкретного процесса, могут подразумевать очень многое.
Конкретные концепции, о которых приходится говорить в современности — это те концепции, что изложены в Agile manifesto. Agile Manifesto был принят в 2001.
Концепции, изложенные в пресловутом манифесте, преподносятся именно как концепции (и не вменяют конкретных практицизмов).

Это следующие концепции:

1. процесс разработки должен быть ориентирован на результат процесса, а не на ход процесса (всегда стоит помнить о цели);
2. возможность достижения цели процесса разработки ПО — единственная объективная метрика его эффективности;
3. эффективность процесса безусловно должна интересовать разработчика ПО;
4. эффективность процесса безусловно должна повышаться;
5. в качестве меры совершенствования процесса следует использовать в том числе и упрощение (в силу его эфф- ти);
6. требования к продукту могут быть изменены даже на поздних этапах его реализации (иначе это же могут позволить себе конкуренты);
7. в процессе разработки ПО между разработчиком и потребителем должна быть и использоваться связь (это способствует разработке конкурентноспособного продукта);
8. в качестве способа связи стоит использовать в том числе и непосредственное общение (в силу эффективности этого способа общения);
9. продукт следует поставлять как можно чаще — с периодом от двух недель до двух месяцев (форсированное развитие продукта способствует его конкурентноспособности);
10. процесс разработки ПО направлен на удовлетворение соответствующих интересов следующих групп лиц — инвесторов, разработчиков, и потребителей; в интересах всех этих групп лиц процесс разработки должен быть поставлен на поток (принято считать, что это в интересах всех этих лиц);
11. в процессе разработки ПО должны задействоваться профессионалы, и они должны быть мотивированы (только тогда качество оптимально);
12. наиболее профессиональными являются те группы разработчиков, что способны к самоорганизации;

сам манифест приводит эти концепции в другой последовательности (они приводятся в нем не отсортированными по характеризуемому ими аспекту прцесса разработки, а в разнобой), и в других словах (не в официальном стиле речи, а сродни «Эзоповому языку»). Кроме того, в манифесте приводится именно четыре ценности и двенадцать концепций — это наводит на мысль, что каждая ценность описывается

в трех концепциях. Однако, сам этот прием основан на эмоциональном восприятии, а не на рациональных доводах. Это изобличает подход к манифесту (он мог бы быть серьезней — под стать документу, а не развлекательному ресурсу). Подход к документу изобличает так же исполнение его источника (сайт не развивается), и его перевод (выполнен лишь частично). Несмотря на несерьезный подход, этот документ выражает более чем серьезный смысл.
Смысловая нагрузка манифеста может быть выражена в следующем формальном высказывании — в интересах отрасли разработки ПО безусловно следующее: форсирование развития технологии разработки, форсирование индустриализации этой отрасли, и форсирование подготовки доподлинно профессиональных разработчиков; все эти факторы действуют взаимосвязанно, и поэтому изменения в них приводят к изменениям (динамического характера) в отрасли разработки ПО; при условии, что в ходе изменений достигается именно развитие, следование доводам манифеста к развитию и приведет; форсирование развития отрасли разработки ПО представляется наилучшим способом удовлетворения интересов как инвесторов, так и разработчиков, и потребителей.
Доводы этого манифеста безусловно используются (по сути, он призывает к повышению уровня индустриализации, что сулит известный уровнь доходов).

Специфическая современная модель SCRUM

Эта специфическая модель официально считается одной из моделей гибкой разработки. Однако, здесь ключевое слово - «официально». Эта модель вменяет процессу следующее: он формально разбивается на ряд этапов (этап длится от одной до четырех недель), каждый этап подразделяется на задачи (каждая задача занимает строго один день), каждая задача подразделяется на составляющие, и выполнение этих составляющих подвергается мониторингу (мониторинг осуществляется во временных рамках задачи, и происходит на предмет того что планировалось, того что получилось в результате, и каковы расхождения с планом). Причем, мониторинг производится в такой метрике, как часы. Каждый этап процесса предваряется совещанием (обсуждается повестка этапа), и постановка каждой задачи предваряется совещанием (обсуждается повестка дня), и кроме того, каждый этап завершается совещанием (обсуждается результат этапа).

В этих совещаниях участвуют все специалисты, участвующиев процессе разработки ПО (начиная от такого технического специалиста, как архитектор ПО, и заканчивая таким гуманитарным специалистом, как SCRUM- мастер). Кроме того, проводятся совещания в целях взаимодействия с заказчиком (либо его представителем), такие совещания проводятся обычно раз за период этапа.

Эта специфическая модель фактически расходится с доводами гибкой методологии в следующем: имеются строгие формальные предписания процессу разработки; эти предписания строго ограничивают не только набор составляющих процесса разработки, но и время составляющих процесса разработки; организационные мероприятия проводятся каждый день, при этом мероприятия по анализу проделанной работы проводятся только раз за период этапа (и процесс получает корректирующее воздействие не чаще); мероприятия по взаимодействию с заказчиком производятся только раз за период этапа (что более чем ограничивает взаимодействие с заказчиком); период этапа может быть месяц (а продукт рекомендуется (по Agile-манифесту) поставлять с периодом две недели — два месяца), успешность хода процесса привязана к часам (а не к поставке продукта в срок и в надлежащем качестве).

Вышеописанное сполна характеризует эту методологию.

О парадигмах

Парадигмы программирования

Парадигма — общепринятое утверждение. Парадигма программирования — устоявшаяся (в то или иное время) концепция, в рамках которой происходит программирование. Речь идет о концепциях, значимых для самого процесса программирования.
Парадигмы программирования, по ходу исторического процесса, сменяют друг друга. И языки программирования наделяются средствами реализации новых парадигм программирования.
Далее будут рассмотрены парадигмы программирования в порядке прихода к ним.

ПП

ПП — процедурное программирование. Это простейшая концепция организации программ. Она предполагает, что регулярные последовательности операций организуются в модули — подпрограммы, и эти модули затем используются как готовые решения.

СП

СП — структурное программирование. Это одна из простейших концепций организации программ. Она предполагает, что программа безусловно включает в себя структуры управления ходом вычислений; и, эти структуры управления ходом вычислений могут быть классифицированы как типовые, если в программе не осуществляется безусловный переход напроизвольный участок кода; эта концепция предписывает организовать программы как именно такие структуры управления ходом вычислений (иными словами, позволяет избежать макаронного кода).

ООП

ООП — объектно- ориентированное программирование.
Как следует из названия, во главу угла ставятся объекты, в свете применения которых и происходит программирование. Для этого ЯПВУ наделяется (характерными ему) средствами поддержки объектов. Объекты образуют те или иные иерархии, трактуются языком с теми или иными различиями, инкапсуллируют в себе те или иные данные и алгоритмы работы с характерными им данными. Таким образом, объекты представляют собой способ организации программы по модулям — объектам.
ООП возникло как средство организации больших программ (когда уже существовавшие парадигмы программирования не позволяли конструировать достаточно большие программы), и его основная задача — это возможность организации больших программ.
ООП опрерирует такими понятиями как наследование, инкапсулляция, и полиморфизм. Все эти понятия доподлинно общеупотребительные, и не требуют комментариев. Комментариев требует лишь принятый относительно них подход — при использовании концепции наследования особый упор делается на то, чтобы в иерархии объектов наследовались лишь действительно необходимые особенности, а осбенности специфические появлялись (и были) только где это необходимо; при использовании концепции инкапсулляции особый упор делается на то, чтобы скрыть инкапсуллированные данные, сделав их недоступными подсистемам, не предназначенным для работы с ними; при использовании концеции полиморфизма особый упор делается на то, чтобы единый алгоритм мог обрабатывать различные данные, причем, если надо — то специфическими для них способами.

ФП

ФП — функциональное программирование. В рамках этой концепции программа представляет собой секвенцию вызовов функций (под функциями понимается прежде всего просто наборы логик, которыми располагает программа); функции в результате могут возвращать те или иные функции, которые в последствии вызываются на целевом участке программы; и, как следствие, функции могут принимать в аргументах эти самые функции, и вызывают их по своей логике.

Однако, эту парадигму характеризует прежде всего следующее: математики могут использовать формальные математические языки, говоря о ФП в своем понимании; и программисты могут использовать языки программирования, имеющие формальные правила, реализуя данную парадигму на практике; однако, эта прадигма, как парадигма представляет собой эвфемерную концепцию, не сводящуюся к формальным тезисам. Действительно, приведенное описание не является секвенции тезисов, состоявшихся как формальные, не эвфемерные.

Примечание

Два пресловутых подхода к программированию — императивное и декларативное программирование, зачастую приравниваются к парадигмам программирования. Однако, это не целесообразно. Под парадигмой программирования принято понимать более конкретные концепции. Поэтому, столь обобщающие концепции, как императивное/ декларативное программирование, в настоящее время не целесообразно называть парадигмой программирования. Чтобы определиться с подходящим термином, эти концепции стоит называть подходами к программированию.
Об этих концепциях также следует помнить следующее — их зачастую выражают в подробнейших описаниях, за которыми легко скрывается сама их суть. По этому, целесообразно остановиться на следующих лаконичных определениях: императивный подход — это подход, в рамках которого программист формально прописывает все вычисления, составляющие вычислительный процесс; декларативный подход — это подход, в рамках которого программист лишь предписывает цель вычислительного процесса, оставляя сам вычислительный процесс на откуп средству, которому предназначен составленный программистом код. Строго говоря, декларативный подход к программированию трудно считать подходом к именно программированию — по той простой причине, что в рамках этого подхода программирования вычислений в принципе не происходит.

О профессионализме

Понятие профессионализма

О понятии профессионализма часто ведутся споры. Например, следует ли считать профессионализм и квалификацию равнозначными понятиями (либо первое понятие стоит считать более нагруженным смыслом).
Поэтому, чтобы избежать этих споров, целесообразно прийти к следующей трактовке этого понятия.
Профессионализм лиц, ведущих профессиональную деятельность в области IT, следует трактовать как множество следующих составляющих:
обладание полнотой (то есть, должным объемом и качеством) профессиональных знаний;
факт применения профессиональных знаний на практике — непременно должное (во всех смыслах этого слова) применение профессиональных знаний на практике;
отношение к своей деятельности как к основной (либо одной из основных) профессиональной деятельности;

это критерии профессионализма, применимые ко всем лицам — как частным, так и корпоративным.
Итак, было рассмотрено понятие профессионализма. Объемлемое им понятие — квалификация, было рассмотрено выше. Для большей полноты представлений об этих вопросах, целесообразно рассмотреть и объемлющее понятие — экспертность.
Эксперт — это профессионал (в своей области), привлекаемый третьими лицами для выдачи заключения по поставленным перед ним вопросам (в рамках соответствующей области). Таким образом, это профессионал, мнение которого считается наиболее объективным (в сравнении с мнением прочих профессионалов, не привлекаемых в качестве экспертов).
Экспертность заключается в обладании профессионализмом (на уровне, соответствующем привлечению лица в качестве эксперта), и фигурировании в качестве эксперта (наличие соответствующей роли в команде разработчиков, участие в коллегиальных органах, etc).

О паттернах

Паттерны проектирования

Суть понятия

Паттерн проектирования (применительно к ПО) — это устоявшееся решение в архитектуре ПО.
При применении различных парадигм программирования применяются различные решения в архитектуре ПО. Разумеется, программы (созданные при применении различных парадигм программирования) могут обладать сходной функциональностью, но применяемые в них архитектурные решения будут разными.
При использовании таких решений (паттернов проектирования) архитектура ПО приобретает вид, при котором приложение представляет из себя систему устоявшихся решений. Разумеется, необдуманное применение таких решений (например, с самоцелью унификации применяемых решений) приводит к отказу от более уместных решений.
Паттерны проектирования ПО принято рассматривать на всех уровнях архитектуры ПО. Паттерны самого высокого уровня в архитектуре ПО принято называть паттернами архитектуры. Паттерны более низкого уровня — собственно паттерны. Паттерны наиболее низкого уровня — идиомы (на данном уровне их целесообразно (и принято) рассматривать без абстракции от реализующего их программного кода; действительно, низкоуровневые составляющие ПО решают глубоко конкретизированные задачи — таким образом, абстрагироваться от деталей этой задачи — это абстрагироваться от самой ее сути).

Историческое

Исследования прменения устоявшихся решений в архитектуре ПО велись более чем давно.
В 1991 вышла книга Coplien “Advanced C++ Idioms”. С тех пор исследования в данной области велись особенно интенсивно (особенно, известной группой лиц — Gang of Four).

В 1994 вышла книга Gamma, Helm, Johnson, Vlissides “Design Patterns – Elements of Reusable Object- Oriented Software” (пресловутая «Book by Gang of Four»). С тех пор, тема паттернов проектирования стала «широко известной в узких кругах».

Классификация паттернов

Как и говорилось выше, паттерны можно классифицировать по уровню архитектуры ПО, на котором они применяются. Также их можно классифицировать по назначению.
Паттерны проектирования подразделяются по назначению на следующие:
порождающие паттерны — отвечают за создание составляющих ПО;
структурные паттерны — отвечают за структуру связей между составляющими ПО;
поведенческие паттерны — отвечают за реализацию поведения составляющих ПО;

Порождающие паттерны

Singleton

Это порждающий паттерн, реализация которого приводит к тому, что у класса создан лишь один экземпляр, и он доступен из глобальной области видимости.
Этот паттерн целесообразно реализовать, когда требуется доступ к общим для ряда составляющих ПО данным. Иначе — не следует.

Prototype

Это порождающий паттерн, реализация которого позволяет клонировать объекты.
Этот паттерн целесообразно реализовать, когда требуется клонировать объекты методами клонируемых объектов (это — наиболее удачный спооб клонирования, так как он позволяет абстрагироваться от устройства объекта). Иначе — не следует.

Builder

Это порождающй паттерн, реализация которого позволяет пошагово создавть объекты.
Этот паттерн целесообразно реализовать, когда требуется создавать очень сложные объекты (в целях простоты организации, конструктор пошагово вызывает соответствующие составляющие своего кода; причем, определяет эти составляющие по своей логике — анализируя переданные параметры, что позволяет передать ограниченное число параметров, и не передавать параметры, напрямую управляющие ходом вычислений в коде конструктора). Иначе — не целесообразно.

Factory Method

Это порождающий паттерн, реализация которого позволяет конструктору создавать объекты разных классов.
Этот паттерн целесообразно реализовать, когда требуется, чтобы конструктор мог создавать различные подклассы своего класса (речь идет именно о подклассах, так как создавать объекты прочих классов не характерно конструктору). Иначе — не целесообразно.

Abstract Factory

Это порождающий паттерн, реализация которого позволяет создавать объекты различных классов.
Этот паттерн целесообразно реализовать, когда требуется создавать секвенции объектов разных классов для какого- либо совместного применения. Иначе — не целесообразно.

Структурные паттерны

Proxy

Это структурный паттерн, реализация которого позволяет перехватывать вызовы какой- либо составляющей ПО.
Этот паттерн целесообразно применять, когда целесообразен перехват вызовов к какой- либо составляющей ПО. Иначе — не целесообразно.

Flyweight

Это структурный паттерн, реализация которого позволяет секвенции объектов разделять использование каких- либо данных.
Этот паттерн целесообразно применять, когда требуется, чтобы объекты разделяли использование нужных им данных (а не дублировали их в себе). Иначе — не целесообразно.

Facade

Это структурный паттерн, реализация которого позволяет абстрагироваться от реализации сложной подсистемы ПО.
Этот паттерн целесообразно применять, когда требуется, чтобы модуль (взаимодействующий с реализующим этот паттерн модулем) вызывал целый модуль, а не его подсистемы. Иначе — не целесообразно.

Decorator

Это структурный паттерн, реализация которого позволяет реализовать дополнительный функционал составляющей ПО (без изменения самой этой составляющей).
Этот паттерн целесообразно реализовать, когда потребна обертка для объекта. Иначе — не целесообразно.

Composite

Это структурный паттерн проектирования, реализация которого позволяет обращаться к подсистеме как к самой системе.
Этот паттерн целесообразно реализовать, когда требуется обращаться к объекту подкласса по ссылке на объект его суперкласса. Иначе — не целесообразно.

Bridge

Это структурный паттерн проектирования, реализация которого позволяет составляющей ПО иметь составляющие, которые могут изменяться независимо друг от друга.

Этот паттерн целесообразно реализовать, когда необходимо, чтобы опциональные составляющие ПО не конфликтовали (например, чтобы языковые и пользовательские члены объектов могли инициализироваться опциональными значениями без риска конфликта). Иначе — не целесообразно.

Adapter

это структурный паттерн проектирования, реализация которого позволяет взаимодействовать подсистемам, реализующим разные протоколы взаимодействия.
Этот паттерн целесообразно применять, когда необходима трансляция протоколов. Иначе — не целесообразно.

Поведенческие паттерны

Visitor

Это поведенческий паттерн проектирования, реализация которого позволяет привнести в составляющую ПО новую функциональность, не модифицируя саму эту составляющую.
Этот паттерн целесообразно реализовать, когда классу следует реализовать интерфейс. Иначе — не целесообразно.

Template Metod

Это поведенческий паттерн проектирования, реализация которого позволяет методу принимать параметры разных типов, и реализовать в соответствии с ними при необходимости разную логику.
Этот паттерн целесообразно реализовать, когда от метода требуется принимать разные типы параметров. Иначе — не целесообразно.

Strategy

Это поведенческий паттерн проектирования, реализация которого позволяет разместить ряд алгоритмов в ряде подклассов одного класса, и задействовать их при необходимости.

Этот паттерн целесообразно реализовать, когда класс изменяется на протяжении жизненного цикла ПО (и изменяется набор доступных ему алгоритмов). Иначе — не целесообразно.

State

Это поведенческий паттерн проектирования, реализация которого позволяет составляющей ПО реализовать различное поведение в зависимости от ее состояния.
Этот паттерн целесообразно реализовать, когда класс оснащается подклассами (с их функциональностью), которые в последствии удаляются или добавляются. Иначе — не целесообразно.

Observer

Это поведенческий паттерн проектирования, реализация которого позволяет составляющей ПО организовать подписку на целевые события в другой составляющей ПО.
Этот паттерн целесообразо применять, когда работа одного модуля ПО должна быть согласована с работой другого (и в этих целях нужно реализовать протокол обмена данными). Иначе — не целесообразно.

Memento

Это поведенческий паттерн проектирования, реализация которого позволяет сохранять состояния объектов.
Этот паттерн целесообразно применять, когда имеется потребность работать со снимками состояний объектов. Иначе — не целесообразно.

Mediator

Это поведенческий паттерн проектирования, реализация которого позволяет диспетчеризовать сообщения между моделями.
Этот паттерн целесообразно применять, когда есть необходимость диспетчеризации сообщений. Иначе — не целесообразно.

Iterator

Это поведенческий паттерн проектирования, реализация которого позволяет производить итерации по коллекциям.
Этот паттерн целесообразно применять, когда есть необходимость в итерации по коллекциям. Иначе — не целесообразно.

Command

Это поведенческий паттерн проектирования, реализация которого позволяет преобразовать сообщения к целевому представлению, вставлять их в очередь сообщений, и журналировать эти действия.
Этот паттерн целесообразно применять, когда необходим протокол взаимодействия между различными модулями. Иначе — не целесообразно.

Chain of Command

Это поведенческий паттерн проектирования, реализация которого позволяет передавать сообщения обработчикам сообщений по цепочке.
Этот паттерн целесообразно реализовать, когда целесообразна передача сообщений по иерархии обработчиков сообщений. Иначе — не целесообразно.