# Методологии разработки программного обеспечения

В данный момент нет необходимости разрабатывать методологию разработки программного обеспечения (ПО) "с нуля". Существует широкий выбор готовых методологий на все случаи жизни. И, хотя, практически каждый достаточно опытный руководитель разработки программного обеспечения со временем находит свои, более удобные для решаемых задач, методы, все же за основу берется одна из стандартных, общепризнанных методологий.  
Как же выбрать нужную методологию разработки ПО из такого множества вариантов? По каким параметрам можно оценить различные методологии и оценить их эффективность для решаемых задач? Основными метриками, обычно, называют уровень формализма и модель жизненного цикла проекта.  
Различные модели жизненного цикла могут изменяться от итеративной до каскадной разработки и подробно описаны на странице [этапы выполнения работ](http://project24.ru/di_developmentlevels.php).  
По степени формализма методологии могу отличаться по количеству создаваемых документов, степени их актуальности, аккуратности, полноты заполнения и формальности процедур рецензирования.

**Неконтролируемая разработка** -  имеет 2 основных проявления. Когда правил разработка ПО нет вообще, либо когда они не выполняются. Чаще всего используется новичками в разработке либо коллективами, в которых разработка программ не является профильной деятельностью. Обычно основывается на личном опыте главного разработчика либо руководителя разработки. Степень формализма низкая. Документирование разработки может отсутствовать как класс, либо заканчиваться на комментировании кода и рисовании пары картинок с общей структурой проекта. Модель жизненного цикла может быть любой, либо даже смешением нескольких моделей.  
**Гибкие методологии** (**Agile**) - в последние годы получили активное развитие и приобрели большую популярность. Основываются на 12 основных принципах:  
1. Главным приоритетом является удовлетворение потребностей заказчика за счет ранней и непрерывной поставки работоспособного программного обеспечения.  
2. Приветствуются меняющиеся требования, даже на поздних стадиях разработки. Гибкие процессы используют изменения как средство получить конкурентные преимущества для заказчика.  
3. Поставлять работоспособное программное обеспечение часто: от раза в несколько недель, до раза в несколько месяцев, отдавая предпочтение коротким интервалам.  
4. Представители заказчика (бизнеса) и разработчики должны работать вместе в течение всего проекта.  
5. Проекты строятся вокруг мотивированных личностей. Предоставьте им среду и поддержку, в которой они нуждаются, и доверьте им самим сделать работу.  
6. Наиболее эффективный способ передачи информации в команду проекта (а также передавать её внутри команды) - это непосредственное живое общение.  
7. Основной мерой прогресса проекта является работоспособное программное обеспечение.  
8. Гибкие процессы поощряют разработку с постоянной скоростью. Спонсоры проекта, разработчики и пользователи должны быть способны поддерживать постоянную скорость на неограниченной дистанции.  
9. Постоянное внимание техническому совершенству и хорошему дизайну увеличивает степень гибкости.  
10. Простота - искусство максимизации работы, которую не надо делать, - является существенным фактором.  
11. Наилучшие архитектурные решения, требования и дизайн создаются самоорганизующимися командами.  
12. Через регулярные промежутки времени команда должна проводить анализ того, как стать более эффективной и улучшать свой процесс работы.  
 Таким образом, методология является ориентированной на итеративную разработку и минимальную формализацию процесса.  
Число методологий, относящихся к этой группе, велико. Самые известные из них **eXtreme Programming** (**XP**)  - **экстремальное программирование**, **Crystal Clear** и **Feature Driven Development** (**FDD**) - **функционально-ориентированная разработка**.  
**Экстремальное программирование** - пожалуй, наиболее известная из гибких методологий. Состоит из 12 основных принципов: игра в планирование, переработки кода (refactoring),  короткие релизы, метафоры, простой дизайн, разработка «тестами вперед», коллективное владение кодом, парное программирование, 40-часовая рабочая неделя, постоянное присутствие заказчика и стандарты кода. При использовании XP тщательное планирование разработки ПО заменяется на постоянной присутствие заказчика, готового ответить на любые вопросы разработчиков и постоянную переработку (рефакторинг) кода. Низкая формализация разработки обычно не идет дальше хорошего комментирования кода, что позволяет сократить большое количество времени разработки, а следовательно, снизить общую стоимость разработки. При этом существенное внимание уделяется тестированию. Зачастую для каждого модуля системы сначала пишется тест, который продолжает выполняться на протяжении всей разработки после любого изменения кода. Впрочем, не все принципы являются строго обязательными. Например 40-часовая рабочая неделя и парное программирование носят второстепенный характер.  
**Crystal Clear** - методология, позволяющая менять  степень формализации процесса разработки в зависимости от критичности задач и количества участников разработки. Основные особенности: итеративная инкрементная разработка, автоматическое регрессионное тестирование, пользователи привлекаются к активному участию в проекте, состав документации определяется участниками проекта, как правило, используются средства контроля версий кода. Ориентирована на поддержание естественных привычек разработчиков. Считается, что если в организации не используется какая-то определенная методология, то достаточно квалифицированный коллектив рано или поздно сам естественным образом придёт к этой **Crystal Clear**.  
**Feature Driven Development** (**FDD**) - функционально-ориентированная разработка. Разработка состоит из 5 основных этапов: разработка общей модели, составление списка необходимых функций системы, планирование работы над каждой функцией, проектирование функций, конструирование функций. Работа над проектом происходит итеративно, частый выпуск версий, каждая из которых реализует определенный набор функций.  
**SCRUM** - методология, предназначенная для небольших команд (до 10 человек). Весь проект делится на итерации (спринты) продолжительностью 30 дней каждый. Выбирается список функций системы, которые планируется реализовать в течении следующего спринта. Самые важные условия - неизменность выбранных функций во время выполнения одной итерации и строгое соблюдение сроков выпуска очередного релиза, даже если к его выпуску не удастся реализовать весь запланированный функционал. Руководитель разработки проводит ежедневные 20 минутные совещания, которые так и называют - scrum, результатом которых является определение какие функции системы были реализованы за предыдущий день, с какими сложностями столкнулись и что планируется сделать за следующий день. Такие совещания позволяют постоянно отслеживать ход проекта, быстро выявлять возникшие проблемы и оперативно на них реагировать.  
**Разработка по ГОСТ** не является методологией разработки ПО. ГОСТы не описывают сам процесс разработки, а только формулируют предъявляемые к нему требования. В настоящее время используются немного устаревшие, но все еще актуальные ГОСТы 19 и 34 серии, а так же более новый ГОСТ 12207. Следуя ГОСТам разработку следует разделить на несколько строго определенных этапов, по завершении каждого из которых составляется достаточно большая по объему и строго форматизированная документация. Таким образом, разработка программного обеспечения по ГОСТам приводит к каскадному подходу и высокому уровню формализации формализованной разработки. В последнее время такой подход имеет все меньшую популярность и используется в основном для государственных заказчиков.  
**Rational Unified Process** (**RUP**) — методология разработки программного обеспечения, созданная компанией Rational Software. В основе методологии лежат 6 основных принципов:  
Компонентная архитектура, реализуемая и тестируемая на ранних стадиях проекта.  
Работа над проектом в сплочённой команде, ключевая роль в которой принадлежит архитекторам.  
Ранняя идентификация и непрерывное устранение возможных рисков.  
Концентрация на выполнении требований заказчиков к исполняемой программе.  
Ожидание изменений в требованиях, проектных решениях и реализации в процессе разработки.  
Постоянное обеспечение качества на всех этапах разработки проекта.  
Использование методологии **RUP** направлено на итеративную модель разработки. Полных жизненный цикл программы состоит из 4 фаз:

**Inception** (начало)

**Elaboration** (проектирование)

**Construction** (построение)

**Transition** (внедрение)  
Особенность методологии состоит в том, что степень формализации может меняться в зависимости от потребностей проекта. Можно по окончании каждого этапа и каждой итерации создавать все требуемые документы и достигнуть максимального уровня формализации, а можно создавать только необходимые для работы документы, вплоть до полного их отсутствия. За счет такого подхода к формализации процессов методология является достаточно гибкой и широко популярной. Применима как в небольших и быстрых проектах, где за счет отсутствия формализации требуется сократить время выполнения проекта и расходы, так и в больших и сложных проектах, где требуется высокий уровень формализма, например, с целью дальнейшей сертификации продукта. Это преимущество дает возможность использовать одну и ту же команду разработчиков для реализации различных по объему и требованиям проектов без необходимости освоения новой методологии.

# Технологии разработки программного обеспечения

Существует множество платформ для разработки Web-приложений (.Net Framework, Java, PHP, Python и многие другие). Большая часть выполненных нами проектов работают со следующей связкой:  
**Web-сервер IIS 7.0**, установленный на  Windows Server 2008 RC2 – новейшая разработка компании Microsoft, сочетающая в себе простоту настройки, легкость управления, высокую производительность и безопасность.  
**Среда разработки MS Visual Studio 2008, Framework 3.5** – мощное и простое в использовании средство для быстрой и эффективной разработки приложений с использованием всех передовых технологий. Платформа позволяет, при необходимости, легко перенести часть функционала или превратить всю систему в обычное приложение Windows.  
**Язык программирования C# + JavaScript** для клиентских скриптов. C# - популярный и мощный язык программирования, позволяющий разработать сложные приложения. JavaScript – скриптовый язык, выполняемый браузером пользователя. Существенно повышает удобство использования системы, позволяет производить оперативные подсчеты и использовать красивые эффекты.  
**Сервер баз данных MSSQL Server 2005 (2008)** – одна из самых популярных систем управления базами данных. Имеет высокую производительность, позволяет работать с большими объемами данных, есть бесплатный вариант для небольших приложений, множество дополнительных функций, удобная среда разработки.  
Большинство разрабатынных приложений это **веб-приложения**, т.е. приложения, к которым можно получить доступ из любой точки мира посредством сети Интернет.  Приложения представляют собой сайт, войдя в защищенную область которого пользователь получает весь необходимый ему функционал.  
Почему же именно эта технологическая платформа была выбрана нами приоритетной в разработке программного обеспечения? Давай оценим ее достоинства и недостатки.

Достоинства:  
•    Получить доступ к системе может любой человек из любой точки мира с практически любого устройства, подключенного к интернет. Это может быть компьютер, ноутбук или даже мобильный телефон. Единственное требование – подключение к Интернет и возможность запустить браузер для просмотра сайтов.  
•    Доступность системы 24 часа в сутки 7 дней в неделю 365 дней в году.  
•    Любые данные или изменения, внесенные пользователем в систему, тут же становятся доступны всем остальным пользователям системы.  
•    Существенно снижаются требования, предъявляемые к аппаратному обеспечению пользователей. Это может быть старенький, маломощный компьютер покажет высокую производительность, ведь все вычисления происходят на сервере, а клиенту передаются только обработанные данные.  
•    Повышенная надежность и сохранность данных. Все данные хранятся на сервере, следовательно, легко создавать резервные копии. Данные никуда не пропадут и не потеряются.   
•    Безопасность доступа к данным не ниже, чем при работе с обычными, десктопными, приложениями. При необходимости, безопасность может быть легко повышена за счет создания защищенного соединения и дополнительной идентификации пользователе с помощью аппаратных или программных ключей.  
•    Защита данных. Все данные находятся на удаленном сервере, защищенном от взломов, вирусов, троянов и прочего вредоносного ПО.  
•    Дешевизна обслуживания системы обусловлена тем, что не требуется никакой дополнительной защиты, не нужно создавать дополнительные серверные или аппаратные комнаты и мощными и дорогими серверами. Достаточно лишь найти подходящего провайдера, который позаботиться об этом сам, либо разместить сайт на одной из наших площадок. Подробнее он услуге читайте на станице [размещение и поддержка сайтов](http://project24.ru/siteshostingandsupport.php).  
•    Легкая масштабируемость. Если вам нужно предоставить доступ к системе новому пользователю или группе то, все что вам нужно – просто предоставить им учетные данные для входа в систему.  
•    Если нагрузка на сервер слишком возросла  - Вам нужно просто поставить более мощный сервер или, если сервер находится у хостинг-провайдера, договориться о другом тарифном плане.  
•    Нет необходимости покупать дорогое программное обеспечения для серверов и рабочих станций. Его можно взять в аренду у хостера вместе с сервером.

Недостатки:  
•    Требуется доступ в Интернет. Некоторые приложения могут требовать хороший канал связи.  
•    Иногда невозможно реализовать некоторые функции, доступные десктопному приложению.

Описанная платформа является основной, но не единственной. Каждая зада уникальна и для каждого случая может быть подобрано индивидуальное решение. При необходимости проект может быть разработан, например, с использование только Open Source решений, для мобильной платформы или для других операционных систем (Linux, MacOS).

# Этапы разработки программного обеспечения

Обычно, говоря о разработке программного обеспечения (ПО), выделяют 6 основных этапов:  
1.    [Планирование](http://project24.ru/di_developmentlevels.php#plan)  
2.    [Составление требований](http://project24.ru/di_developmentlevels.php#need)  
3.    [Проектирование](http://project24.ru/di_developmentlevels.php#proj)  
4.    [Разработка](http://project24.ru/di_developmentlevels.php#dev)  
5.    [Тестирование](http://project24.ru/di_developmentlevels.php#test)  
6.    [Сопровождение](http://project24.ru/di_developmentlevels.php#sop)  
Последовательность и длительность этапов зависит от выбранной методологии разработки ПО и может существенно отличаться.

**Планирование программного проекта** - в зависимости от потребностей и выбранной методологии разработки может либо вообще отсутствовать, либо занимать достаточно большую часть разработки. На этом этапе определяются основные задачи, которые должны быть решены в рамках разработки ПО, производится оценка необходимого функционала, техническое обследование объекта автоматизации, оценка финансовых, временных, человеческих, технических и других ресурсов, необходимых для осуществления разработки. Так же определятся, какие будут использованы методы разработки и тестирования. Могут быть построены временные графики, составлен бюджет, план работ и прочие документы.  
**Составление требований заказчика** - на этом этапе происходит сбор, анализ и формализация требований к разрабатываемому ПО со стороны заказчика. Этап служит для выработки максимального взаимопонимания между заказчиком и исполнителем. Обсуждаются форма предоставления информации, необходимый функционал, проблемы и ограничения, которые могут возникнуть при разработке. Составленные требования могут быть протестированы.  
**Проектирование программного продукта** - на этом этапе происходит разработка и детализация модели разрабатываемого программного продукта. На основании построенной модели определяется структура и архитектура ПО, организация и взаимодействие модулей и интерфейсов, структура базы данных, строится диаграмма классов и т.д.. Процесс проектирования проводится с учетом методологии, выбранной на этапе планирования. Может быть составлен прототип разматываемого ПО.  
**Разработка программного обеспечения** - единственный этап, которые не может быть пропущен, вне зависимости от выбранной методологии. На этом этапе происходит преобразования результатов проектирования системы в программный код на используемом языке программирования в соответствии с используемыми стандартами кодирования. На этом же этапе разработчики предоставляют информацию инженерам по тестированию для разработки комплекса тестов, разрабатывают техническую документацию и производят планирование интеграции ПО.  
**Тестирование программного обеспечения** - этап, не имеющий четко определенного начала. Может начаться еще на этапе составления требований. Чем раньше начнется тестирование тем выше вероятность, что программное обеспечение будет в точности соответствовать требованиям и потребностям заказчика, тем раньше будут выявлены критические ошибки проектирования и разработки и тем дешевле обойдется их исправление. Тестирование может проводиться в ручном или автоматическом режиме. По результатам тестов составляется отчет. Методы и виды тестирования подробно описаны на странице [тестирование программного обеспечения](http://project24.ru/di_testing.php).  
**Сопровождение программного обеспечения** - на этом этапе основное внимание уделяется внесению изменений в программное обеспечение. Изменения могут быть связаны с доработками по желанию заказчика, устранением ошибок, изменением функционала или среды окружения. Так же осуществляется консультация, обучение и поддержка пользователей.  
Перечисленные этапы, в зависимости от выбранной методологии и модели жизненного цикла, могут следовать друг за другом (в каскадной модели), проходить параллельно либо периодически (в итерационной модели). Под моделью жизненного цикла разработки программного обеспечения подразумевается последовательность процессов, задач и действий, выполняемые при разработке программного обеспечения. Выбор модели зависит от требований, предъявляемых к разработке, выбранной методологии, сложности проекта и прочих условий.  
   
Наибольшей популярность пользуются:  
1.    [Каскадная модель (водопад)](http://project24.ru/di_developmentlevels.php#kask)  
2.    [V-образная модель](http://project24.ru/di_developmentlevels.php#v)  
3.    [Прототипирование](http://project24.ru/di_developmentlevels.php#prot)  
4.    [Быстрая разработка (RAD-модель)](http://project24.ru/di_developmentlevels.php#rad)  
5.    [Многопроходная модель](http://project24.ru/di_developmentlevels.php#iter)  
6.    [Спиральная модель](http://project24.ru/di_developmentlevels.php#spir)

**Каскадная модель** (водопад) - все этапы разработки следуют друг за другом в строгой последовательности. Переход к следующему этапу происходит только после окончания предыдущего. Возвращение к пройденным этапам и внесение изменений не предусмотрено. По завершении каждого этапа создается полный комплект документации. Качество разработанного программного обеспечения оценивается по точности соответствования требованиям технического задания. Каскадный подход обычно применяют, если в начале разработки удается достаточно точно сформулировать все требования. В током случае разработка идет выполняется в логичной последовательности, что позволяет достаточно точно оценить сроки выполнения работ и их стоимость. К недостаткам модели относится отсутствие гибкости, сложность и дороговизна внесения изменений в разрабатываемый проект.  
**V-образная модель** - является одной из разновидностей каскадной модели. Особое внимание уделяется тестированию продукта. Выполнение всех этапов разработки происходит, как и в каскадной модели, последовательно. Отличие состоит в том, что каждому из этапов проектирования и разработки соответствует определенный этап тестирования. Так, после кодирования выполняется Модульное и Интеграционное тестирование, проверяющее качество проектирования и реализации системы, затем выполняется Системное тестирование, проверяющее соответствие и правильность требований к программному продукту. Затем начинает этап эксплуатации и сопровождения, на котором проверяется работоспособность проекта в целом и его соответствие поставленным задачам. По сравнению с Каскадной моделью имеет преимущество в высоком качестве тестирования и контроля разработки. Модель, чаще всего, использует если предъявляются особо высокие требования к надежности программного обеспечения.  
**Модель прототипирования** - отличается от предыдущих моделей тем, что на ранних этапах проектирования и разработки программного продукта создается его прототип. Потенциальные пользователи начинают использовать созданный прототип, выявляют его сильные и слабые стороны. За счет этого обеспечивается обратная связь между пользователями и разработчиками, позволяющая вносить изменения в проект на ранних этапах его разработки и проектирования, что существенно снижает стоимость вносимых изменений и максимально приближает готовый продукт к потребностям заказчика. Данная модель, обычно, применяется в случаях, когда требования к разрабатываемому программному обеспечению заранее не известны, неполны или требуют уточнения. Так же эта модель хорошо использовать, если производится новая разработка, не имеющая аналогов.  
**Модель быстрой разработки** (**RAD-модель**) - как понятно из названия, модель направленна на быструю разработку приложений. Особенность модели состоит в постоянном привлечении к работе заказчика, что существенно снижает риски внесения изменений в проект на поздних стадиях, укоряет выработку требований и апробацию прототипов разрабатываемого программного обеспечения. Разработка ориентирована на автоматизацию программирование, повторное использование кода и использование уже готовых сторонних модулей.  
**Многопроходная модель** (**Итерационная**) -  отличается тем, что этапы проектирования, кодирования и тестирования программного продукта за все время разработки проводятся насколько раз (проходят насколько итераций).

**Спиральная модель** - очень похожа на многопроходную модель. Особенность модели состоит в том, что ПО создается не сразу целиком, а по частям. Создается прототип, который можно начинать использовать. При этом производится постепенное наращивание функционала, улучшение и переработка интерфейса, доработки и исправление ошибок. При выборе спиральной модели не требуется раннее формулирование всех требований к программному обеспечению, поскольку они уточняются в процессе разработки. Модель заключает все достоинства каскадной и интеграционной модели, позволяет заказчику увидеть программный продукт на более ранних этапах и вовремя внести корректировки в разработку. Особенность модели состоит в том, что она может не иметь четко ограниченных по времени этапов и продолжать до бесконечности, поскольку любая реакция заказчика может вызвать новый цикл разработки. Является одной из самых гибких моделей. Подходит для больших и очень сложных проектов, а так же для проектов, требования к которым заранее не известны.

# Тестирование (системы качества)

Вопрос качества разрабатываемого программного обеспечения становиться все более актуальным, особенно если речь идет о продуктах, обеспечивающих выполнение основного бизнес-процесса компании, о фронт-офисных продуктах, которые используют клиенты, или о продуктах к надежности которых предъявляются повышенные требования.  
Основная задача тестирования – проверка,с оответствует ли работа программы предъявляемым требованиям. По другому, выполняет ли программа то, что от нее ожидает заказчик.   
Тестирование может начать еще на этапе согласования с заказчиком требований к програмному продукту. Чем раньше начнется тестировани и чем раньши будут выявлены ошибки тем дешевле обойдется их исправлени.  
В зависимых от предъявляемых тербований к програмному обеспечению тестироватся могут сами требования, на полноту и непротиворечивость, крограмный код, на наличие ошибок, функционал системы, на соответствие требованиям и документация на полноту, соответствие требованиям и работе системы.  
Можно выделить  пять уровней тестирования: модульное, интеграционное, системное, выходное и приемочное.  
Модульное тестирование – этот этап тестирования является частью разработки. Проводится непосредственно разработчиками  и представляет собой проверку отдальных частей системы (модулей), отдлельных процедур или функций системы. На этом этапе производится проверка правильности работы функционала, организации данных каждого модуля, производится синтаксическая проверка программного кода, соответствие стандартам кодирования и производится технических обзор кода.  
**Интеграционное тестирование** – в ходе этапа производится проверка совместной работы отдельных модулей, связь между модулями, корректность обмена данными, проверка функциональности, проверка промежуточных результатов работы модулей и проверка интерграции. Осущаствляется тестировщками. По результатам тестирования составлется отчет.  
Системное тестирование – производится проверка программного продукта вцелом, проверка фунциональности, соответствования трбованиям, тестирование работы системы в граничных условиях, проверка документации. Если удается – производится целевое тестирвоание – проверка работы системы на целевой платформе. Выполняется тестироващиком после завершения интерграционного тестирования. По результатам тестов составляется отчет.  
**Выходное тестирование** – производится на завершающем этапе разработки. Проверяется готовность системы к запуску в эксплуатацию. Производится конечное тестирование системы, проверка  подсистмы инсталяции (если такая система предусмотрена функцоналом), проверки документации пользователем. Может осуществляться проверка совместимости с предыдущими версиями, сторонним ПО и прочее. По редультатам тестирования составляется отчет, поторый предоставляется заказчику всесте с готовым программным продуктом.  
По желанию заказчика может производиться **приемочное тестирование**. Обычно проводится, если програмное обеспечения будет поддерживать и обслуживать не огранизация-разработчик, а сторонняя обслуживающая организация.  
Основными задачами тестирования обычно считают выявление программных ошибок и проверка соответствия работы системы техническим требованиям. Програмные ошибки деляться на функциональные недостатки, недостатки интерфейса, недостатки производительности, некорректная обработка ошибок, некорректная обработка входных данных, в частности, пограничных данных, ошибки вычисланий, ошибки управления потоком, ошибки «гонок», перегрузки, некорректная работа а аппаратным обеспечением.  
**Функциональные недостатки** – ситуации, когда программа на выполняет одну или часть своих функций либо выполняет их не верно и не полностью. Функционал программы должен быть подробно описан в спецификации. Именно  на основании расхождения спецификации и фуктической работы системы принимается решение о функциональных недостатках программы.  
**Недостатки интерфейса** могут быть выявлены в процессе работы с программным продуктом. Это можно сделать либо в результате эксплуатации готового программного обеспечения либо по средством создания прототипа, на котором производится проверка, уточнение и согласование требований. Создание прототипа является более предпочительнам, т.к. позволяет внести изменения в интерфейс и функционал программы до полного его завершения и ввода в эксплуатацию. Это может существенно удешевляет разработку и позволяет сделать программу более удобной для пользователей и более точно соответствующею выполняемым задачам.  
**Недостаточная производительность** – может являться существенным недостатком, особенно если в спецификации на разработку указаны строгие требования к производительности.  Даже если четки требований нет, слишком медленная работа программы будет создавать у пользователей негативное впечатление и требует оптимизации.  
**Некорректная обработка ошибок** – программа не должна аварийно завершать работу или продолжать работать не правильно при ошибочных действиях пользователя или не корректных входных данных. Все возникающие ошибки должны быть обратотаны, а пользователь должен получить корректное и понятно сообщение.  
**Некоррекная обработка граничных условий** – часто бывает так, что программа хорошо работает на большом диапазоне входных данных. Однако, можеть оказать, что программа не верно обрабатывает какой-то специфичный набор входных данных. Такие ситуации должны быть рассмотрены и описаны аналитиком на этапе проектирования системы и проверены тестеровщиком при ее тестировании.  
**Ошибки вычисления** – обычно носят математический характер. Могут быть связаны с неправильным выбором алгоритма расчета, не правильной формулой, недостаточной точностью вычислений, ощибкой округления и другими факторами.  
Ошибки управления потоком и ошибки гонок - чаще всего связаны с не правильным проектирование системы. Типичны для многопользовательский распределенных систем. Возникают,если порядок выполнений определенных действий в системе не соответствует проэктируемому и ожидаемому результату. Являются очень сложными для тестирования и воспроизведения.   
**Перегрузки** – могут быть связаны как с нехваткой системы резурсов для работы программы, например памяти, мощности процессора, мест на диске, так и со слишком большой нагрузкой на программу – большое число одновременно работающих пользователей, слишком большой объем обрабатываемых данных и т.д. Работы программы в таких ситуация должна быть продуманна заранее и описани в требованиях к программному обеспечению.  
**Некорректная работа с  аппаратным обеспечением** – могут возникать, если программа пытается использовать некорректно работающее устройство, либо устройство, на работу с которым программа не расчитана. Такие ситуации должны корректно обрабатываться и не приводить к сбоям или зависания программы. Ошибка, наиболее характерна для компьютерных игр.  
Помимо тестирования самого програмного продукта производится **тестирование документации**. Документация тестируется на соответствие тербования, полноту, понятность для пользователя и точность соответствия разрабатываемому програмному обеспечению. Часто бывает так, что в процессе разработки было принято решение изменить, добавить или удалить часть функционала или, например, был изменен интерфейс. По завершении проекта необходимо убедиться что окончательная версия документации точно соответствует окончательной версии програмного обеспечения. В противном случае, внести соответствующие изменения.

# Документирование разработки ПО

 Согласно государственным и международным стандартам, при разработке программного обеспечения могут быть составлены следующие документы:

•    Техническое задание  
•    Эскизный проект  
•    Технический проект  
•    Рабочий проект  
•    Программная документация  
   -    Спецификация  
   -    Ведомость держателей подлинников  
   -    Текст программы  
   -    Описание программы  
   -    Программа и методика испытаний  
   -    Пояснительная записка

•    Эксплуатационные документы  
   -    Ведомость эксплуатационных документов  
   -    Формуляр  
   -    Описание применения  
   -    Руководство системного программиста  
   -    Руководство программиста  
   -    Руководство оператора (пользователя)  
   -    Описание языка  
   -    Руководство по обслуживанию (руководство администратора)

Содержание этих документов строго регламентировано  ГОСТ 19 (Единая система программной документации (ЕСПД)), с которыми Вы можете ознакомиться на странице нормативные документы.   
Не в каждом проекте требуется создавать весь перечень документов. **Мы не тратим Ваши деньги на создание тонн макулатуры, которая в результате Вам не понадобиться**.  Для большинства проектов бывает достаточно 2-3х документов. Обычно это техническое задание и документация пользователя и администратора системы. Однако, по требованию заказчика, для любого из проектов может быть предоставлен полный перечень документов, соответствующий государственным или международным стандартам или стандартам, разработанным заказчиком.  
Документация может быть разработана как для всего программного комплекса, так и для отдельных его компонентов.