**Динамическая спецификация критического ПО**

**• Введение**

**• Определение и реферативный анализ\*) объекта изучения и предметной области выбранной темы**

**• Выводы**

**• Заключение**

**• Список фактически использованной литературы**

**Анализ требований к ПО**

Требования к программному обеспечению — совокупность утверждений относительно атрибутов, свойств или качеств программной системы, подлежащей реализации. Создаются в процессеразработки требований к программному обеспечению, в результатеанализа требований.

Требования могут выражаться в виде текстовых утверждений и графических моделей.

В классическом техническом подходе совокупность требований используется на стадии проектирования ПО. Требования также используются в процессе проверки ПО, так как тесты основываются на определённых требованиях.

Этапу разработки требований, возможно, предшествовало технико-экономическое обоснование, или концептуальная фаза анализа проекта. Фаза разработки требований может быть разбита на выявление требований (сбор, понимание, рассмотрение и выяснение потребностей заинтересованных лиц), анализ (проверка целостности и законченности), спецификация (документирование требований) и проверка правильности.

Анализ требований — часть процесса разработки программного обеспечения, включающая в себя сбор требований к программному обеспечению (ПО), их систематизацию, выявление взаимосвязей, а также документирование. Является частью общеинженерной дисциплины «инженерия требований».

В процессе сбора требований важно принимать во внимание возможные противоречия требований различных заинтересованных лиц, таких как заказчики, разработчики или пользователи.

Полнота и качество анализа требований играют ключевую роль в успехе всего проекта. Требования к ПО должны быть документируемые, выполнимые, тестируемые, с уровнем детализации, достаточным для проектирования системы. Требования могут быть функциональными и нефункциональными.

Анализ требований включает три типа деятельности:

Сбор требований — общение с клиентами и пользователями, чтобы определить, каковы их требования; анализ предметной области.

Анализ требований — определение, являются ли собранные требования неясными, неполными, неоднозначными или противоречащими; решение этих проблем; выявление взаимосвязи требований.

Документирование требований — требования могут быть задокументированы в различных формах, таких как простое описание, сценарии использования, пользовательские истории, или спецификации процессов.

Анализ требований может быть длинным и трудным процессом, во время которого вовлечены много тонких психологических навыков. Новые системы изменяют окружающую среду и отношения между людьми, поэтому важно определить все заинтересованные лица, принять во внимание все их потребности и гарантировать, что они понимают значения новых систем. Аналитики могут использовать несколько методов, чтобы выявить следующие требования от клиента: проведение интервью, использование фокус-групп или создание списков требований. Более современные методы включают создание прототипов и сценариев использования. Где необходимо, аналитик будет использовать комбинацию этих методов, чтобы выявить точные требования заинтересованных лиц так, чтобы была создана система, которая удовлетворяет деловые потребности.

Требования склонны к проблемам двусмысленности, неполноты, и несогласованности. Их устранение на этапе разработки требований стоит на несколько порядков меньше, чем устранение этих те же проблемы на поздних стадиях разработки. Анализ требований направлен на решение данных проблем.

Существует технический компромисс между слишком неопределёнными требованиями и требованиями столь детализированными что они:

1. требуют много времени для разработки, иногда даже рискуют устареть к концу разработки

2. ограничивают возможные способы реализации

3. являются слишком дорогостоящими

Фазы

Процесс анализа требований к информационной системе включает следующие фазы:

1) Разработка требований

- Выявление требований

- Анализ требований

- Спецификация требований

- Проверка требований

2) Управление требованиями

Выявление и анализ требований

1) Идентификация стейкхолдеров

Стейкхолдер — физическое лицо или организация, имеющая права, долю, требования или интересы относительно системы или её свойств, удовлетворяющих их потребностям и ожиданиям (ISO/IEC 15288:2008, ISO/IEC 29148:2011).

2) Опрос стейкхолдеров

Опрос стейкхолдеров является широко используемой техникой при сборе требований. Эти опросы могут выявлять требования, не попавшие в рамки проекта либо противоречащие ранее собранным. Однако каждый стейкхолдер будет иметь собственные требования, ожидания и видение системы.

3) Сеансы совместного развития требований (СРТ)

Требования часто имеют сложное пересекающееся функциональное назначение, не известное отдельным стейкхолдерам. Такие требования часто упускаются или не полностью определяются во время их опросов. Такие требования могут быть выявлены при проведении сеансов СРТ. Такие сеансы проводятся под надзором подготовленного специалиста. Стейкхолдеры участвуют в обсуждениях, чтобы определить требования, проанализировать их детали и выявить скрытые пересекающиеся взаимосвязи между требованиями.

**Детальное проектирование и конструкторская разработка ПО**

Если на этапе архитектурного проектирования определялись крупномасштабные проектные решения, а на этапе технического проектирования определялись группы взаимодействующих объектов, то на этапе детального проектирования выполняется подробное специфицирование отдельных объектов.

Детальное проектирование - это полный объем работ по проектированию, исключая архитектуру и реализацию. Оно содержит определение классов предметной области и классов архитектуры. Детальное проектирование – это техническая деятельность, которая следует за этапом выбора архитектуры. Основной целью этой деятельности является как можно более полная подготовка проекта к его реализации, то есть к созданию программного кода.

Существует определенная взаимосвязь вариантов использования, архитектуры и детального проектирования. Варианты использования являются частью требований, отталкиваясь от которых разработчики выбирают архитектуру, после чего они разрабатывают детальный проект для реализации требуемых вариантов использования с учетом выбранной архитектуры.

На первом шаге проектирования ПО варианты использования фиксируются как часть требований. На втором шаге проектирования варианты использования используются для определения классов предметной области. На третьем шаге – разрабатывается программная архитектура, и добавляются соответствующие классы проекта. Четвертый шаг заключается в проверке того факта, что архитектура и детальный проект удовлетворяют требуемым вариантам использования.

В общем случае на данном этапе определяются решения относительно следующих характеристик объектов:

* Структуры данных;
* Реализация ассоциаций;
* Множество операций, определенных на данных;
* Видимость данных и операций;
* Алгоритмы, используемые для реализации этих операций;
* Возбуждаемые и обрабатываемые исключения.

Типичная схема детального проектирования:

· Начинайте с архитектурных моделей, то есть, моделей классов предметной области и архитектуры; общей модели переходов состояний, потоков данных и модели вариантов использования.

· Представьте классы и образцы проектирования, которые включают в себя классы архитектуры и предметной области.

· Совершенствуйте модели, обеспечивая их непротиворечивость

· Для каждого класса определите их инварианты.

· Для каждого метода используйте пред- и пост - условия, блок схемы и псевдокод.

· Набросайте план модульного тестирования.

· Проинспектируйте планы тестирования и проектирования.

· Запускайте в реализацию.

В унифицированном процессе разработки ПО (USDP – Unified Software Development Process) детальное проектирование имеет место в основном во время итераций проектирования и конструирования. USDP поддерживает три стереотипа классов на уровне анализа, не связанных с классами проектирования: классы сущности, граничные классы и классы управления.

Проектирование с использованием интерфейсов можно сравнить с заключением контракта, причем элементы приложения, использующие эту основу функциональности, следует проектировать так, чтобы они «не знали» подробности реализации функциональности.

Следующим приемом детального проектирования является повторно используемые компоненты, развитию которого способствовало широкое распространение объектно – ориентированных, объектно - подобных и других парадигм компонентов. Примером повторного использования программного кода является применение библиотеки Microsoft MFC, элементов управления Visual Basic, объектов COM, Java Beans и других классов Java API. Стандартом для распределенного повторного использования является архитектура CORBA консорциума OMG. Веб – приложения (не компоненты), также часто бывают повторно используемыми, и, наконец, STL – библиотека стандартных шаблонов, применяется к различным структурам данных и к объектам практически любых классов.

В детальном проектировании, в качестве инструментов для проектирования, используются диаграммы последовательности и диаграммы потоков данных. Покажем соответствующие способы усовершенствования моделей для детального проектирования.

**ОС реального времени. Классификации.**

Операционные системы реального времени (ОСРВ) предназначены для обеспечения интерфейса к ресурсам критических по времени систем реального времени. Основной задачей в таких системах является своевременность выполнения обработки данных.

В качестве основного требования к ОСРВ выдвигается требование обеспечения предсказуемости или детерминированности поведения системы в наихудших внешних условиях, что резко отличается от требований к производительности и быстродействию универсальных ОС. Хорошая ОСРВ имеет предсказуемое поведение при всех сценариях системной загрузки (одновременные прерывания и выполнение потоков).

Существует некое различие между системами реального времени и встроенными системами. От встроенной системы не всегда требуется, чтобы она имела предсказуемое поведение, и в таком случае она не является системой реального времени. Однако даже беглый взгляд на возможные встроенные системы позволяет утверждать, что большинство встроенных систем нуждается в предсказуемом поведении, по крайней мере, для некоторой функциональности, и таким образом, эти системы можно отнести к системам реального времени.

Сформулированы следующие необходимые требования для ОСРВ:

- ОС должна быть многозадачной и допускающей вытеснение (preemptable),

- ОС должна обладать понятием приоритета для потоков,

- ОС должна поддерживать предсказуемые механизмы синхронизации,

- ОС должна обеспечивать механизм наследования приоритетов,

- поведение ОС должно быть известным и предсказуемым (задержки обработки прерываний, задержки переключения задач, задержки драйверов и т.д.); это значит, что во всех сценариях рабочей нагрузки системы должно быть определено максимальное время отклика.

Cтруктура ОСРВ эволюционировала от монолитной к многослойной структуре ОС и далее к архитектуре клиент-сервер:

- Монолитная структура - ОС состоит из набора модулей, и изменения одного модуля влияют на другие модули. Чем больше модулей, тем больше хаоса при эксплуатации такой системы. Кроме того, невозможно распределить ОС в многопроцессорной системе.

- Многослойная структура - изменения одного слоя влияют на соседние слои; кроме того, обращение через слой невозможно. Для систем реального времени должно быть обеспечено прямое обращение к каждому слою ОС, а иногда напрямую к аппаратуре.

- Клиент-серверная структура - сведение базиса ОС к минимуму (планировщик и примитивы синхронизации). Вся остальная функциональность выносится на другой уровень и реализуется через потоки или задачи. Совокупность таких серверных задач отвечает за системные вызовы. Приложения являются клиентами, которые запрашивают сервисы через системные вызовы. Клиент-серверная технология позволяет создавать масштабируемые ОС и упрощает распределение в многопроцессорной системе. При эксплуатации системы замена одного модуля не вызывает эффекта “снежного кома”; кроме того, сбой модуля не всегда влечет за собой отказ системы в целом. Появилась возможность динамической загрузки и отгрузки модулей. Главной проблемой в этой модели является защита памяти, поскольку серверные процессы должны быть защищены. При каждом запросе сервиса система должна переключаться с контекста приложения на контекст сервера. При поддержке защиты памяти время переключения с одного процесса на другой увеличивается.

Как правило, большинство современных ОСРВ построено на основе микроядра (kernel или nucleus), которое обеспечивает планирование и диспетчеризацию задач, а также осуществляет их взаимодействие. Несмотря на сведение к минимуму в ядре абстракций ОС, микроядро все же должно иметь представление об абстракции процесса. Все остальные концептуальные абстракции операционных систем вынесены за пределы ядра, вызываются по запросу и выполняются как приложения.

*Классификация*.

1. Системой жесткого реального времени называется система, где неспособность обеспечить реакцию на какие-либо события в заданное время является отказом и ведет к невозможности решения поставленной задачи. Жесткие системы реального времени - системы, в которых абсолютно обязательно, чтобы ответы выдавались в пределах требуемого крайнего срока. Системы жесткого реального времени не допускают никаких задержек реакции системы ни при каких условиях:

- в случае опоздания результаты окажутся бесполезными;

- в случае задержки реакции может произойти катастрофа;

- стоимость опоздания может оказаться бесконечно велика.

К системам жесткого реального времени относятся бортовые системы управления, например, системы управления полетом, системы аварийной защиты, регистраторы аварийных событий. Большинство систем жесткого реального времени являются системами контроля и управления. Такие СРВ сложны в реализации, так как для них предъявляются особые требования в вопросах безопасности.

2. Гибкие системы реального времени (мягкого реального времени) - системы, где крайние сроки важны, но которые все еще будут функционировать правильно, если крайние сроки иногда пропущены, например, система сбора данных. Система мягкого реального времени может не успевать ВСЕ и ВСЕГДА делать в заданное время. Грамотно разработанные системы, как правило, имеют уровень безопасности/коррекции поведения даже для случая, когда вычисления не закончились в необходимый момент, так что если компьютер чуть-чуть не успевает, то это может быть скомпенсировано. Системы мягкого реального времени характеризуются тем, что задержка реакции не критична, хотя и может привести к увеличению стоимости результатов и снижению производительности системы в целом.

Основное различие между системами жесткого и мягкого реального времени можно сформулировать так: если система жесткого реального времени никогда не опоздает с реакцией на событие, то система мягкого реального времени не должна опаздывать с реакцией на событие.

3.Также СРВ можно разделить на системы специализированные и универсальные:

Специализированной СРВ называется система где конкретные временные требования априори определены. Такая система должна быть специально спроектирована для удовлетворения этих требований.

Универсальная СРВ должна уметь выполнять произвольные (заранее не определенные) временные задачи без применения специальной техники. Разработка таких систем безусловно является самой сложной задачей, хотя обычно, требования, предъявляемые к таким системам, мягче чем требования для специализированных систем.

В настоящее время в документах и публикациях с различной тематикой встречаются слова «работа в режиме реального времени», «режим реального времени» или просто «реальное время». Что же такое «режим реального времени» применительно к компьютерным системам?

С уверенностью можно сказать, что смысл термина «реальное время» трактуется специалистами по-разному в зависимости от области их профессиональных интересов, от того, являются они теоретиками или практиками, и даже просто отличного опыта и круга общения.