Лекция 4 1. Основные этапы разработки

В процессе разработки интерфейса можно выделить три основных этапа, а именно первоначальное проектирование, создание прототипа и тестирование/модификация прототипа. Фактически процесс разработки, чтобы быть успешным и безусловным, всегда стремится происходить в этой последовательности: проектирование, затем создание прототипа, затем бесконечные циклы тестирование/модификация до достижения удовлетворительного результата или до тех пор пока не остановят. Т.е. основным этапом оказывается не проектирование (т.е. собственно дизайн), но полировка уже сделанного дизайна. С другой стороны, при тщательном проектировании длительного тестирования обычно удается избежать – но, с другой стороны, при этом проектирование становится достаточно длительным, так что неизвестно еще, что лучше сокращать.

Этап проектирования сам по себе состоит из нескольких составляющих, причем количество этих составляющих довольно велико. Радует, однако, то, что существует очень мало работ, при которых нужно выполнять *все* составляющие (собственно говоря, если выполнять всё-таки придется, это должны делать разные специалисты). При этом важно понимать, что здесь описываются только методы создания *новой* системы. Модернизация старой требует совершенно других методов и усилий, поскольку при этом приходится еще и безболезненно исправлять имеющиеся недостатки (что чаще всего вообще невозможно по организационным причинам). Как это ни печально, но в настоящее время не найдено сколько-нибудь универсального метода радикального улучшения существующих систем. Зачастую вместо полноценного дизайна приходится удовлетворяться наведением общего глянца, что, в лучшем случае, придает системе сомнительный шарм молодящегося старика.

В то же время методология проектирования описана в литературе слабо (основной момент уделяется менеджменту, что в нашей стране не очень актуально). Объясняется это тем, что общие принципы еще не выработаны (за исключением отраслевых стандартов военных ведомств). В результате все пользуются самодельными методами работы, в чём, по сути, нет ничего плохого (трудности появляются при синхронизации работы нескольких человек, о неактуальности чего уже сказано).

Необходимо также отметить еще и следующее. В процессе проектирования вам понадобится незаслуженно забытые инструменты – а именно ручка и бумага. Дело в том, что использование компьютера само по себе медленно, во-первых, поскольку интерфейс программ несовершенен, а во-вторых, из-за того, что, используя компьютер, вы будете подсознательно стараться сделать работу *красиво*, а не просто

будете фиксировать свою мысль. Известно, например, что текстовые процессоры не принесли ускорения письма – человек, пишущий на компьютере тратит уйму времени на полировку фраз. Владелец же печатающей машинки полирует фразы в голове, что гораздо быстрее. Более того, почти постоянно приходится делать много вариантов одного и того же – так зачем полировать до блеска вариант, который вы отбросите через пять минут? Тем более что после получения устраивающего результата всегда можно перенести его в компьютер. Бумага имеет еще и то преимущество, что её можно с успехом показывать руководству в качестве доказательств своей активности. Практика показывает, что вид сколотых скрепкой мятых исчерканных листов почти всегда производит на руководящий состав исключительное впечатление.

Роль технического писателя

Первым человеком, которого плохой интерфейс приводит в трепет, является отнюдь не конечный пользователь, а технический писатель – если интерфейс понятен, от писателя требуется мало работы, если непонятен – много. Это делает технических писателей лучшими друзьями дизайнеров интерфейсов, более того, писатели могут очень много дать дизайнерам.

Дело в том, что множество систем ни при каких обстоятельствах не могут быть используемы без подготовки, независимо от качества их интерфейса. Система автоматизации, например, может быть эффективно использована только в том случае, когда пользователь этой системы понимает *суть* автоматизируемых процессов. Обязанность же технического писателя, прежде всего, заключается в том, чтобы снабдить пользователя этим пониманием. Это значит, что концепции и суть сложной системы могут быть безболезненно вынесены из интерфейса в документацию, освобождая ресурсы дизайнера. С другой стороны, зачастую *описание* концепций влияет на их *реализацию* в системе, особенно в ситуациях, когда качество обучения работе с системой является важнейшей составляющей общего качества. Это наблюдение вполне подтверждается опытом. Например, Джеф Раскин, Отец Макинтоша, изначально был начальником отдела документации. После того, как он обнаружил, что имеющуюся систему невозможно приемлемо описать, он создал новую, описывающуюся хорошо. Побочным свойством новой системы, компьютера Макинтош, было то, что его интерфейс был понятен и удобен в работе.

Документация есть часть интерфейса, причем в сложных системах – большая часть

С практической точки зрения это значит, что технический писатель почти всегда знает о конструируемой системе не меньше, чем дизайнер, или, во всяком случае, знает *другое*. Это делает необходимость коммуникации дизайнера с писателем насущной необходимостью, более того, во многих источниках напрямую рекомендуется ставить писателя выше дизайнера, поскольку его вклад в проект более значителен.

Проблема этого подхода в наших условиях заключается в том, что роль документации чрезвычайно недооценена. Создание документации воспринимается в нашей стране как сравнительно необязательный процесс, нужный исключительно для утяжеления коробки с готовым продуктом. От этого документация почти всегда пишется после того, как система создана, более того, писателям очень мало платят. В результате работа технического писателя не приносит работнику ни денег, ни удовлетворения. Неудивительно, что средний уровень отечественных писателей, мягко говоря, невысок, что не позволяет рассчитывать на какую-либо помощь от них.

2. Первоначальное проектирование

Важность этого этапа трудно переоценить. На нем закладываются основные концепции системы, влияющие абсолютно на все показатели качества её интерфейса. Как будет описано в разделе о тестировании, структурные проблемы практически не могут быть обнаружены и решены на остальных этапах (для их обнаружения нужно слишком много везения, а для исправления – денег). Это значит, что чем больше внимания будет уделено проектированию, тем выше будет общее качество.

Собственно проектирование состоит из следующих этапов:

1. Определение необходимой функциональности системы.
2. Создание пользовательских сценариев.
3. Проектирование общей структуры.
4. Конструирование отдельных блоков.
5. Создание глоссария
6. Сборка и начальная проверка полной схемы системы.

Каждый последующий этап в такой системе зависит от результатов предыдущих этапов. Соответственно, пропуск какого-либо этапа (за исключением, разве что, создания глоссария) негативно влияет на результаты всех последующих этапов.

Определение необходимой функциональности системы

На первом этапе необходимо определить функциональность будущей системы. Это исключительно важный этап, поскольку именно функциональность будет определять весь интерфейс. Очень важно сознавать, что практически невозможно убрать из уже продающейся системы какие-либо функции. Во-первых, программы до сих пор

продаются по функциям, т.е. чем больше список функций на коробке с программой, тем легче её продать, даже если большинство функций либо не работает, либо не нужно пользователям. Происходит это потому, что существенную часть тиража программы покупают новые пользователи, которые ничего не знают про истинное положение вещей. Во-вторых, имеющиеся пользователи обычно с исключительной неохотой переучиваются для использования новых функций взамен прежних (при этом еще и обижаются из-за того, что их инвестиции в обучение оказались выброшенными на ветер). Это значит, что ненужная функция системы будет кочевать из версии в версию, раздувая размеры программы, снижая надежность и быстродействие, и, что для нас более важно, портя интерфейс (при этом и длительность разработки возрастает). Несколько лучшая ситуация с сайтами – никого пока не удивляет, когда сайт после изменения дизайна почти не напоминает прежний. Так что оптимальным вариантом работы почти всегда является проектирование функциональности сразу на несколько версий вперед.

Традиционно требования к функциональности исходят от отдела продаж или от его аналога. Такие требования имеют два источника, а именно жалобы имеющихся клиентов и системы конкурентов. К сожалению, оба эти источника сомнительны.

Всегда может оказаться, что желание клиента получить новую функцию, обусловлено не реальной потребностью в ней, а собственно концептуальными проблемами системы. Системы же конкурентов страдают как от такого же непонимания проблемы, так и от множества других причин. Это значит, что прислушиваться к сторонним требованиям к системе, безусловно, следует, но рассматривать эти требования нужно не как директивы, но как признаки общего неблагополучия. Некой фирмой было разработано техническое задание к системе каталогизации изображений, рассчитанной на домашних пользователей, которая состояла из сведенного в таблицу списка *всех* функциональных возможностей *всех* конкурентов. В результате система получила огромное количество никому не нужных возможностей, что не только никак не помогло ей на рынке, но и безмерно удлинило срок её разработки.

Не копируйте слепо интерфейс конкурентов.

Так как всё-таки определить нужную функциональность? Современная наука выдвинула два основных способа, а именно ***анализ целей*** и ***анализ действий пользователей.*** Эти способы фактически не конфликтуют друг с другом, более того, в процессе определения функциональности желательно использовать оба.

Анализ целей пользователей

Идеей, лежащей в основе данного метода, является простое соображение, гласящее, что людям не нужны инструменты сами по себе, нужны лишь результаты их работы. Никому не нужен молоток, если не нужно забивать гвозди. Никому не нужен текстовый процессор, но нужна возможность, с удобством, писать тексты. Никому не нужна программа обработки изображений – нужны уже обработанные изображения.

Это значит, что сами по себе функции никому не нужны и не важны. Людям нужно средство вообще, делающее возможным выполнять какую-либо работу. Разницу подходов к выбору функциональности в такой системе удобно проиллюстрировать на примере тостера. Стандартный подход, при котором функции выбираются фактически произвольно, в лучшем случае приведет к такому заданию:

«Нужен ящик с узкой прямоугольной дыркой и нагревателем внутри»

Анализ целей пользователя приведет к другой формулировке:

«Нужен поджаренный хлеб. Похоже, что проще всего добиться этого созданием ящика с дыркой по форме куска хлеба и нагревателем внутри. С другой стороны, похоже, что этот способ не единственный».

Второй вариант при полном развитии этого метода может привести не только к созданию тостера, но и также и ростера (т.е. устройства, в котором можно поджаривать *не только* хлеб). Главное же другое. Ни в коем случае нельзя дать обмануть себя ненужной конкретикой, т.е. описанием того, *какова* должна быть будущая функциональность. Как правило, одного и того же результата можно добиться несколькими разными способами, при этом важно не только реализовать какой-либо способ, но и выбрать лучший.

Анализ целей пользователя как раз и позволяет избежать ненужной конкретики.

Результатом этого процесса должен являться список целей, например, для тостера финальный список целей должен выглядеть очень просто:

«Должен поджаривать мелкие предметы, преимущественно хлеб»

После того, как истинные цели пользователей установлены (и доказано, что таких пользователей достаточно много, чтобы оправдать создание системы), приходит время выбирать конкретный способ реализации функции, для чего используется второй метод.

Анализ действий пользователей

Достижение почти всех целей требует от пользователей совершения определенных действий. Разумеется, эти действия могут различаться при разных способах достижения. В сколько-нибудь сложных интерактивных

системах сами по себе выбранные стратегии действий влияют на требования к функциональности.

Возвращаясь к примеру с тостером, можно указать, что помимо возможности поджаривать хлеб он должен включаться и выключаться, более того, он должен быть устроен таким образом, чтобы его можно было удобно мыть. С другой стороны, возможно, что тостер можно сделать так, чтобы он не требовал мойки, в этом случае функция «возможность мыть» становится лишней. Разумеется, взаимодействие человека с тостером очень просто, здесь можно дойти до всего чисто логическим анализом. В компьютерных же системах взаимодействие обычно многократно сложнее, при этом логический анализ неприемлем.

Единственным выходом является банальное наблюдение за людьми, выполняющими свою задачу, пользуясь уже имеющимися инструментами, а именно системами конкурентов (если они есть) и предметами реального мира (поскольку очень немного новых действий появилось только после появления компьютеров). Неплохим источником материала для анализа часто служит даже не наблюдение за людьми, но анализ результатов их работы – если оказывается, что результат работы практически не зависит от используемого инструмента, это значит, что нужна только та функциональность, которая оказала воздействие на результат (т.е. функции, которыми никто не воспользовался, не нужны).

Тут очень важно избежать эгоцентризма. Очень трудно отказаться от мысли «если это нужно мне, это нужно и многим». Возможно, что и нужно. А возможно, что и нет. Единственным же способом проверить, нужна функция или нет, является наблюдение за пользователями и анализ их действий.

Как уже было сказано, обычно есть несколько разных способов реализации одной и той же функции. Анализ действий пользователей как раз и позволяет определить, какой именно способ следует реализовывать. Поскольку на этом этапе мы узнаём, какая именно функциональность нужна для каждого варианта, можно избрать верный путь по правилу «чем меньше действий требуется от пользователя, тем лучше» (благо компьютер есть, прежде всего, великое средство автоматизации). Не стоит забывать и про другое правило: чем меньше функций, тем легче их сделать.

Низкоуровневые и высокоуровневые функции

Существует два принципиально разных подхода к определению функциональности системы. При первом подходе система снабжается максимальным количеством функций, при этом результаты многих из них являются суммой результатов других функций. При другом подходе все составные функции (метафункции) из системы изымаются. Ярким

представителем первого подхода является Corel PhotoPaint, не менее ярким представителем другого – Adobe PhotoShop.

Оба подхода имеют как недостатки, так и достоинства. Подход, при котором количество функций ограничено, позволяет упрощать интерфейс, но при этом требует от пользователя понимать, как из многих низкоуровневых функций «собирать» функции более сложные. Подход, при котором помимо низкоуровневых функций есть высокоуровневые, позволяет потенциально обеспечивать большую скорость работы (за счет отсутствия пауз между низкоуровневыми функциями), но зато требует от пользователя знаний о том, где эти высокоуровневые функции найти и как с ними работать, при этом они перегружают интерфейс.

Судя по всему, людям больше нравится пользоваться низкоуровневыми функциями, поскольку это позволяет добиваться более тонких и предсказуемых результатов. С другой стороны, доказательств этого не так уж много (сам факт того, что PhotoShop продается лучше, чем PhotoPaint, говорит не так уж о многом). Таким образом, выбор подхода не слишком прост. С другой стороны, остается возможность компромисса: всегда можно включить в систему средства автоматизации, чтобы пользователи получали возможность создавать (и распространять) свои метафункции, каковой подход как раз и применен в PhotoShop.

Создание пользовательских сценариев

Это самый веселый этап работы. Его цель – написать словесное описание взаимодействие пользователя с системой, не конкретизируя, как именно проходит взаимодействие, но уделяя возможно большее внимание всем целям пользователей. Количество сценариев может быть произвольным, главное, что они должны включать все типы задач, стоящих перед системой, и быть сколько-нибудь реалистичными. Сценарии очень удобно различать по именам участвующих в них вымышленных персонажей.

Предположим, что необходимо разработать сценарии для будущей почтовой программы. Судя по всему для этой задачи необходимо три сценария:

А) запустить почтовую программу, скачать новую почты. Получив почту, прочитать все сообщения, затем часть их удалить, а на одно сообщение ответить. После чего выключить почтовую программу.

Б) Сделать активным окно уже открытой почтовой программы и включить процесс скачивания новой почты. Получив почту, прочитать ее. Одно сообщение переслать другому адресату, после чего удалить его, а еще одно печатать. После чего переключиться на другую задачу.

С) Пришло новое сообщение, о чем сообщает соответствующий индикатор. Сделать активным окно почтовой программы и открыть

полученное сообщение. Прочитать его, после чего переместить его в другую папку. После чего переключиться на другую задачу.

Польза этих сценариев двояка. Во-первых, они будут полезны для последующего тестирования. Во-вторых, сам факт их написания обычно (если не всегда) приводит к лучшему пониманию устройства проектируемой системы, побуждая сразу же оптимизировать будущее взаимодействие.

Дело в том, что на таких сценариях очень хорошо заметны ненужные шаги, например, в третьем сценарии после получения сигнала индикатора необходимо было открыть окно системы, найти нужное сообщение, открыть его и только тогда прочесть. Понятно, что от этих ненужных этапов смело можно избавиться уже на этой, весьма ранней, стадии проектирования.

Проектирование общей структуры

Итак, информация о будущей системе собрана. Теперь, пользуясь этой информацией, необходимо создать общую структуру системы («вид с высоты птичьего полета»), т.е. выделить отдельные функциональные блоки и определить, как именно эти блоки связываются между собой. Под отдельным функциональным блоком будем понимать функцию/группу функций, связанных по назначению или области применения в случае программы и группу функций/фрагментов информационного наполнения в случае сайта.

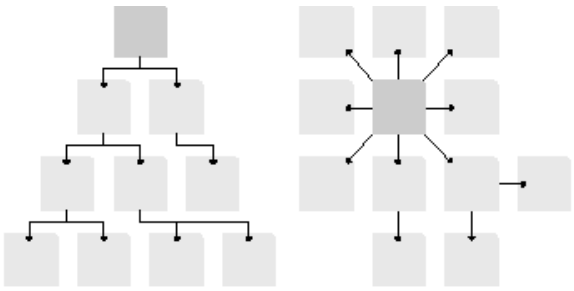


Рис. 1. Типичная структура сайта (слева) и типичная структура программы. Если сайты обычно разветвлены, в том смысле, что функции обычно размещаются в отдельных экранах, то программы обычно имеют только один изменяющийся экран, в котором и вызываются почти все функции.

Проектирование общей структуры состоит из двух параллельно происходящих процессов: *выделения независимых блоков* и *определения связи между ними*. Если проектируется сайт, в завершении необходимо также создать схему навигации.

Выделение независимых блоков

Для этой работы трудно дать какие-либо конкретные рекомендации, поскольку очень многое зависит от проектируемой системы. Тем не менее, можно с уверенностью рекомендовать избегать помещения в один блок более трех функций, поскольку каждый блок в результирующей

системе будет заключен в отдельный экран или группу управляющих элементов. Перегружать же интерфейс опасно.

Результатом этой работы должен быть список блоков с необходимыми пояснениями. В качестве примера рассмотрим гипотетическую программу ввода данных. От пользователя требуется выбрать из списка клиента (или добавить в список нового) и указать, какие именно товары клиент заказал (товары в список тоже можно добавлять). Несколько клиентов постоянно что-то заказывают, так что заставлять пользователя каждый раз искать в списке такого клиента неправильно. При этом блоки разделяются следующим образом:

Основной экран	Навигация между функциями системы
Создание нового заказа	
Добавление существующего товара в заказ	А так же простой поиск товара в списке
Сложный поиск товара	
Добавление нового товара в список	
Добавление существующего клиента в заказ	А так же простой поиск клиента в списке
Добавление нового клиента в список	
Выбор постоянного клиента	
Обработка заказа	Печать и его переход в папку Исполняемые

Определение смысловой связи между блоками

Существует три основных вида связи между блоками. Это *логическая* связь, связь *по представлению пользователей* и *процессуальная* связь. Логическая связь определяет взаимодействие между фрагментами системы с точки зрения разработчика (суперпользователя). Пользователи имеют свое мнение о системе, и это мнение тоже является важным видом связи. Наконец, процессуальная связь описывает пусть не вполне логичное, но естественное для имеющегося процесса взаимодействие: например, логика напрямую не командует людям сначала приготовить обед, а потом съесть его, но обычно получается именно так. Все три типа взаимосвязи должны быть заранее предусмотрены при конструировании системы. Разберем это подробнее.

*Логическая связь***.** С установлением логической связи между модулями обычно проблем не возникает. Важно только помнить, что полученные связи очень существенно влияют на навигацию в пределах системы (особенно, когда система многооконная). Поэтому, чтобы не перегружать интерфейс, стоит избегать как слишком уж отдельных

блоков (их трудно найти), так и блоков, связанных с большим количеством других. По опыту, для одного блока оптимальным числом связей является число три.

*Связь по представлению пользователей***.** В информационных системах, когда необходимо гарантировать, что пользователь найдет всю нужную ему информацию, необходимо устанавливать связи между блоками, основываясь не только на точке зрения разработчика, но основываясь на представлениях пользователей. Дело в том, что чуть ли не единственный распространенный способ поиска, а именно поиск по классификации признаков, работает только в том случае, когда пользователи согласны с принципами этой классификации. Большинство же понятий однозначно классифицированы быть не могут из-за наличия слишком большого количества значимых признаков. Также проблема состоит в том, что реальный классификационный признак может отличаться от широко распространенного признака. Например, нужно как-то классифицировать съедобные растения. Помидор, который почти все считают овощем, на самом деле ягода. Не менее тяжело признать ягодой арбуз. Это значит, что классификация, приемлемая для ботаника, не будет работать для всех остальных, причем обратное не менее справедливо.

В то же время, существует очень простой способ классификации, с явной ненаучностью, сочетающий не менее явную практическую пользу. Способ этот называется карточной сортировкой, при этом его название полностью совпадает с его сущностью. Все понятия, которые требуется классифицировать, пишутся на бумажных карточках из расчета «одно понятие – одна карточка». После чего группе пользователей из целевой аудитории предлагается эти карточки рассортировать (при этом каждый субъект получает свой набор карточек). Получившиеся кучки из карточек нужно разобрать на составляющие и свести результаты от разных субъектов в один способ классификации. Ничего более работоспособного до сих пор человечеством не придумано. В то же время этот способ имеет определенные недостатки: во-первых, трудно заполучить на несколько часов представителей целевой аудитории, а во-вторых, при малом количестве субъектов результаты могут быть сомнительны (как минимум, нужно 4-5 человек).

*Процессуальная связь***.** Установление качественной процессуальной связи обычно довольно трудная задача, поскольку единственным источником информации является наблюдение за пользователями. В то же время установление такой связи дело исключительно полезное. Зачем, например, рисовать на экране сложную систему навигации, если точно известно, к какому блоку пользователь перейдет дальше? В этом смысле

зачастую оправдано навязывать пользователю какую-либо процессуальную связь, жертвуя удобством, зато, выигрывая в скорости обучения (поскольку пользователю приходится думать меньше). Жестко заданная связь позволяет также уменьшить количество ошибок, поскольку от пользователя при ней не требуется спрашивать себя «не забыл ли я чего?». Замечательным примером жестко заданной процессуальной связи является устройство мастеров (wizards), при котором пользователя заставляют нажимать кнопку Далее.

Результат. В конце этого этапа должна получиться примерно такая схема (Рисунок 2).

Существует любопытная закономерность: чем эстетически привлекательней выглядит схема (без учета цветового кодирования и веселеньких шрифтов), тем она эффективней. Всегда надо стараться сделать схему возможно более стройной и ясной.

Рисовать такие схемы очень удобно в MS Visio или подобной ей системе. Результат, в особенности сложно выглядящий и напечатанный на листе бумаги большого формата, очень хорошо смотрится на стене и животворяще действует на заказчиков и руководство.

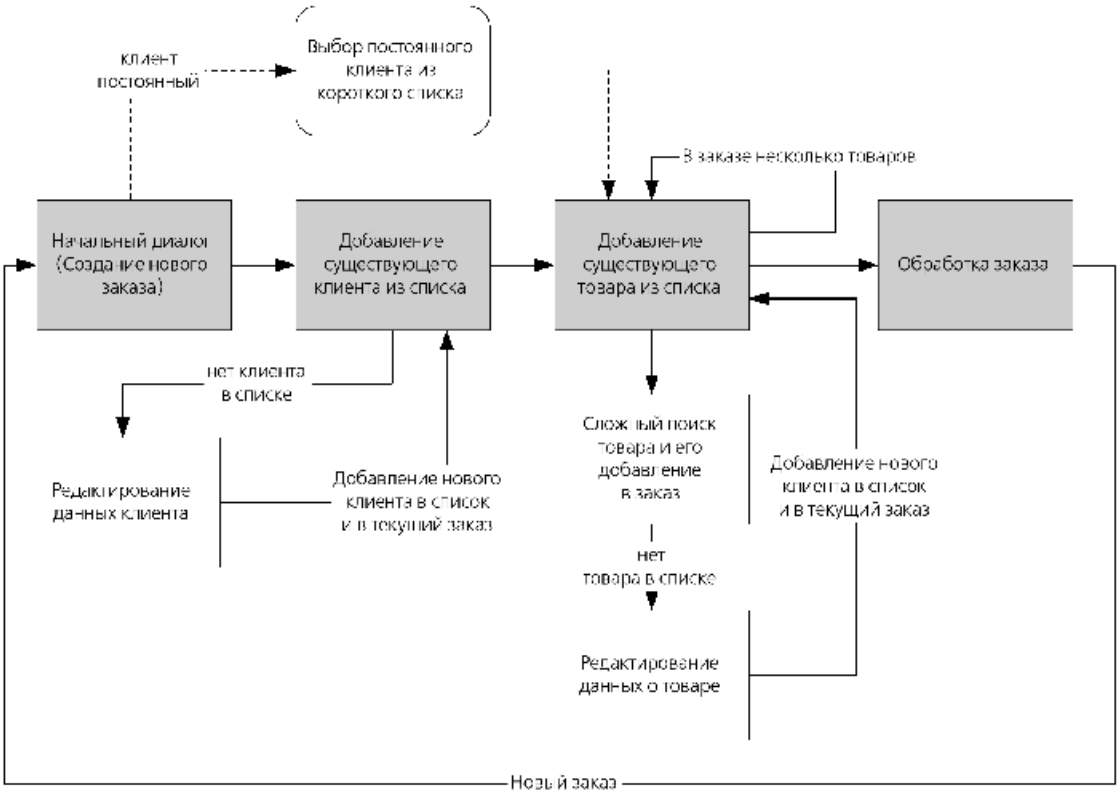


Рис. 2. Пример общей схемы. Прямоугольник обозначает отдельный экран, прямоугольник со скругленными углами – область экрана, пунктирная линия – альтернативное действие. Обратите внимание, что в этой схеме интерфейс заставляет пользователя выполнять задачу в сугубо определенной последовательности.

Проектирование отдельных блоков

Итак, теперь вы знаете, сколько экранов (страниц) вам нужно и что должно происходить на каждом экране. Настало время проектировать отдельные экраны. Это, пожалуй, самая сложная часть работы (не считая наблюдения за пользователями). Хуже того, она плохо поддается алгоритмизации. Но помимо этого есть еще две вещи, которые вам нужно узнать: **GOMS** и **адаптивная функциональность.**

Предсказание скорости

Часто приходится выбирать между разными вариантами реализации интерфейса, причем отбрасывать варианты жалко, потому что они хорошие. Можно, конечно, сделать несколько прототипов и протестировать их на пользователях, но это довольно длительный и трудоемкий процесс. К счастью, есть метод оценки интерфейса, позволяющий быстро выбрать лучший вариант.

В 1983 году Кард, Моран и Ньювел создали метод оценки скорости работы с системой, названный аббревиатурой GOMS (Goals, Operators, Methods, and Selection Rules – цели, операторы, методы и правила их выбора). Идея метода очень проста: все действия пользователя можно разложить на составляющие (например, взять мышь или передвинуть курсор). Ограничив номенклатуру этих составляющих, можно замерить время их выполнения на массе пользователей, после чего получить статистически верные значения длительности этих составляющих. После чего предсказание скорости выполнения какой-либо задачи, или, вернее, выбор наиболее эффективного решения, становится довольно простым делом – нужно только разложить эту задачу на составляющие, после чего, зная продолжительность каждой составляющей, всё сложить и узнать длительность всего процесса. Обычно тот интерфейс лучше, при котором время выполнения задачи меньше.

Впоследствии было разработано несколько более сложных (и точных) вариантов этого метода, но самым распространенным всё равно является изначальный, называемый Keystroke-level Model (KLM). К сожалению, этот вариант метода имеет определенные недостатки (что, впрочем, уравновешивается его простотой):

- он применим в основном для предсказания действий опытных пользователей;
- он никак не учитывает ни прогресса в обучении, ни возможных ошибок, ни степени удовлетворения пользователей;
- он плохо применим при проектировании сайтов из-за непредсказуемого времени реакции системы.

Для его использования достаточно знать правила разбиения задачи на составляющие и длительность каждой составляющей

Правила GOMS:

- Нажатие на клавишу клавиатуры, включая Alt, Ctrl и Shift (К): 0,28 сек
- Нажатие на кнопку мыши (М): 0,1 сек
- Перемещение курсора мыши (П): 1,1 сек (разумеется, время, затрачиваемое на перемещение курсора, зависит как от дистанции, так и тот размера цели. Тем не менее, это число представляет достаточно точный компромисс).
- Взятие или бросание мыши (В): 0,4 сек
- Продолжительность выбора действия (Д): 1,2 сек. (В среднем, за 1.2 секунды пользователь принимает решение, какое именно действие он должен совершить на следующем шаге. Обычно это самый сложный оператор, поскольку часто непонятно, в каких именно местах процедуры его необходимо ставить. Например, иногда, когда пользователь совершал искомую последовательность действий не раз и при этом совершенно уверен в том, что общий ход процедуры не будет отличаться от обычного, это время затрачивается только в самом начале выполнения (далее действия будут совершаться автоматически). С другой стороны, начинающим пользователям приходится выбирать действие перед каждым своим шагом. Однако в большинстве случаев достаточно считать, что это время нужно добавлять перед всеми нажатиями, которые не приходятся на область с установленным фокусом, перед всеми командами, инициированными мышью и после существенных изменений изображения на экране (но и здравый смысл тут не помешает). С практической точки зрения важнее устанавливать этот оператор везде одинаково, нежели устанавливать его возможно более точно).
- Время реакции системы (Р): от 0,1 сек до бесконечности. (Для базовых операций, таких как работа с меню, это время можно не засчитывать, поскольку с момента создания метода производительность компьютеров многократно возросла.

Адаптивная функциональность

Помимо общей логики работы, в системе должна быть ещё одна логика, упрощающая первую и делающую работу пользователя более простой и естественной. Эту логику называют адаптивной функциональностью.

Возьмем пульт от телевизора. Телевизор выключается только одной кнопкой на пульте, но включается от нажатия любой кнопки. Это не следует напрямую из логики системы, но это естественно. Когда на этаж приезжает лифт с неавтоматическими дверями, дверь можно открыть ещё

до того, как погаснет кнопка вызова (чтобы лифт не увели). Это не вполне логично, но естественно. Другой известный, но не всеми осознаваемый, пример: когда Windows при входе в систему спрашивает пароль, нужно нажать Ctrl+Alt+Delete. В этом же диалоговом окне есть кнопка Справка, нажатие на которую открывает ещё одно диалоговое окно, повествующее о том, как нажать эти три клавиши. Так вот, чтобы войти в систему, это окно не нужно закрывать, нажать Ctrl+Alt+Delete можно по-прежнему. С системной точки зрения это неправильно (почему пользователь не закрыл сначала окно с подсказкой?), но для пользователей это естественно.

Все три примера демонстрируют готовность системы (а точнее, её разработчиков) усложнить свою логику, чтобы упростить логику пользователя. Результат: систему легче использовать. Наличие адаптивной функциональности служит отличным индикатором качества дизайна системы. Систему, которая не подстраивается под пользователей, невозможно назвать зрелой.

Остается один вопрос: как определить, какие фрагменты и функции системы должны быть адаптивными? Ответ: единственным решением является детальный анализ взаимодействия пользователей с системой. Помочь здесь может только тестирование интерфейса на пользователях.

Создание глоссария

Еще в процессе проектирования полезно зафиксировать все используемые в системе понятия. Для этого нужно просмотреть все созданные экраны и выписать из них все уникальные понятия (например, текст с кнопок, названия элементов меню и окон, названия режимов и т.д.). После этого к получившемуся списку нужно добавить определения всех концепций системы (например, книга или изображение).

Теперь этот список нужно улучшить. Для этого:

- Уменьшите длину всех получившихся элементов.
- Покажите этот список любому потенциальному пользователю системы и спросите его, как он понимает каждый элемент. Если текст какого-то элемента воспринимается неправильно, его нужно заменить.
- Уменьшите длину всех получившихся элементов.
- Проверьте, что одно и то же понятие не называется в разных местах по-разному.
- Проверьте текст на совпадение стиля с официальным для выбранной платформы (если вы делаете программу, эталоном является текст из MS Windows).
- Убедитесь, что на всех командных кнопках стоят глаголы-инфинитивы.

После чего список стараться не менять его в будущем.

Сбор полной схемы

К этому моменту вы обладаете:

- общей схемой системы
- планами отдельных экранов
- глоссарием.

Пора свести всё это воедино. Работа эта довольно скучная и утомительная, но и от неё есть существенная польза. Рисовать такую схему гораздо легче, чем делать прототип, множество же ошибок можно выловить и в ней, не переделывая прототипа. Рисовать схему очень удобно в уже упоминавшейся Visio. Результат должен выглядеть примерно так:

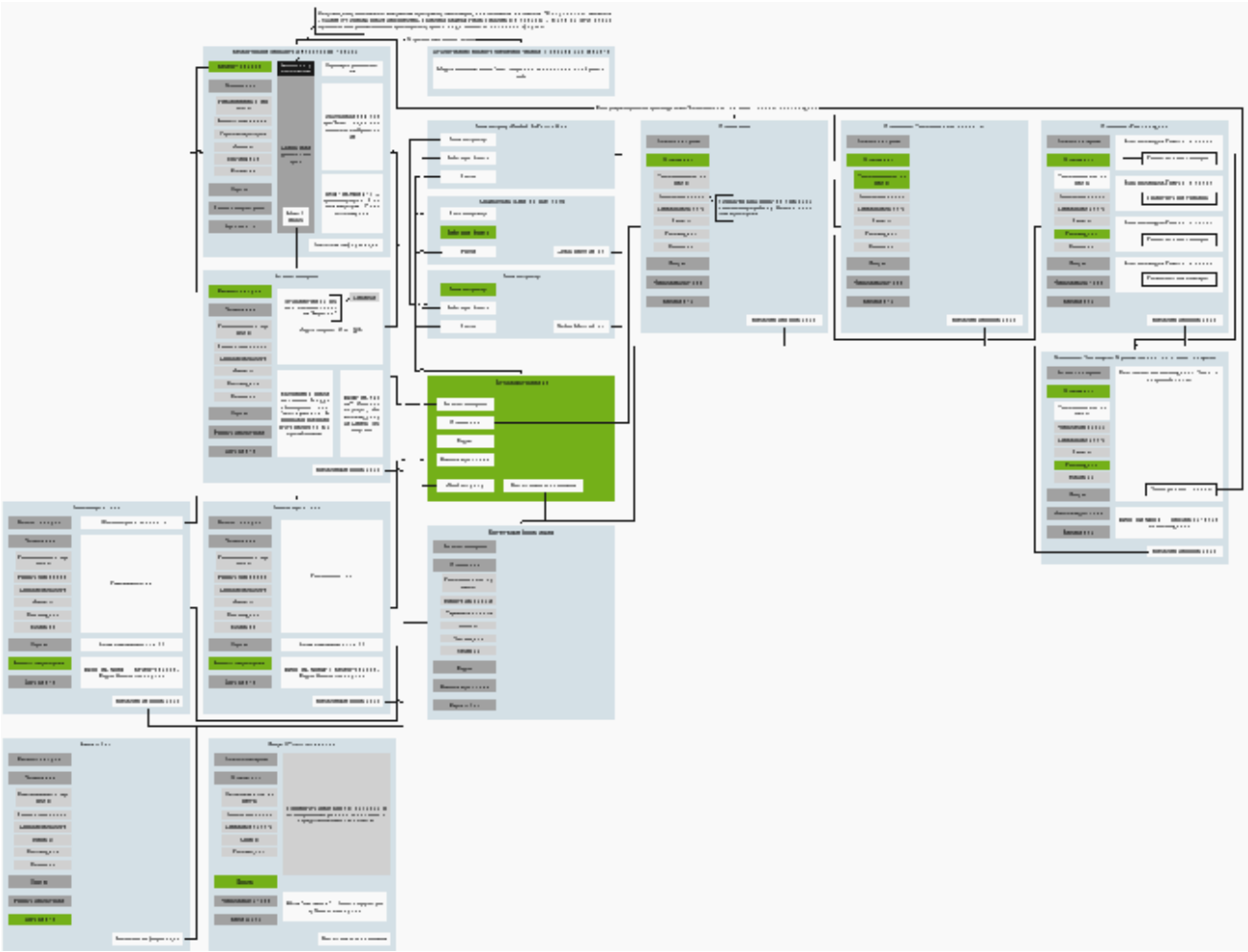


Рис. 3. Пример готовой схемы интерфейса сайта.

Если серьезно, то самой важной целью этого этапа является создание плана обработки системой исключительных ситуаций интерфейса. Необходимо определить, что делать системе, если пользователь вызвал команду, которую для этого конкретного объекта

выполнить невозможно (например, пользователь пытается послать письмо человеку, почтовый адрес которого системе неизвестен).

Проверка схемы по сценарию

Последней задачей перед построением прототипа является проверка внутренней логики системы. Дело в том, что всегда существует вероятность того, что вы что-то забыли или спланировали неправильно. Как уже было сказано, исправить эти ошибки лучше всего до построения прототипа (даже первой его версии). Конечно, многие структурные ошибки нельзя найти никакими методами, кроме длительного логического анализа. С другой стороны, практика показывает, что почти все найденные ошибки будут существенными. Так что лишняя проверка не повредит.

Для финальной проверки схемы вам пригодятся разработанные вами пользовательские сценарии. Не глядя на схему, необходимо подробно описать, как все вымышленные пользователи будут взаимодействовать с системой, не пропуская ни одного элемента управления. После чего сверить полученный текст со схемой.

Тут возможно три варианта развития событий: либо вы обнаружите, что вы что-то забыли задокументировать в схеме, либо обнаружите, что свеженаписанный рассказ значительно лучше схемы, вероятнее же всего, что и то и другое произойдет одновременно. На четвертый вариант, а именно на полное отсутствие проблем, рассчитывать, как правило, не стоит.

Экспертная оценка

Весьма эффективным средством оценки получающегося интерфейса является его экспертная оценка. Часто оказывается, что сравнительно дорогое тестирование показывает то, что было бы легко видно постороннему, тем более вооруженному опытом и квалификацией, взгляду. Хотя экспертная оценка не может быть полноценной заменой тестирования, она обладает одним существенным преимуществом – для её проведения не требуется прототип. Это значит, что эксперт может быть приглашен на ранних стадиях работы, когда польза от обнаружения ошибок максимальна.

Для проведения экспертной оценки нужно знать следующее:

- Разные люди обнаруживают разные ошибки. Это значит, что метод работает лучше, когда количество экспертов больше единицы.
- Лучше привлекать несколько экспертов не одновременно, но последовательно.
- Чем больше информации о проектируемой системе будет предоставлено эксперту, тем более сложные проблемы он сможет выявить.

- Нельзя требовать от эксперта работы по весу. В большинстве случаев результатом его работы будут одна или две страницы текста (поскольку описание одной проблемы требует обычно всего двух или трех предложений). Если от эксперта будет требоваться объемный результат работы, он включит в него много несущественных подробностей.

3. Построение прототипа

Итак, первый этап пройден. У вас есть полная схема, описывающая всё взаимодействие пользователя с системой. Настало время делать прототип системы для тестирования. При создании прототипа наиболее частой ошибкой является чрезмерное наведение глянца и вообще стремление сделать прототип возможно более похожим на результирующую систему. В самом таком подходе нет ничего плохого (всё равно определенные части прототипа приходится делать максимально совершенными), проблема в том, что в большинстве случаев прототип после тестирования оказывается неправильным. Его приходится переделывать, причем иногда полностью, при этом все инвестированные в прототип ресурсы оказываются выброшенными на ветер.

Не полируйте прототип

Поэтому всегда правильно делать прототип настолько похожим на результирующую систему, насколько версия прототипа поздняя. Первый прототип стоит делать максимально примитивным. Только после того, как тестирование подтверждает его правильность, стоит делать более детализированный прототип.

Итак, как быстрее и дешевле построить прототип?

Первая версия. Бумажная

Необходимо нарисовать на бумаге все экраны и диалоговые окна (распечатать соответствующие части схемы). Нужно только убедиться, что все интерфейсные элементы выглядят единообразно и сколько-нибудь похоже на реальные. Эта распечатка и является первым прототипом. На нём вполне можно тестировать восприятие системы пользователем и её основную логику. Польза начального прототипирования на бумаге заключается, во-первых, в исключительной простоте модификации по результатам тестирования, а во-вторых, в возможности безболезненно отлавливать представителей целевой аудитории. Разумеется, значение слова «версия», весьма условно. В действительности после обнаружения каждой ошибки схема и прототип исправляются, а тестирование продолжается уже на новом прототипе. Так что на этом этапе прототип может пережить множество исправлений и, соответственно, много версий.

Вторая версия. Презентация

После исчерпания возможностей бумажной версии прототипа стоит создать новую версию (исправив, разумеется, уже обнаруженные проблемы). Для этого точно так же рисуется интерфейс, но уже не на бумаге, но в какой-либо презентационной программе (MS PowerPoint, например). При этом каждый экран получает отдельный слайд, а результат нажатия кнопок имитируется переходами между.

С этой версией прототипа можно тестировать значительно более сложное взаимодействие человека с системой, нежели с бумажной. С другой стороны, исправление найденных ошибок значительно более трудоемко.

Фактически для большинства систем этой версии оказывается достаточно.

Третья версия

В тех случаях, когда в интерфейсе появляются нестандартные элементы или необходимо проверить реальную скорость взаимодействия пользователя с системой, создается еще одна версия прототипа – реально выглядящая, но лишенная каких-либо алгоритмов и, соответственно, не показывающая реальных данных. Делать этот вариант можно как в средах разработки, благо в них есть визуальные инструменты создания интерфейсов, так и в редакторах изображений, что обычно быстрее.

Фактически при этом создаются фальшивые снимки экрана, на которых и производят тестирование. Понятно, что существенно модифицировать эти экраны затруднительно, так что лучше не увлекаться такой работой, не получив каких-либо гарантий ее правильности.

Четвертая версия

Иногда необходимо тестировать взаимодействие пользователя не только с интерфейсом системы, но и с обрабатываемыми системой данными. Например, работая с графической программой, пользователь не только нажимает на экранные кнопки, но также создает и модифицирует изображения мышью. Область же редактирования данных зачастую вообще не содержит каких-либо визуальных интерфейсных элементов, из чего вовсе не следует, что интерфейса в ней нет, его, наоборот, много. Другой разговор, что счет в нем идет не на кнопки и переключатели, но на пиксели и миллисекунды.

Понятно, что создание прототипа в таких условиях не поможет, поскольку прототип вообще не будет отличаться от проектируемой системы. В таких условиях лучше всего убедить программистов написать нужные участки кода до написания всего остального, и проводить тестирование уже на реальной системе.

4. Тестирование и модификация прототипа

Какими бы не были совершенными логические соображения, приведшие к созданию интерфейса, всегда остается вероятность того, что интерфейс получился плохой, либо, что более вероятно, не такой хороший, каким бы он мог быть. Необходимо иметь какие-либо подтверждения его работоспособности. К счастью, проверка качества интерфейса обычно непроблематична. Всё, что для этого нужно, это несколько пользователей средней квалификации, никогда не видевшие тестируемой системы, плюс прототип (разумеется, при наличии основательного бюджета можно развернуться и пошире, например, купить прибор, фиксирующий направление взгляда пользователя).

В литературе часто встречается мнение, что тестированием можно решить чуть ли не все проблемы интерфейса. Утверждение это сомнительно. Тестированием, скорее, можно определить слабые места интерфейса, но почти невозможно обнаружить сильные, поскольку они пользователями просто не замечаются, и совсем уж невозможно определить новые способы улучшения. Происходит это из-за того, что субъекты тестирования:

- не обладают всей необходимой информацией о системе,
- ничего не знают о проектировании интерфейсов,
- их мотивация существенно отличается от необходимой – вместо того, чтобы стремиться сделать хороший интерфейс, они стремятся оставить в этом интерфейсе свой след.

Вообще, слушать потребителей обычно неправильно. Разве мы спрашиваем канарейку, в какой клетке она хочет жить? Сюжет проамериканский автопром, например, стал уже частью истории бизнеса: все американские потребители в семидесятых годах дружно утверждали, что они хотят большие, мощные машины, при этом так же дружно покупая маленькие и маломощные японские автомобили. Или другой пример – в советское время измученные коммунизмом люди мечтали вовсе не об отпуске на Тенерифе, о котором они ничего не знали, но о финском хромированном смесителе, который поставил себе сосед – хотя Тенериф, безусловно, в качестве мечты интереснее.

В то же время, даже не слушая пользователей, обязательно нужно принимать во внимание их потребности, способности и предпочтения. Например, нередко дизайнер интерфейса знает о предметной области меньше, нежели будущие пользователи. В таких условиях потеря контакта с пользователями грозит крахом продукта, просто потому, что система оказывается неспособна решать задачи, о которых дизайнер ничего не знал. Чаще, однако, случается так, что уровень «компьютерной

грамотности» дизайнера оказывается выше уровня аудитории. В этом случае все получается ещё хуже: дизайнер выбирает решения, которые обеспечивают эффективность работы, а потребителям нужны решения, которые они могут понять, в результате они оказываются неспособными воспользоваться системой (это совершенно нормальная ситуация, особенно в интернете). Например, замечено, что опытные пользователи (к которым относятся дизайнеры) создают значительно менее работоспособные иерархии меню, нежели пользователи начинающие.

Разумеется, если переоценка способностей реальных пользователей и незнание предметной области совпадают, результат бывает ещё хуже.

Постановка задачи

Одной из самых важных предпосылок успешного тестирования является правильная постановка задачи. Всегда есть шансы потратить несколько часов в поисках ответа на ненужный вопрос. Хуже того – случается, что после окончания длительного и утомительного сеанса приходит понимание того, что тех же результатов можно было бы добиться с меньшими трудозатратами. Правильная постановка задачи позволяет этих проблем избежать.

Иногда имеющийся вопрос можно переформулировать таким образом, чтобы он сам по себе вел к ответу. Почти всегда – чтобы метод ответа на него обходился дешевле. Не надо так же забывать убеждаться, что задаваемый вопрос действительно нужен.

Например, нужно определить, как пользователи видят какое-либо диалоговое окно. Можно нацепить на тестера уже упомянутый прибор для определения направления взгляда, а потом долго определять, куда тестер смотрел. Можно найти неопытного пользователя, который помогает себе мышью (многие пользователи постоянно перемещают курсор мыши в то место, куда они смотрят). А можно переформулировать вопрос и поступить совсем иначе (но об этом позже).

Собственно тестирование

Технически сеанс тестирования довольно прост. Нужно иметь несколько пользователей, которые ни разу не видели текущего состояния системы. За исключением редких случаев, когда ваша система рассчитана на продвинутых пользователей (power user), нужно подбирать не слишком опытных субъектов. Тестерам дается задание, они его выполняют, после чего результаты анализируются. Идея проста, тем не менее, по этому поводу написано довольно много литературы (причем объемной). Впрочем, в большинстве случаев достаточно помнить следующее:

- Тестирование на одном пользователе позволяет найти примерно 60% ошибок. Соответственно решайте сами, сколько пользователей необходимо для одного сеанса.
- Если у вас есть возможность оставить тестера одного, не пренебрегайте этим. Одностороннее зеркало в таких условиях не роскошь.
- Никогда не прерывайте пользователя. Никогда не извиняйтесь за несовершенство тестируемой системы. Никогда не говорите «Мы потом это исправим». Никого не обвиняйте. Никогда не называйте процесс тестирования «пользовательским тестированием» – пользователь решит, что тестируют его, и будет бояться.

Проверка посредством наблюдения за пользователем

Один из самых простых видов тестирования. Пользователю дается задание, он его выполняет, его действия фиксируются для дальнейшего анализа какой-либо программой записи состояния экрана.

Чтобы пользователь не тревожился и не стеснялся, его лучше всего оставить в одиночестве. Метод исключительно полезен для выявления неоднозначности элементов интерфейса. Поскольку каждая неоднозначность приводит к пользовательской ошибке, а каждая такая ошибка фиксируется, обнаружить их при просмотре записанного материала очень легко.

Этот тест замечательно подходит для поиска проблем интерфейса. Кроме того, если замерять время выполнения задания (секундомером), можно оценить производительность работы пользователей. Этот же метод позволяет посчитать количество человеческих ошибок.

Мыслим вслух

Метод довольно нестабильный, но порой дающий интересные результаты (очень зависит от разговорчивости пользователя). Соответствует проверке посредством наблюдения за пользователем, но тестера при этом просят также устно комментировать свои действия. Затем комментарии анализируются.

Метод позволяет легко определить недостатки реализации конкретных интерфейсных идей (неудачно расположение элементов, плохая навигация). Обратите внимание, что субъект, проговаривающий свои впечатления, работает медленней обычного, так что измерять скорость работы этим методом невозможно.

Проверка качества восприятия

Тест позволяет определить, насколько легко интерфейсу обучиться. Поскольку существует разница между понятиями *видеть* и *смотреть*, а запоминается только то, что увидено, необходимо обладать уверенностью в том, что пользователь увидит если не всё, то уж хотя бы всё

необходимое. А значит – запомнит, благодаря чему в будущем ему не придется сканировать меню в поисках «чего-то такого, что, я точно знаю, где-то здесь есть».

Сама по себе методика проста. Пользователю даётся задание, связанное с каким-либо отдельным диалоговым окном. Пользователь его выполняет. Через несколько минут пользователя просят нарисовать (пускай даже грубо и некрасиво) только что виденное им окно. После чего рисунок сравнивается с оригиналом.

Разумеется, пользователь запоминает только то, что ему кажется актуальным в процессе работы с окном (плюс еще что-нибудь за того, что ему показалось интересным, да и то не всегда). Это один из тех редких случаев, когда срабатывает ограничение на объем кратковременной памяти, так что количество запомнившихся элементов управления не может быть выше порога. Например, пользователь, которому нужно сменить шрифт абзаца на Arial из всего диалогового окна выбора шрифта в MS Word запоминает только три элемента управления (разумеется, он помнит, что помимо них были и другие, но точно вспомнить остальные элементы он, как правило, не может).

Как это ни грустно, основное предназначение этого теста состоит в том, чтобы раз за разом убеждаться в том, что запомнить нужное совершенно невозможно. Но и в таком качестве он полезен.

Модификация

Тестирование само по себе имеет существенный недостаток: если тестирование проблем не выявило, получается, что оно было проведено зря. Если выявило, придется проблемы решать, что тоже существенная работа. Таким образом, сама идея тестирования интерфейса создает конфликт интересов у дизайнера – работы от него прибавляется либо много, либо очень много – но всегда прибавляется. Работать же, разумеется, не хочется.

Именно поэтому тестирование бессознательно переносят на самое окончание проекта, когда что-либо исправлять уже поздно. В результате тестирование показывает, что проект сделан плохо, что никому не нравится, включая его создателя, после чего результаты проверки прячутся в дальний ящик.

В то время как сама по себе идея тестирования совсем иная. В самом начале работы, когда только создан прототип будущей системы, он тестируется, после чего найденные ошибки исправляются. А затем прототип тестируется опять. При этом опытность дизайнера проявляется исключительно в уменьшении количества итераций. Соответственно, тестирование должно идти параллельно со всеми остальными операциями.

5. Проектирование интерфейса пользователя АСУ объекта

Современные методы проектирования деятельности пользователей АСУ сложились в рамках системотехнической концепции проектирования, в силу чего учет человеческого фактора ограничился решением проблем согласования «входов» и «выходов» человека и машины. Вместе с тем при анализе неудовлетворенности пользователей АСУ удается выявить, что она часто объясняется отсутствием единого, комплексного подхода к проектированию систем взаимодействия, представляемого как комплексное, взаимосвязанное, пропорциональное рассмотрение всех факторов, путей и методов решения сложной многофакторной и многовариантной задачи проектирования интерфейса взаимодействия. Имеются в виду функциональные, психологические, социальные и даже эстетические факторы.

В настоящее время можно считать доказанным, что главная задача проектирования интерфейса пользователя заключается не в том, чтобы рационально «вписать» человека в контур управления, а в том, чтобы, исходя из задач управления объектом, разработать систему взаимодействия двух равноправных партнеров (человек-оператор и аппаратно-программный комплекс АСУ), рационально управляющих объектом управления. Человек-оператор является замыкающим звеном системы управления, т.е. **субъектом управления.** АПК (аппаратно-программный комплекс) АСУ является **инструментальным средством реализации** его (оператора) управленческой (оперативной) деятельности, т.е. **объектом управления**. По определению В.Ф.Венды, АСУ представляет собой гибридный интеллект, в котором оперативный (управленческий) состав и АПК АСУ являются равноправными партнерами при решении сложных задач управления. Интерфейс взаимодействия человека с техническими средствами АСУ может быть структурно изображен (см. на рис.1.).

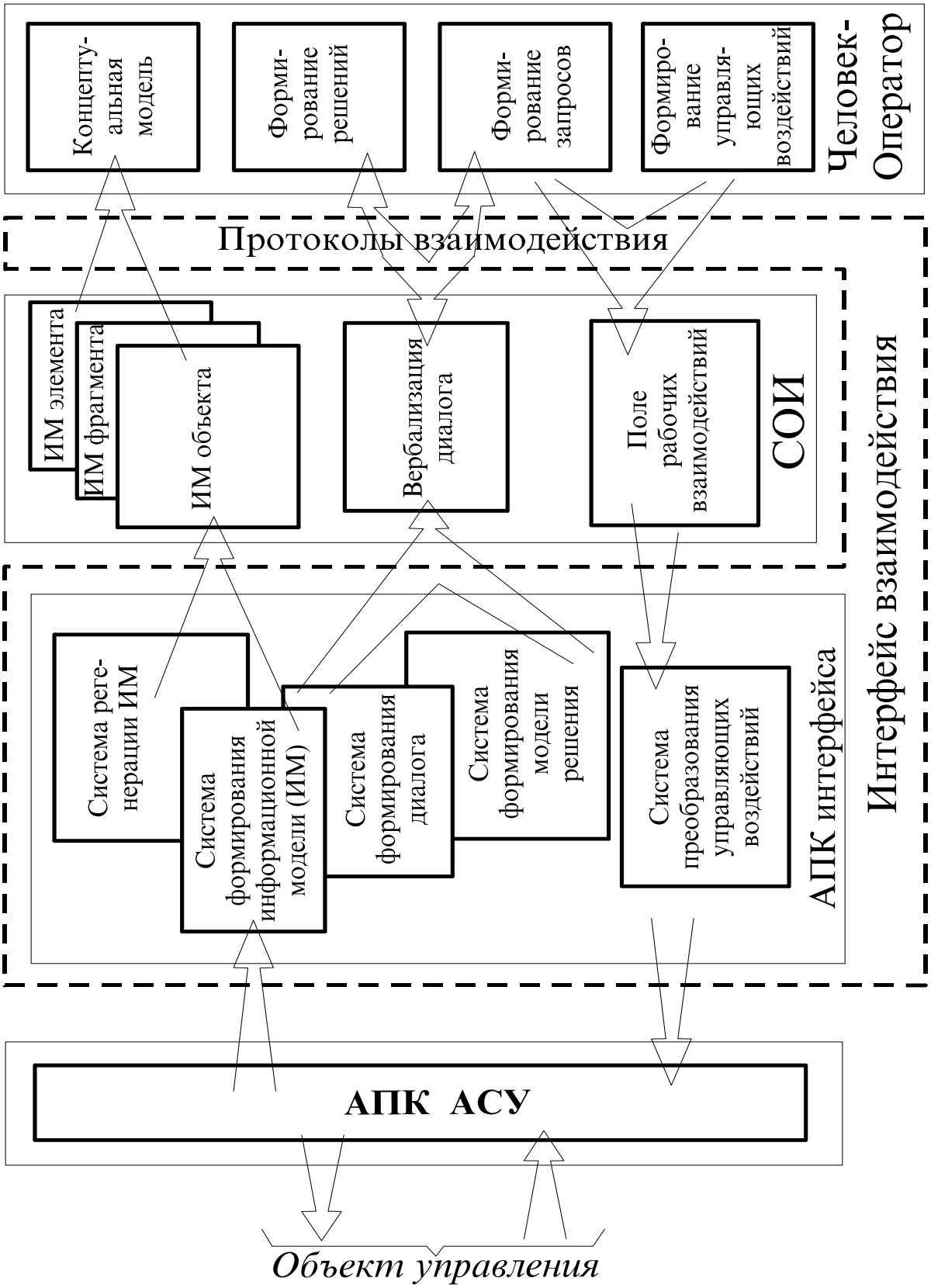


Рис. 1. Информационно-логическая схема интерфейса взаимодействия

Рациональная организация труда операторов АСУ является одним из важнейших факторов, определяющих эффективное функционирование системы в целом. В подавляющем большинстве случаев управленческий труд - опосредованная деятельность человека, поскольку в условиях АСУ он ведет управление, «не видя» реального объекта. Между реальным

объектом управления и человеком-оператором находится **информационная модель объекта** (средства отображения информации). Поэтому возникает проблема проектирования не только средств отображения информации, но и средств взаимодействия человека-оператора с техническими средствами АСУ, т.е. проблема проектирования системы, которую нам следует назвать **интерфейс пользователя.**

Он состоит из АПК и протоколов взаимодействия. Аппаратно-программный комплекс обеспечивает выполнение функций:

- преобразование данных, циркулирующих в АПК АСУ, в информационные модели, отображаемые на мониторах (СОИ - средства отображения информации);
- регенерация информационных моделей (ИМ);
- обеспечение диалогового взаимодействия человека с ТС АСУ;
- преобразование воздействий, поступающих от ЧО (человека-оператора), в данные, используемые системой управления;
- физическая реализация протоколов взаимодействия (согласование форматов данных, контроль ошибок и т.п.).

Назначение протоколов состоит в том, чтобы обеспечить механизм достоверной и надежной доставки сообщений между человеком-оператором и СОИ, а следовательно, между ЧО и системой управления. **Протокол** - это правило, определяющее взаимодействие, набор процедур обмена информацией между параллельно выполняемыми процессами в реальном масштабе времени. Эти процессы (функционирование АПК АСУ и оперативная деятельность субъекта управления) характеризуются, во-первых, отсутствием фиксированных временных соотношений между наступлением событий и, во-вторых, отсутствием взаимозависимости между событиями и действиями при их наступлении.

Функции протокола связаны с обменом сообщениями между этими процессами. Формат, содержание этих сообщений образуют логические характеристики протокола. Правила же выполнения процедур определяют те действия, которые выполняют процессы, совместно участвующие в реализации протокола. Набор этих правил является процедурной характеристикой протокола. Используя эти понятия, мы можем теперь формально определить протокол как совокупность логических и процедурных характеристик механизма связи между процессами. Логическое определение составляет синтаксис, а процедурное - семантику протокола.

Генерирование изображения с помощью АПК позволяет получать не только двумерные спроецированные на плоскость изображения, но и реализовать картинную трехмерную графику с использованием

плоскостей и поверхностей второго порядка с передачей текстуры поверхности изображения.

При создании сложных АСУ велико значение разработки программного обеспечения, т.к. именно программные средства создают интеллект компьютера, решающий сложные научные задачи, управляющий сложнейшими технологическими процессами. В настоящее время при создании подобных систем значительно возрастает роль человеческого фактора а, следовательно, эргономического обеспечения системы. Основной задачей эргономического обеспечения является оптимизация взаимодействия между человеком и машиной не только в период эксплуатации, но и при изготовлении, и при утилизации технических компонентов. Таким образом, при систематизации подхода проектирования интерфейса пользователя, можно привести некоторые основные функциональные задачи и принципы построения, которые должна решать система.

Принцип минимального рабочего усилия разработчика ПО и пользователя, имеющий два аспекта:

- минимизация затрат ресурсов со стороны разработчика ПО, что достигается путем создания определенной методики и технологии создания, свойственной обычным производственным процессам;
- минимизация затрат ресурсов со стороны пользователя, т.е. ЧО должен выполнять только ту работу, которая необходима и не может быть выполнена системой, не должно быть повторений уже сделанной работы и т.д.

Задача максимального взаимопонимания пользователя и АПК в лице разработчика ПО. Т.е. ЧО не должен заниматься, например, поиском информации, или выдаваемая на видеоконтрольное устройство информация не должна требовать перекодировки или дополнительной интерпретации пользователем.

Пользователь должен **запоминать как можно меньшее количество информации**, так как это снижает свойство ЧО принимать оперативные решения.

Принцип максимальной концентрации пользователя на решаемой задаче и локализация сообщений об ошибках.

Принцип учета профессиональных навыков человека-оператора. Это значит, что при разработке системы на основе некоторых задаваемых в техническом задании исходных данных о возможном контингенте кандидатов проектируется «человеческий компонент» с учетом требований и особенностей всей системы и её подсистем. Формирование же концептуальной модели взаимодействия человека и технических средств АСУ означает осознание и овладение алгоритмами

функционирования подсистемы «человек - техническое средство» и овладение профессиональными навыками взаимодействия с ЭВМ.

Ключ для создания **эффективного интерфейса** заключается **в быстром**, насколько это возможно, **представлении оператором простой концептуальной модели интерфейса**. Общий Пользовательский доступ осуществляет это через согласованность. Концепция согласованности состоит в том, что при работе с компьютером у пользователя формируется система ожидания одинаковых реакций на одинаковые действия, что постоянно подкрепляет пользовательскую модель интерфейса. Согласованность, обеспечивая диалог между компьютером и человеком-оператором, может снизить количество времени, требуемого пользователем как для того, чтобы изучить интерфейс, так и для того чтобы использовать его для выполнения работы.

Согласованность является свойством интерфейса по усилению пользовательских представлений. Другой составляющей интерфейса является **свойство его конкретности и наглядности.** Это осуществляется применением плана панели, использованием цветов и другой выразительной техники. Идеи и концепции затем обретают физическое выражение на экране, с которым непосредственно общается пользователь.

На практике высокоуровневое проектирование пользовательского интерфейса предваряет первоначальное проектирование, которое позволяет выявить требуемую функциональность создаваемого приложения, а также особенности его потенциальных пользователей. Указанные сведения можно получить, анализируя техническое задание на автоматизируемую систему управления (АСУ) и руководство по эксплуатации (РЭ) на объект управления, а также информацию, поступающую от пользователей. С этой целью производят опрос потенциальных операторов и операторов, работающих на неавтоматизированном объекте управления.

После определения целей и задач, стоящих перед ними, переходят к следующему этапу проектирования. Этот этап связан с составлением пользовательских сценариев. Сценарий - это описание действий, выполняемых пользователем в рамках решения конкретной задачи на пути достижения его цели. Очевидно, что достигнуть некоторой цели можно, решая ряд задач. Каждую их них пользователь может решать несколькими способами, следовательно, должно быть сформировано несколько сценариев. Чем больше их будет, тем ниже вероятность того, что некоторые ключевые объекты и операции будут упущены.

В то же время, у разработчика имеется информация, необходимая для формализации функциональности приложения. А после

формирования сценариев становится известным перечень отдельных функций. В приложении функция представлена функциональным блоком с соответствующей экранной формой (формами). Возможно, что несколько функций объединяются в один функциональный блок. Таким образом, на этом этапе устанавливается необходимое число экранных форм. Важно определить навигационные взаимосвязи функциональных блоков. На практике установлено наиболее подходящим число связей для одного блока равное трем. Иногда, когда последовательность выполнения функций жестко определена, между соответствующими функциональными блоками можно установить процессуальную связь. В этом случае их экранные формы вызываются последовательно одна из другой. Такие случаи имеют место не всегда, поэтому навигационные связи формируются либо, исходя из логики обработки данных, с которыми работает приложение, либо основываясь на представлениях пользователей (карточная сортировка). Навигационные связи между отдельными функциональными блоками отображаются на схеме навигационной системы. Возможности навигации в приложении передаются через различные навигационные элементы.

Основным навигационным элементом приложения является главное меню. Роль главного меню велика еще и потому, что оно осуществляет диалоговое взаимодействие в системе «пользователь-приложение». Кроме того, меню косвенно выполняет функцию обучения пользователя работе с приложением.

Формирование меню начинается с анализа функций приложения. Для этого в рамках каждой из них выделяют отдельные элементы: операции, выполняемые пользователями, и объекты, над которыми осуществляются эти операции. Следовательно, известно какие функциональные блоки должны позволять пользователю осуществлять, какие операции, над какими объектами. Выделение операций и объектов удобно проводить на основе пользовательских сценариев и функционала приложения. Выделенные элементы группируются в общие разделы главного меню. Группировка отдельных элементов происходит в соответствии с представлениями об их логической связи. Таким образом, **главное меню может иметь каскадные меню**, выпадающие при выборе какого либо раздела. **Каскадное меню ставит в соответствие первичному разделу список подразделов.**

Одним из требований к меню является их стандартизация, целью которой выступает формирование устойчивой пользовательской модели работы с приложением. Существуют требования, выдвигаемые с позиций стандартизации, которые касаются места размещения заголовков разделов, содержания разделов часто используемых в разных

приложениях, формы заголовков, организации каскадных меню и др. *Наиболее общие рекомендации стандартизации следующие:*

– *группы функционально связанных разделов отделяют разделителями (черта или пустое место);*

– *не используют в названиях разделов фраз (желательно не больше 2 слов);*

– *названия разделов начинают с заглавной буквы;*

– *названия разделов меню, связанных с вызовом диалоговых окон заканчивают многоточием;*

– *названия разделов меню, к которым относятся каскадные меню, заканчивают стрелкой;*

– *используют клавиши быстрого доступа к отдельным разделам меню. Их выделяют подчеркиванием;*

– *допускают использовать «горячие клавиши», соответствующие комбинации клавиш отображают в заголовках разделов меню;*

– *допускают использовать включение в меню пиктограмм;*

– *измененным цветом показывают недоступность некоторых разделов меню в ходе работы с приложением;*

– *допускают делать недоступные разделы невидимыми.*

Недоступность некоторых разделов меню обуславливается следующим. Главное меню является статическим и присутствует на экране в течение всего времени работы с приложением. Таким образом, при работе с разными экранными формами (взаимодействии с разными функциональными блоками) не все разделы меню имеют смысл. Такие разделы принято являются недоступными. Поэтому в зависимости от контекста решаемых пользователем задач (иногда от контекста самого пользователя) главное меню приложения выглядит различным образом. О подобных различающихся внешних представлениях меню принято говорить как о различных состояниях меню. В отличие от схемы навигационной системы, составленной ранее и необходимой, в основном, разработчику, с меню пользователь входит в непосредственное взаимодействие. Меню определяет количество окон и их разновидность. Весь интерфейс сопровождается окнами предупреждений, окнами подсказок, окнами мастеров, задающих последовательность действий пользователей при выполнении некоторых необходимых операций.

6. Пример выполнения

В качестве примера рассматриваемой АСУ, на которую будет проектироваться интерфейс пользователя принята автоматизированная система управления и контроля вакуумного выключателя.(АСУКВВ).

Система предназначена для автоматизированной проверки и регулировки параметров работы механизмов выключателя, а также для автоматизации процесса приемо-сдаточных, типовых и периодических испытаний.

Система осуществляет следующие функции:

- снятие временных и скоростных характеристик;
- снятие аналоговых сигналов тока и напряжения;
- преобразование временных, скоростных и аналоговых сигналов в цифровое отображение и вывод их на экран монитора.
- вывод на принтер и запись на магнитный диск осциллограмм и протоколов испытаний;

Описание объекта управления и его ментальной модели

В качестве объекта управления принят вакуумный выключатель.

На рис. 2 изображено фото выключателя. На рис. 3 изображен вид выключателя сбоку в двухмерном изображении.

Рис. 2 Выключатель вакуумный.

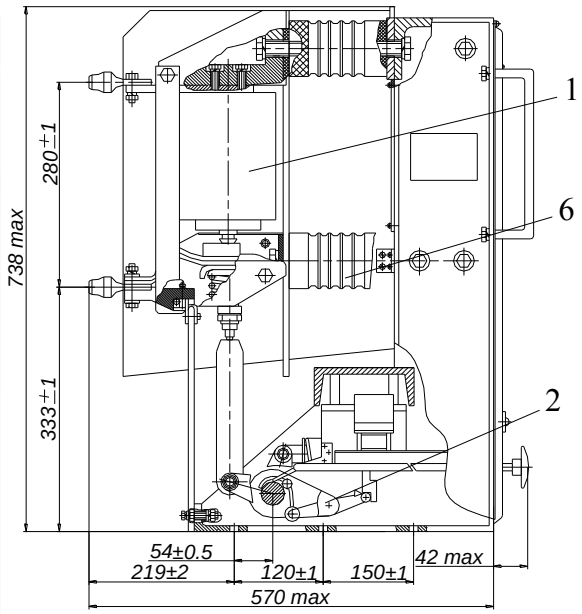


Рис. 3 Вид выключателя сбоку.

На рис. 4 и 5 представлена ментальная модель выключателя в режимах «Включено» и «Отключено». На модели показаны измеряемые и вычисляемые параметры контактов и электромагнитов включения (YAC) и отключения (YAT) выключателя.

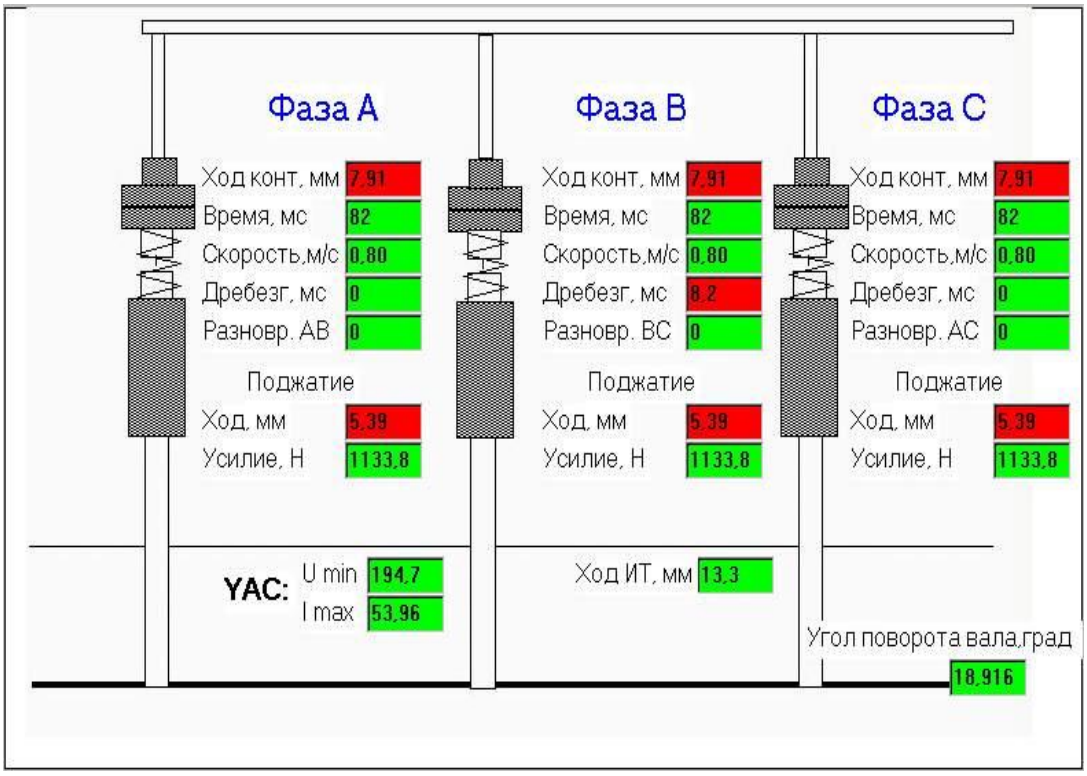


Рис.4 Ментальная модель выключателя в режиме «Включено»

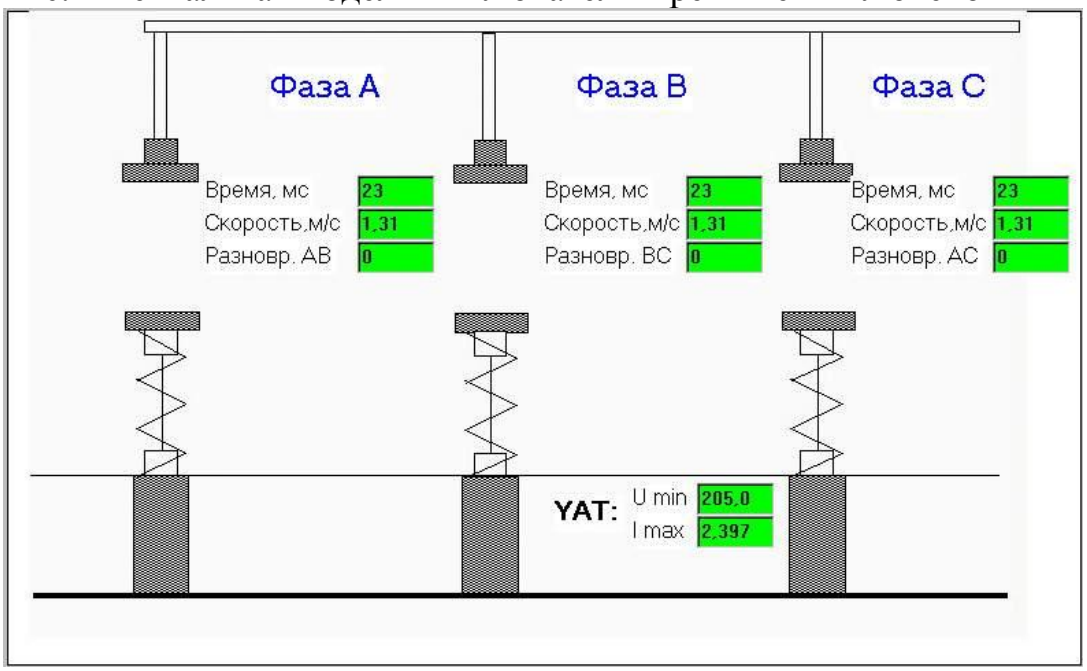


Рис. 5 Ментальная модель выключателя в режиме «Отключено»

Система АСУКВВ должна обеспечивать в процессе выполнения циклов «включения – отключения» автоматический контроль следующих параметров, характеризующих функциональное состояние, настройку и регулировку механизма выключателя

- ход подвижных контактов ВДК;
- полный ход изоляционных тяг выключателя;
- собственное время включения;
- собственное время отключения;
- полное время отключения;

- напряжение питания электромагнитов управления YAC и YAT;
- токи потребления обмоток электромагнитов управления YAC и YAT с выделением их максимальных значений;
- средние скорости движения подвижного контакта ВДК при включении и отключении на заданных временных интервалах;
- выбег хода вала выключателя при отключении;
- суммарное время дребезга контактов при включении;

Перечисленные параметры показаны на ментальных моделях

3.3. Система автоматически отрабатывает следующие типы операций:

- **«В»** включение;
- **«О»** отключение;
- **«В» -t -«О»** **(**включение-пауза-отключение)

Функциональная схема АСУКВВ

Функциональная схема АСУКВВ изображена на рисунке 6.

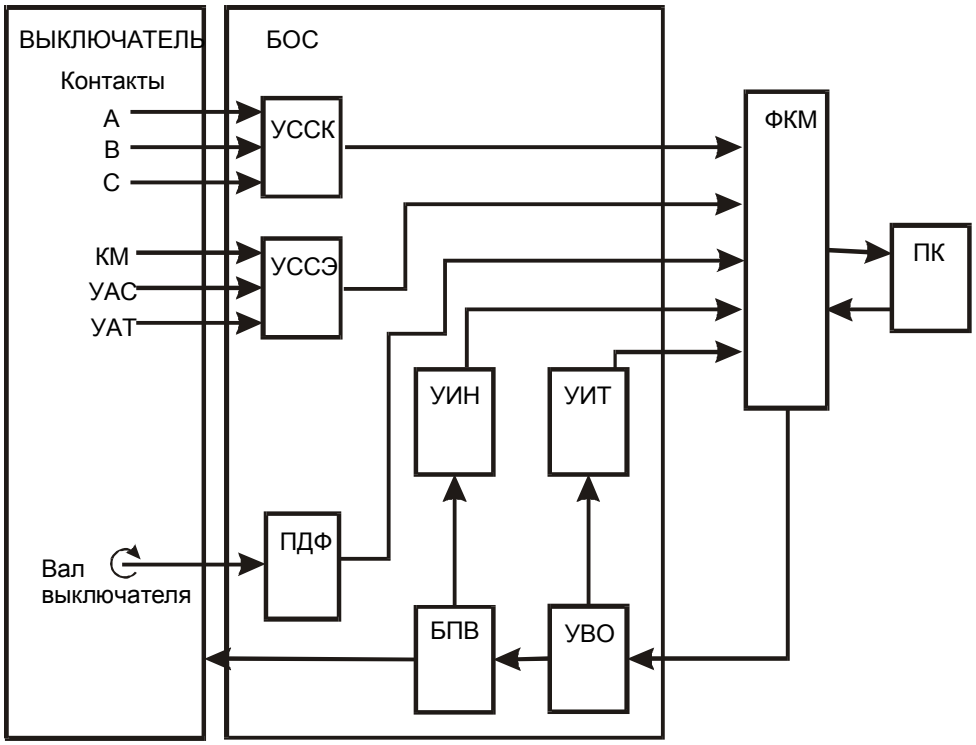


Рис.6 Функциональная схема АСУКВВ

УССК – устройство съема сигналов контактов; УССЭ – устройство съема сигналов электромагнитов; ПДФ – преобразователь (датчик) угла поворота вала фотооптического типа; УИН – устройство измерения напряжения, УИТ – устройство измерения тока (датчики тока); УВО – устройство включения и отключения; БПВ – блок питания выключателя; КМ – коммутатор подачи питания на электромагнит включения

Работа системы происходит следующим образом. Команда на включение выключателя поступает от персонального компьютера (ПК) через блок обработки сигналов (БОС) на контактор КМ. Контактор срабатывает и через свои силовые контакты пропускает питание U=220В, I=60А на электромагнит включения (YAC) выключателя. Шток электромагнита через рычаг поворачивает вал выключателя, на котором закреплен датчик угла поворота типа ПДФ или энкодер, например, ENA фирмы Bourns, с которого снимается для последующей обработки 600 (1000 или 2500) для ПДФ и 128 (256) для ENA импульсов на оборот. Вал через изоляционные тяги приводит в движение подвижные контакты фаз А, В, и С. Момент замыкания которых снимается и передается для отсчета угла положения вала и для последующего вычисления хода контактов, собственного времени включения выключателя и скорости движения контактов. Выключатель включается. В процессе включения выключателя снимается также информация о напряжении и токе в электромагните включения (YAC). Вся информация с датчиков и узлов съема сигналов поступает на ФКМ, который обрабатывает ее и направляет в ПК. Компьютер вычисляет и формирует контролируемые параметры выключателя в цифровом и графическом видах. Полученная информация выводится на экран монитора, а также может быть по желанию оператора выведена на принтер и магнитный диск. Команда на отключение выключателя проходит тем же путем от ПК через ФКМ на электромагнит отключения выключателя (YAT). Напряжение и ток электромагнита включения снимается и подается для дальнейшей обработки. В процессе отключения также автоматически производится снятие заданных временных, скоростных характеристик, линейных перемещений и других параметров. Работа системы может происходить как в ручном режиме подачи команды на включение (отключение), так и в автоматическом режиме выполнения заданного числа циклов «включение – пауза – отключение».

Программа позволяет осуществлять связь с функционально - конструктивным модулем, тестировать его, обрабатывать результаты, полученные от ФКМ, управлять процессом включения-отключения выключателя, тестировать система в целом, проводить наладочные работы системы, анализировать и накапливать информацию, полученную в результате испытаний отдельного выключателя. В процессе работы состояние ФКМ постоянно контролируется и при обнаружении неисправности будет выдано оператору соответствующее сообщение.

Интерфейс программы полностью совместим с интерфейсом Windows9x и 200x, интуитивно понятен, легко осваивается

начинающими пользователями и позволяет в кратчайшее время приступить к работе даже операторам с низкой квалификацией.

В основном рабочем окне программы, показанном на рисунке 7 размещаются следующие элементы:

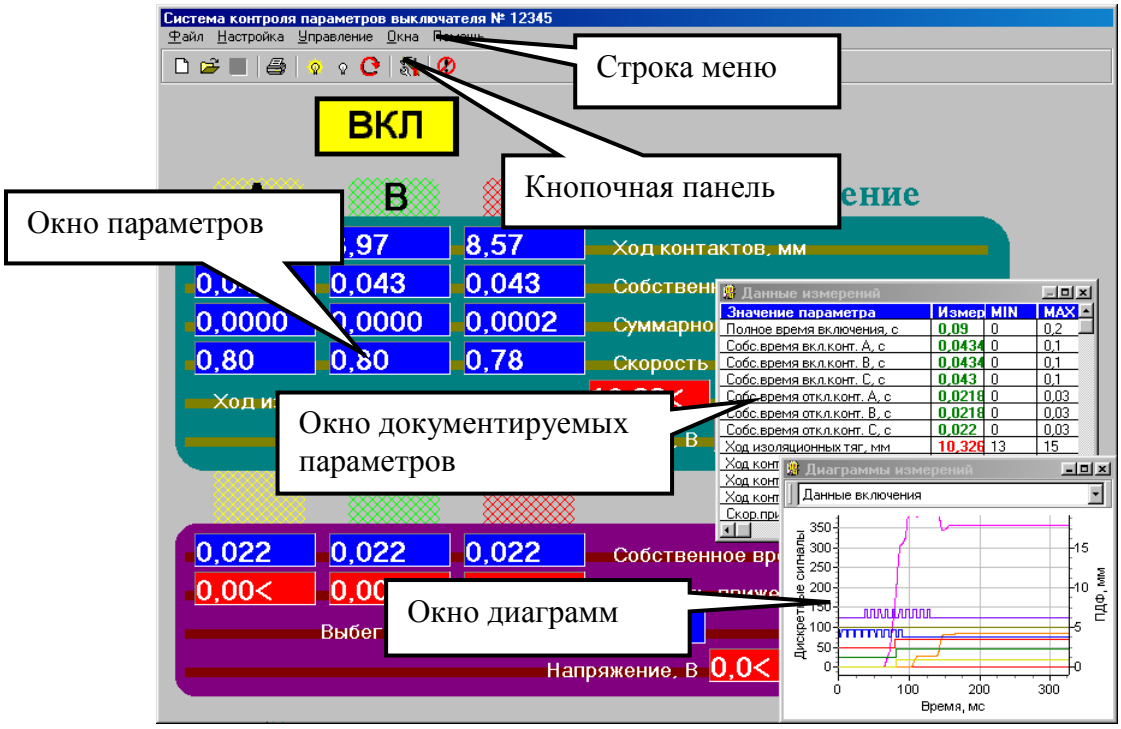


Рис. 7 Основное рабочее окно

- СТРОКА МЕНЮ, открывающего доступ ко всем командам;
- КНОПОЧНАЯ ПАНЕЛЬ - для быстрого доступа к часто используемым командам;
- ОКНО ПАРАМЕТРОВ, требующих регулировки - информация, помогающая оператору производить настройку выключателя;

Дополнительно открываются:

- ОКНО ДИАГРАММ - основные характеристики выключателя, представленные в виде графиков, изменяющихся во времени и облегчающие понимание процессов ВО;
- ОКНО ДОКУМЕНТИРУЕМЫХ ПАРАМЕТРОВ - основные характеристики выключателя, снятые в последнем цикле измерений;
- ОКНА СООБЩЕНИЙ, появляющиеся при необходимости принятия оператором решений или сообщающие ему о ходе процесса измерения, облегчают работу оператора.
- ОКНО ТЕСТИРОВАНИЯ показывает состояние датчиков сигналов, поступающих с блока съема и преобразования информации (БОС) и работу ФКМ.

- ОКНО НАСТРОЙКИ И КАЛИБРОВКИ позволяет изменять константы, используемые для управления выключателем и в процессе обработки полученных данных.

В ходе работы программы информация о проведенных циклах ВО заносится в РАБОЧИЙ ЖУРНАЛ для данного выключателя и может быть сохранена на дисках в виде файла. Анализ и документирование снятых данных могут быть проведены непосредственно после проведенных циклов ВО, либо, в дальнейшем ЖУРНАЛ может быть прочитан и информация из него проанализирована и выведена на печать в виде документа.

Главное меню программы

Описываем пункты главного меню и пункты соответствующих им выпадающих меню и выполняемые ими действия:

Пункт меню	***Пункт выпадающего меню***	***Действия***
Файл Операции с файлами.	Новый журнал	Установить константы для регистрации нового выключателя.
	Открыть журнал…	Открыть файл с ранее сохраненным РАБОЧИМ ЖУРНАЛОМ.
	Сохранить журнал…	Сохранить данные измерений в РАБОЧИЙ ЖУРНАЛ.
	Печать...	Вывести на принтер документируемые данные.
	Выход	Завершить работу программы.
Настройка Изменение основных параметров системы	Предельные параметры	Изменение предельно допустимых значений параметров.
	Настройка и калибровка	Изменение калибровочных констант и значений датчиков и переменных среды.
	Инициализация ФКМ	Выдача сигнала инициализации (сброса) на плату ФКМ.
	Тест ФКМ	Тестирование ФКМ и БОС
Управление Основные операции по управлению Выключателем	Включение	Выдача команды на включение выключателя
	Отключение	Выдача команды на отключение выключателя
	Цикл	Проведение цикла В-О-t с заданным параметром t и числом циклов.
Окна Манипуляции с окнами.	Данные измерений	Вывести/убрать окно с документируемыми параметрами
	Диаграммы	Вывести/убрать окно с диаграммами

Интерфейс калибровки датчика тока и датчика напряжения

Выбираем пункт меню ***Настройка - Настройка и калибровка***. В этом пункте разрабатываем целый ряд диалоговых окон, в которых должны быть предусмотрены закладки для всех видов настройки и калибровки. Одной из закладок делаем ***Ток и напряжение***.

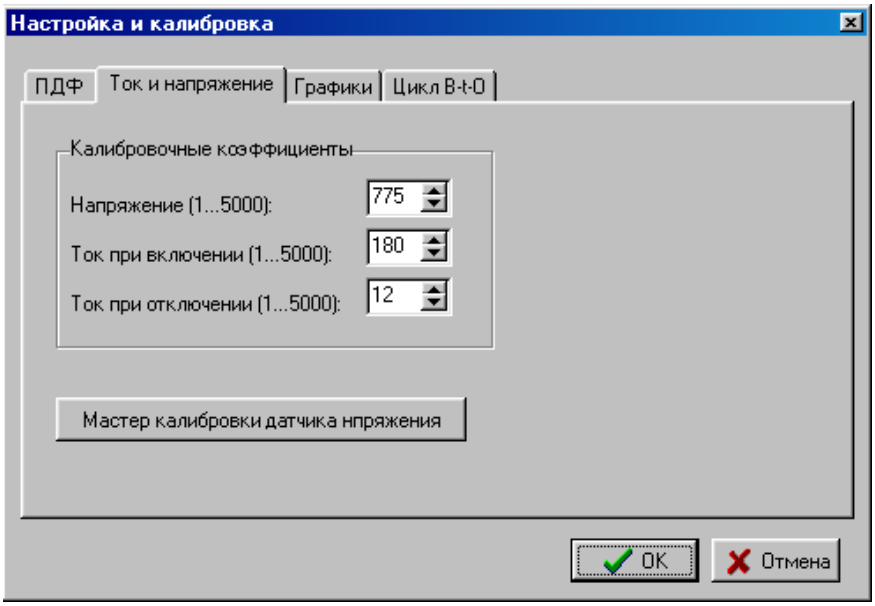


Рис. 8 Диалоговое окно калибровки датчиков тока и напряжения

В диалоговом окне «Настройка и калибровка» с закладкой «Ток и напряжение» предусматриваем поле для ввода калибровочных коэффициентов для напряжения с окном и «крутилкой» для ввода значений, токов включения и токов отключения с аналогичным окном. После надписей «Напряжение», «Ток включения» и «Ток отключения целесообразно указать диапазон изменения соответствующих параметров.

Ниже поля ввода калибровочных коэффициентов расположим кнопку «Мастер калибровки датчика напряжения» и самом низу окна кнопки управления «ОК» и «Отмена»

Для проведения калибровки датчика напряжения на обмотках электромагнитов управления выключателя необходимо убедиться, что выключатель отключен, напряжение на систему подано. Значение напряжения должно быть замерено заранее измерительным прибором. Далее проводятся следующие действия:

При нажатии на кнопку ***Мастер калибровки U*** должно открыться диалоговое окно «Мастер калибровки датчика напряжения» с закладками «Шаг 1», «Шаг 2» и «Шаг 3» Разместим в центре окна окно с цифровым отображением напряжения и полосой прокрутки, расположенное под окном

Внизу диалогового окна располагаем командные кнопки «Назад», «Далее», «Отмена»

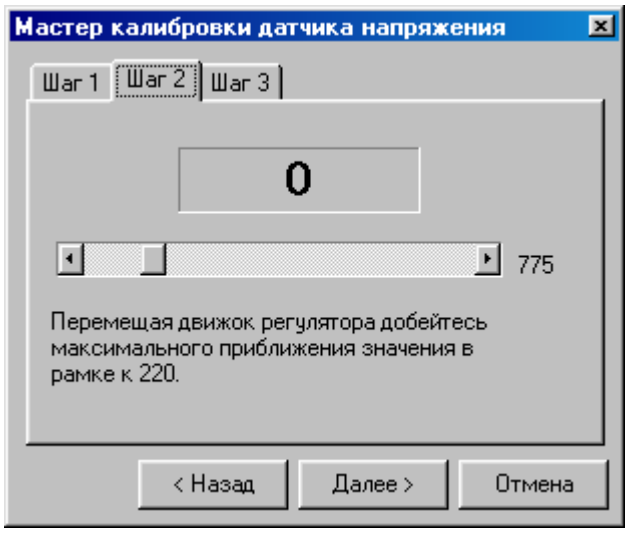


Рис.9 Окно мастера калибровки датчика напряжения

Для закладки «Графики» диалоговое окно должно показывать, какие параметры должны быть выведены в виде графиков.

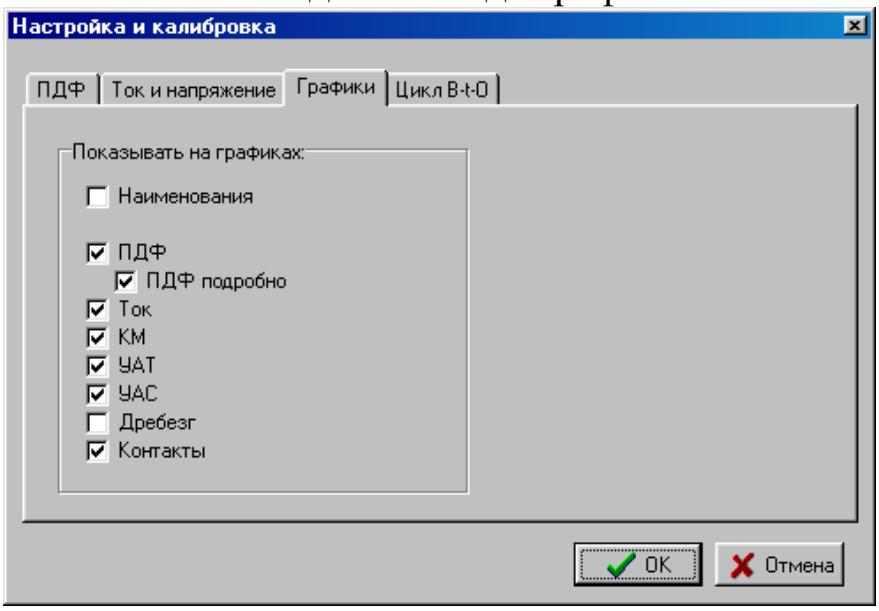


Рис. 10 Окно настройки изображения графиков

Здесь необходимо установить флажок ***Наименования*** для отображения «легенды» на графике, показывающей цвет и наименование кривых на диаграмме сигналов. Установите флажки напротив названий кривых, которые необходимо вывести на диаграмме.

При выборе закладки ***Цикл B-t.-O*** Необходимо диалоговое окно следующего типа

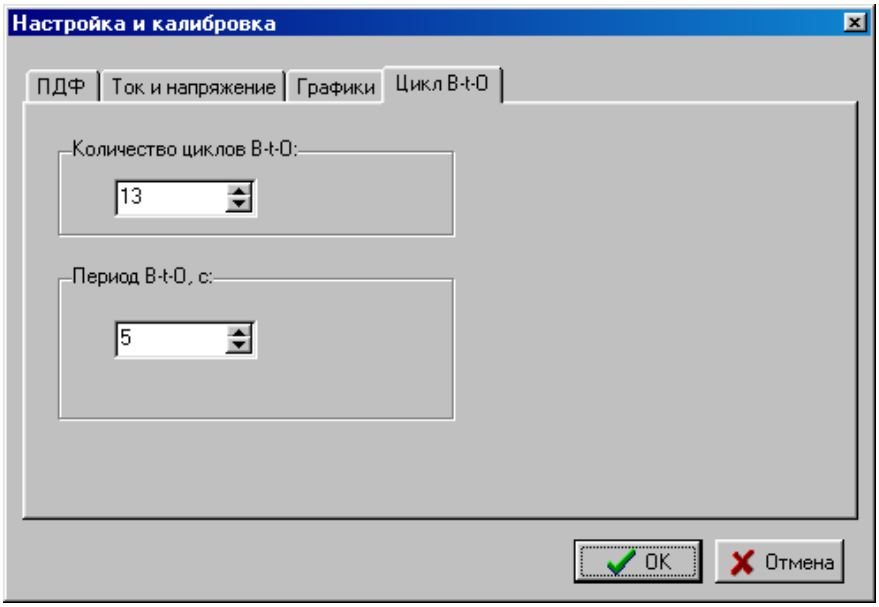


Рис. 11 Диалоговое окно установки параметров циклов

В этом окне необходимо установить значения количества циклов и пауз при работе в цикле B-t –O и нажать кнопку ***Ок***.

Интерфейс настройка значений предельных параметров

Настройка предельных значений параметров производится при выборе пункта меню ***Настройка - Предельные параметры***. При этом открывается окно ***Предельные значения*** со сводной таблицей всех документируемых данных.

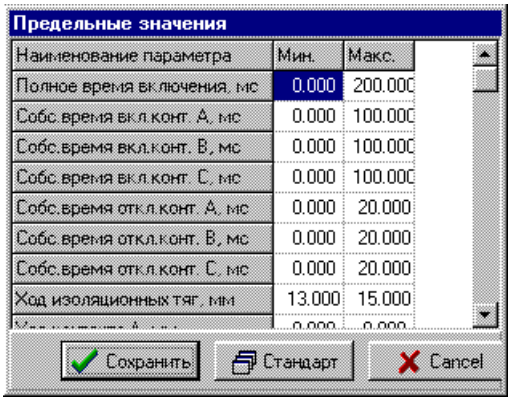


Рис. 12 Окно установки предельных значений параметров

Двойное нажатие мыши на соответствующей ячейке приводит к возможности редактировать эти значения. Кнопка ***Стандарт*** возвращает все значения к фиксированным по умолчанию. Кнопка ***Сохранить*** осуществляет сохранение предельных параметров в постоянной памяти ПК и их загрузку при следующем запуске программы. Кнопка ***Cancel*** отменяет работу с окном ***Предельные значения***.

Снятие характеристик включения

Для снятия характеристик выключателя необходимо открыть новый ЖУРНАЛ ИЗМЕРЕНИЙ, выбрав пункт меню ***Файл - Новый журнал и*** заполнить необходимую информацию в окне ***Новый журнал*** и нажать кнопку ***Ок***. Далее необходимо подготовить выключатель к включению и выбрать пункт меню ***Управление - Включение*** или кнопку ***Включить выключатель***. Произойдет автоматическое включение выключателя и снятие его характеристик.

При успешном выполнении операции включения данные будут записаны в памяти ПК и выведены на дисплей в окне параметров, окне документируемых параметров либо на графиках по требованию оператора.

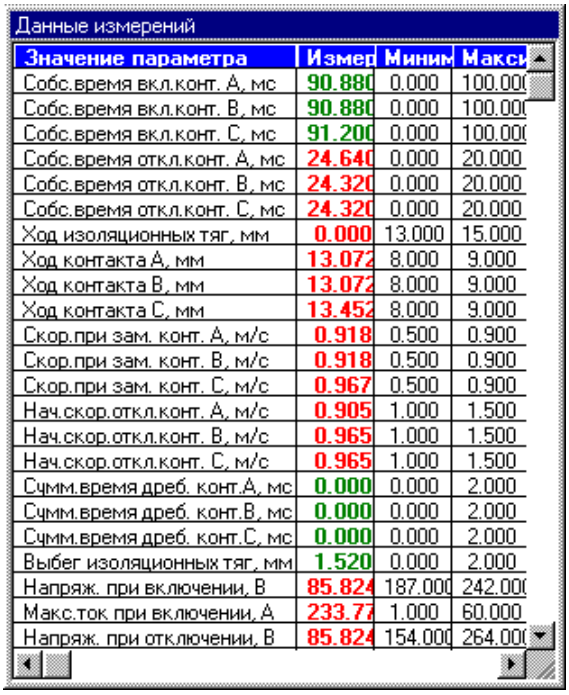
Окно параметров является основным информационным окном для оператора.

Значения в окне параметров, укладывающихся в норму, показываются синим или зеленым цветом, в противном случае - красным, направление отклонения величины от допустимого диапазона значений указывается знаком «больше» или «меньше».

При неудачном выполнении операции включения оператору будет выдано сообщение о вероятной причине отказа. Данные в этом случае не регистрируются.

Более подробные сведения о работе выключателя можно получить в окне документируемых параметров. Открыть это окно можно, выбрав пункт меню ***Окна - Данные измерений***. Измеренные значения, укладывающиеся в норму, показаны зеленым цветом, в противном случае - красным. Диапазон требуемых значений указан в столбцах «Миним» и «Максим».

В случае, когда указывается предельные значения параметров, тогда параметры, вышедшие за предельные значения показывают красным цветом, а параметры, вышедшие за пределы максимума и минимума обозначаются желтым цветом.



Значение параметра	Измер	Миним	Макси
Собс.время вкл.конт. А, мс	90.880	0.000	100.000
Собс.время вкл.конт. В, мс	90.880	0.000	100.000
Собс.время вкл.конт. С, мс	91.200	0.000	100.000
Собс.время откл.конт. А, мс	24.640	0.000	20.000
Собс.время откл.конт. В, мс	24.320	0.000	20.000
Собс.время откл.конт. С, мс	24.320	0.000	20.000
Ход изоляционных тяг, мм	0.000	13.000	15.000
Ход контакта А, мм	13.072	8.000	9.000
Ход контакта В, мм	13.072	8.000	9.000
Ход контакта С, мм	13.452	8.000	9.000
Скор.при зам. конт. А, м/с	0.918	0.500	0.900
Скор.при зам. конт. В, м/с	0.918	0.500	0.900
Скор.при зам. конт. С, м/с	0.967	0.500	0.900
Нач.скор.откл.конт. А, м/с	0.905	1.000	1.500
Нач.скор.откл.конт. В, м/с	0.965	1.000	1.500
Нач.скор.откл.конт. С, м/с	0.965	1.000	1.500
Сумм.время дреб. конт.А, мс	0.000	0.000	2.000
Сумм.время дреб. конт.В, мс	0.000	0.000	2.000
Сумм.время дреб. конт.С, мс	0.000	0.000	2.000
Выбег изоляционных тяг, мм	1.520	0.000	2.000
Напряж. при включении, В	85.824	187.000	242.000
Макс.ток при включении, А	233.77	1.000	60.000
Напряж. при отключении, В	85.824	154.000	264.000

Рис 13 Окно данных измерений

Открыть диаграммы включения можно, выбрав пункт меню ***Окна - Диаграммы***. Данные включения или отключения будут показываться, если выбран соответствующий пункт в выпадающем списке вверху окна.

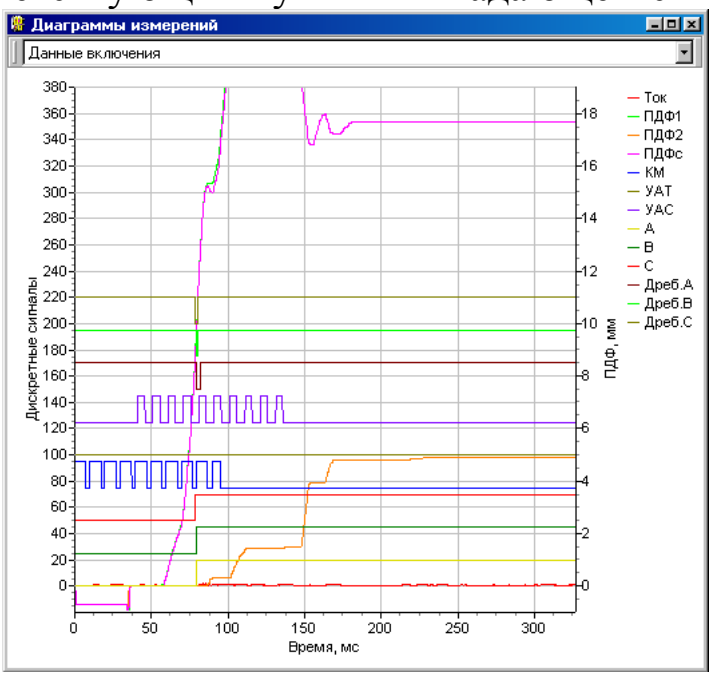


Рис 14 Окно диаграмм измерений

Необходимое расположение окон можно задать, изменяя размер и положение окна мышью, как любого стандартного окна Windows.

Снятие характеристик отключения

Для снятия характеристик отключения необходимо подготовить выключатель к отключению и выбрать пункт меню ***Управление -***

Отключение или кнопку «***Откл»***. Произойдет автоматическое отключение выключателя и снятие его характеристик.

При успешном выполнении операции отключения данные будут записаны в памяти ПК и выведены на дисплей по требованию оператора. При неудачном выполнении операции включения оператору будет выдано сообщение о вероятной причине отказа. Данные в этом случае не регистрируются.

Просмотр информации осуществляется также как и при включении выключателя.

Выполнение цикла В-О

Для выполнения циклов В-О необходимо заранее установить параметры цикла В-О при настройке режимов, выбрав пункт меню ***Настройка -Настройка и калибровка-Цикл В-t -О***).

Откроем, при необходимости, окна диаграмм и документируемых параметров.

Выберем пункт меню ***Управление - Цикл*** или кнопку ***Цикл В-t-О***. Откроем вспомогательное окно управления циклом.

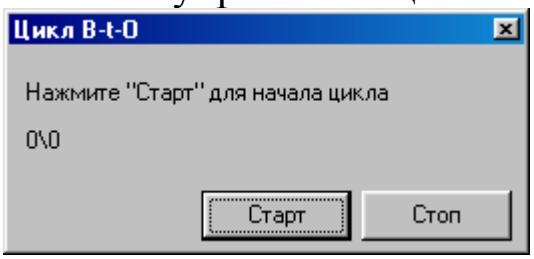


Рис 15 Окно выполнения циклов

Расположим с помощью мыши окно в наиболее подходящем месте экрана так, чтобы не заслонять данные и другие окна. Нажмите кнопку ***Старт*** в окне ***Цикл***. Произойдет включение выключателя и далее автоматическое выполнение цикла В-О-t и снятие его характеристик после каждого цикла. Информация о количестве проведенных циклов, а также дополнительная информация будет отражаться в этом окне в соответствующих строках.

Прервать выполнение цикла можно кнопкой ***Стоп***.

Сохранение данных и работа с ранее произведенными измерениями

Примечание: Корректное сохранение данных и облегчение их обработки может быть получено только при правильно заполненном журнале измерений.

Для сохранения в журнале последнего проведенного измерения выберем пункт меню ***Файл - Сохранить журнал***. В открывшемся

стандартном окне при необходимости укажем папку, где будет сохранен текущий журнал и нажимаем ***Ок***. Сохранение журнала возможно на любом носителе информации: жестком диске, дискете и т.п. Название файла будет совпадать с заводским номером выключателя.

Восстановление ранее сохраненной информации и работа с ней возможна после открытия ранее сохраненного журнала. Для этого выберем пункт меню ***Файл - Открыть журнал***. В открывшемся стандартном окне при необходимости укажем папку, откуда будет восстановлен журнал, и нажимаем ***Ок***. Далее можно анализировать и документировать информацию.

Документирование данных

Примечание: для выполнения этого пункта к ПК должно быть подключено стандартное выводное устройство (принтер, плоттер и т.п.) и установлены необходимые драйверы для него.

Для вывода документируемых данных и графиков работы выключателя на печатающее устройство выбираем пункт меню ***Файл - Печать***. Откроется вспомогательное окно, где необходимо указать вид документа, выводимого на печать.

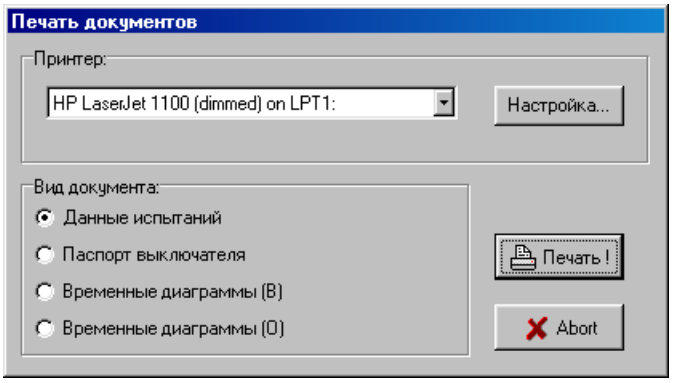


Рис 16 Диалоговое окно печати документов

Здесь же указан тип принтера или другого выводного устройства. В случае необходимости измените его тип или проведем необходимые настройки изображения (например, ориентацию бумаги) нажмаем кнопку ***Настройка***. Нажатие кнопки ***Печать!*** приводит к выдаче программой Диспетчеру печати требуемого документа.
Кнопка ***Abort*** прерывает печать и закрывает окно *Печать документов*.

Завершение работы программы

Для завершения работы с программой необходимо отключить выключатель если он был включен и выбрать пункт меню ***Файл-Выход*** (либо нажать на клавиатуре сочетание клавиш ***Alt+X***, либо закрыть окно программы клавиатурным сочетанием ***Alt+F4***, либо нажать кнопку "Завершить работу программы" на кнопочной панели). При этом

основные установки программы будут автоматически сохранены (и восстановлены при последующем запуске программы) и программа завершит работу. После этого можно выключить питание системы.