# Конструирование программных интерфейсов

Игорь Николаевич Скопин [1]

[1] *Новосибирский государственный университет, НГУ, Новосибирск, 630090, Новосибирская область, г. Новосибирск, ул. Пирогова, 2, Российская Федерация*
[2] *Институт вычислительной математики и математической геофизики Сибирского отделения Российской академии наук ИВМиМГ СЩ РАН 630090, г. Новосибирск, проспект Академика Лаврентьева, 6 Country, Российская Федерация*

**Аннотация**

Изучаются проблемы повышения полезности программных систем, связанные с разработкой интерфейсов. Предлагается методика разработки интерфейсов, преодолевающая эти проблемы за счет предварительного анализа операционных маршрутов пользователей разрабатываемой системы.

**Ключевые слова**

Юзабилити, абстрактный интерфейс, конкретный интерфейс, деятельность, полнота поддержки деятельности, интерфейсные стандарты.

## 1. Введение

Интерфейс программной системы — важная составляющая качества, как основы оценки полезности приложения для пользователя. В то же время, при разработке интерфейсам должное внимание уделяется не всегда. В результате в принципе полезные программы часто оказываются неудобными или даже не востребованными пользователями. Интерфейсные ошибки сплошь и рядом «украшают» популярные программы. Положение усугубляется тем, что реальное использование таких программ создает предпосылки формирования стихийных стандартов, закрепляющих и тиражирующих интерфейсные ошибки.

Сведения о том, какая доля трудозатрат при создании системы приходится на интерфейс, довольно разноречивы. Есть мнение, что в среднем она составляет более половины времени реализации проекта. Проверить это трудно, в частности потому, что продуманный интерфейс обычно требует большего времени, чем непродуманный. Разработка интерфейса, основанного принципах, которые не удовлетворяют пользователя, может усложнять реализацию. Для разных приложений роль интерфейсных характеристик различна, а потому различна трудоемкость интерфейсной части проекта.

В одной из немногих книг [1], посвященных интерфейсам, приводятся рекомендации и предостережения, обоснованные хорошим анализом. Однако автор обсуждает действия пользователя, не связывая интерфейс с функциональностью и архитектурой, что ограничительно. Мы предлагаем восполнить этот пробел, рассматривая интерфейсы в двух аспектах. Во-первых, это абстрактное управление поведением приложения, а во-вторых — отображение абстрактного управления в конкретные интерфейсные формы. Соответственно, вводятся понятия *абстрактного* и *конкретного интерфейсов* и их *полноты*. Полнота абстрактного интерфейса — это соответствие функциональности всем элементам деятельности, автоматизируемой приложением, а полнота конкретного интерфейса — отражение в его элементах всех аспектов абстрактного поведения в формах, отвечающих пользовательской потребности автоматизируемой деятельности.

Ключевым положением предлагаемого подхода является анализ всех видов деятельности пользователя, на которые влияет включение приложения в качестве средства или инструмента.

7th International Workshop on Information, Computation, and Control Systems for Distributed Environments (ICCS-DE 2025), July 7–11, 2025, Irkutsk, Russia
EMAIL: iskopin@gmail.com

ICCS-DE 2025 Workshop Proceedings
DOI: 10.47350/ICCS-DE.2025.15

Цель анализа — определить место разрабатываемого приложения в контексте работы пользователя и, в частности, обосновать необходимые элементы интерфейса и построить типовые сценарии взаимодействия с приложением. Эти сценарии рассматриваются как основа отображения абстрактного управления в конкретный интерфейс. В результате разработчик имеет возможность обосновать использование активных, пассивных и декоративных элементов интерфейса для тех или иных целей.

## 2. Интерфейс как составляющая качества приложения

Обычные критерии качества программного изделия связываются, в первую очередь, с его функциональностью и производительностью. Что же касается интерфейса, то, признавая существенный его вклад в полезность программной разработки равным функциональности и производительности, часто считают, что качество, привносимое интерфейсом, гарантируется использованием при его реализации сложившихся стандартов и хорошо себя зарекомендовавших библиотек поддержки. Причины тому следующие. Если реализацию функций системы можно спланировать и специфицировать на предпроектной стадии и фиксировать результаты этой работы в виде вполне проверяемых требований (во времена советского программирования такие требования предписывалось включать в технические задания), то, за редким исключением, запланировать проверяемые интерфейсные характеристики не удается. Предпроектные требования к интерфейсам сводятся к утверждениям об их соответствии хорошо *зарекомендовавшим* себя образцам, играющим роль стандарта. Чаще всего программные проекты организуются лишь для предоставления новых функций и/или более производительных версий существующих систем. Интерфейсные проекты сводятся к предложению стандартных решений и их библиотечных реализаций, которые отражают довольно произвольное мнение о стандартах элементов интерфейса, но не специфику приложений, для которых применение библиотек оправдано.

Характерными примерами поддержки конструирования интерфейсов могут служить библиотеки, использованные для разработки многочисленных интегрированных сред разработки, так называемых IDE-систем (Integrated development environment см. [3]), предназначенных для разработчиков программного обеспечения. Свое начало все они берут от Turbo Vision [4] и послушно наследуют его средства стандартизованного диалога. Однако в такой среде все приложения становятся похожими друг на друга, их специфика теряется. Безусловно, стандартизация элементов интерфейса нужна, но ее роль должна ограничиваться рамками объективно обусловленных эргономических и когнитивных критериев, учитывать складывающиеся у пользователей привычки и предпочтения. Так же нет сомнения в том, что должны быть выработаны унифицированные правила компоновки элементов и технологически выверенные приемы работы пользователя в интерфейсной среде, подобно тому, как стандартизованы расположение клавиш на пишущей машинке и правила печатания "вслепую". Но унификация не должна вступать в противоречие с требованием обеспечивать максимально полную поддержку конкретной пользовательской деятельности, для которой предназначена программная система.

Такое положение дел нельзя считать удовлетворительным. Большинство интерфейсных решений весьма неоднозначны, во многих случаях оценка качества интерфейса субъективна, о приемлемости или непригодности интерфейса часто удается говорить лишь в целом, без обоснованного указания на четкие критерии.

Потребительская ценность программного продукта, его полезность с точки зрения практического использования, складывается из трех составляющих:

- *функциональность* — способность программы выполнять требуемые от нее функции (набор задач из области приложения, решению которых способствует данное программное изделие, адаптивность для решения новых задач, уровень автоматизации деятельности пользователя при применении программы и другие подобные характеристики);

- *эффективность* — скоростные характеристики и требования памяти, связанные с выполнением программы (они в конечном итоге определяют границы возможного практического использования программы на данном оборудовании);
- *интерфейс* — средства управления работой программы, благодаря которым осуществляется включение ее в человеко-машинную систему.

Эти составляющие качества программного обеспечения не являются независимыми. К примеру, когда программа мало пригодна для практических целей из-за низкой эффективности (о непригодности вообще здесь речи нет, если выполнены функциональные требования), правомерна постановка задачи разработки новой программы, для которой уровень эффективности фиксируется как функциональное требование. Аналогичная ситуация возможна с мало приемлемым интерфейсом.

Функциональная составляющая доминирует в оценке программного изделия: если не выполнены функциональные требования, то говорить об эффективности и интерфейсе не приходится. Функциональные требования естественно рассматривать как регламент, фиксирующий *модель вычислений программы*, а саму функциональность отождествлять с такой моделью. В свою очередь, требования эффективности — это ограничения на допустимость реализации функциональной модели на данном оборудовании, а реализация ее есть *отображение модели вычислений на* реальное *оборудование*, удовлетворяющее требованиям эффективности. Наконец, требования к интерфейсу есть не что иное, как *спецификации языка управления поведением модели*.

Важно отметить, что под языком здесь понимается произвольная знаковая система для передачи информации, включающая не только символьные последовательности, но и визуальные образы (пиктограммы и иные изображаемые знаки), указание картин и их фрагментов с помощью программно-аппаратных средств, задание расцветки изображений, использование меню, транспарантов, табло, обрамлений и др., а также звуковые сигналы и, возможно, речевые средства ввода/вывода. Данный перечень относится к уровню человеко-машинного интерфейса программной системы. Когда речь идет о программах, обеспечивающих доступ к внешнему оборудованию, например, о драйверах, роль языка играют сигналы-сообщения от и для этого оборудования, а также система прерываний. Рассматривая межмодульные интерфейсы, мжно заметить, что для окружения, использующего модуль, предоставляемые им средства являются расширением языка реализации системы. Таким образом, всякий раз, когда речь идет об интерфейсе, нужно иметь в виду тот или иной язык, предоставляемый на уровень использования, и как следствие, если есть возможность выбирать язык, критерием выбора его средств должны быть требования со стороны пользователя.

Выделение модели вычислений, ее отображения как реализации в программе и языка управления поведением модели имеет ряд принципиальных следствий.

1. При проектировании программной системы эти три составляющие разграничивают требования к программному изделию, что создает предпосылки для независимого построения модели, повышения эффективности реализации и разработки интерфейса.

В реальной практике независимость указанных составляющих относительна, поскольку связи между ними существуют на уровне взаимовлияния. Так, можно запланировать вычисления, которые не реализуемы при заданных требованиях к эффективности. Управляемость поведением модели также имеет свои пределы: нельзя говорить об интерфейсе, если не фиксировать управляемые объекты, их возможное поведение и т. п., нельзя строить интерфейс без учета приемлемых для функционирования системы характеристик эффективности. Тем не менее при заданной функциональности отображение модели на реальную операционную среду можно рассматривать как выбор реализации, удовлетворяющей требованиям эффективности, а проектирование интерфейса — как выбор его языковых средств, адекватных использованию программы. Эти две задачи не только можно, но и целесообразно ставить как независимые, в частности по той причине, что для них различны критерии отбора подходящих решений.

2. Явное разграничение трех составляющих качества позволяет ставить задачу разработки серий программ с общей моделью вычислений, предназначенных для различного применения.

Примерами серий такого рода являются системы, учитывающие особенности перерабатываемой информации для повышения эффективности вычислений. Пользователю предоставляется право выбора метода решения задачи, а каждый из методов, если их особенности не проявляются в управлении, — это вариант реализации одной и той же вычислительной модели (если указанное условие не выполняется, т.е. выбор метода является частью управления, то имеет место модель вычислений, включающая с себя вариантные методы, а не серия программ-вариантов). В ряде случаев возможен автоматический выбор метода на основе анализа входных данных, и тогда сама серия для пользователя выступает как единая система.

Другой пример серийности на основе общей модели вычислений — различные программы, использование которых связывается с применением оборудования с несопоставимыми интерфейсными возможностями. Выбор варианта программы таких серий целесообразно возлагать на установку системы.

Серийность, учитывающая особенности перерабатываемых данных, отражает неоднозначность критериев эффективности программ. Они не сводятся лишь к требованию оптимизации — минимизации для конкретных вычислений времени выполнения и/или необходимой памяти. Выбор того или иного алгоритма должен учитывать наряду с особенностями обрабатываемых данных архитектуру вычислительной системы, на которой выполняется программа. Если ставится задача соблюдения оптимальности для разных архитектур — это характерно для большинства вычислительных библиотек — то серийность часто удается обеспечить за счет компиляции, и тогда можно говорить, что пользователю предлагается единая модель вычислений. В противном случае решение о выборе варианта программы должен принимать пользователь, для которого все такие варианты естественно рассматривать как серию. Примером серийности в обоих аспектах может служить библиотека Krylov [5], предназначенная для решения сверхбольших систем линейных уравнений. Алгоритмы решения формально одной и той же задачи специализированы для различных особенностей матриц с целью повышения производительности расчетов. Для заданных параметров задачи в автоматизированном или ручном режиме может быть выбран тот вариант, который дает минимальное время счета.

Еще более неоднозначны интерфейсные критерии, и это обусловливает серии, связанные с вариантами управления. Разные пользователи обладают индивидуальными и групповыми психологическими особенностями восприятия, и, как следствие, для их продуктивной работы требуются различные способы взаимодействия с программной системой. На пользовательские критерии предпочтения влияют и уровень предварительной подготовки, и привычки, и то, что по мере освоения системы требуются различные уровни пояснений, помощи и т.д. В этой связи можно ставить задачу конструирования динамичного интерфейса, т.е. явно и/или неявно настраиваемого на особенности пользовательского предпочтения и восприятия информации.

В качестве наглядной иллюстрации этого положения уместно упомянуть о том, как, к примеру, браузер Chrome [6] или YouTube [7] отбирает и упорядочивает информацию для предъявления пользователю с учетом истории взаимодействия с ним. По понятным причинам первая из этих систем делает это на основе более глубокого анализа.

3. При сравнении программных изделий следует разграничивать сопоставление их функциональности, эффективности и интерфейсов.

На практике это разграничение зачастую игнорируется, что снижает полезность информации для пользовательского выбора подходящего изделия. Если при проектировании функциональность является заданным критерием пригодности будущего изделия, то при сравнении программ ее роль двояка.

Во-первых, функциональные характеристики определяют правомерность использования программы для автоматизации конкретной деятельности. Здесь, скорее, следует сравнивать назначение сопоставляемых программных продуктов, а не такие свойства, как применимость их для массового оперирования (эффективность) и удобство для пользователя (интерфейс).

Во-вторых, сравнение эффективности невозможно без выявления точек пересечения в назначении анализируемых программных изделий, т.е. без выделения их функциональной общности. Этот аспект сравнения достаточно ясен, и обычно эффективность проверяется на общих задачах. Тем не менее, иногда не учитывается, что функциональная общность —

понятие относительное. Так, сравнение эффективности компилятора, который работает параллельно с пользовательским вводом программы, и автономного компилятора, скорее всего, будет в пользу последнего, а простой редактор программ можно сделать более быстрым, чем текстовый процессор. Разная быстродействия здесь не является критерием качества: по внешней схожести применения программ нельзя судить об их функциональности в целом. Ситуация меняется, когда сравнительная эффективность не является критерием качества, — достигнута приемлемая скорость работы программы, т.е. когда с точки зрения полезности применения нет смысла повышать ее эффективность.

Не менее важно разграничение трех составляющих качества для сравнения интерфейсов программ. Проектирование интерфейса в большей степени искусство, чем технология, поэтому для него и критерии качества более субъективны. Но когда сравнение интерфейсов отделено от сопоставления программ по другим составляющим качества, возможна постановка общих вопросов, ответы на которые позволяют сделать оценку интерфейса объективнее.

4. При оценке качества программной системы безотносительно к другим аналогичным изделиям нет смысла говорить о хорошем или плохом продукте, если явно не обозначить деятельность, в которой изделие должно использоваться. Только с этой точки зрения правомерны утверждения о функциональной пригодности системы, об эффективной или неэффективной ее реализации, об удобствах пользовательского управления.

Уместно разграничивать два рода деятельности, для поддержки которых предназначается программа: *автоматизируемая деятельность*, существующая и без применения программных средств, и *новая деятельность*, которая была бы невозможна без оцениваемой программы.

В первом случае программа — один из инструментов деятельности, и ее можно сравнивать с позиции заменяемых или дополняемых неавтоматизированных аналогов. Иными словами, здесь аналогами фиксирована функциональная модель, а о качестве изделия можно судить по тому, насколько эффективнее оказалась выполняемая с помощью программы деятельность по сравнению с неавтоматизированным вариантом. Понятно, что имеется в виду эффективность деятельности, а не программы. как воплощения модели. В частности, эффективность такой деятельности существенно зависит от адекватности управляющих воздействий, т.е. от предлагаемого интерфейса. Можно классифицировать возможные интерфейсные средства по степени их влияния на эффективность деятельности, *подбирать интерфейс* под конкретные условия применения программы, анализируя получаемые результаты.

Во втором случае аналогов программного инструмента нет, а потому критерии качества меняются: оценкой программного изделия становятся результаты новой деятельности, то, как она встраивается в систему других деятельностей. Таким образом, вклад эффективности и интерфейса в оценку качества программы менее весом по сравнению с функциональностью. Тем не менее, если изделие претендует на достаточно долгое существование, вопросы эффективности и интерфейса также должны рассматриваться как существенные. Поэтому вместо подбора интерфейсных средств может быть рекомендована стратегия настройки интерфейса, осуществляемая пользователем: в ходе эксплуатации программной системы, будут выявлены желательные "точки роста" ее эффективности и дополнительные управляющие воздействия. Возможность и целесообразность такого развития программы надо планировать.

## 3. Задача конструирования интерфейса

Развитые технологии программирования предоставляют разнообразные средства реализации моделей вычислений сложных программных систем и достижения требуемой эффективности. Как правило, они исходят из внешних спецификаций конструируемого программного продукта, в которых можно достаточно точно фиксировать, что должно быть сделано, а следовательно, проверка функциональности и определение достигнутого уровня эффективности в принципе не затруднительны.

Иначе обстоит дело с интерфейсом. Когда интерфейсные критерии удается описать явно (например, в случае межмодульного интерфейса), его разработка не требует особых усилий и сводится к поддающимся рационализации согласованиям. Интерфейсные трудности возникают

при конструировании пользовательского представления системы. Главная причина их в том, что разработчики часто слабо представляют себе, кто и как будет применять их изделие. Иногда они даже заявляют примерно следующее: "Вам предоставлена программа с определенным набором средств, а как распорядиться ими — ваша проблема". Это в корне неверное утверждение — забота о технологии применения предлагаемого (программ, оборудования, документации и др.) — важная задача ответственных разработчиков, именно они могут и должны точнее других ответить на вопросы о назначении изделия.

Без учета при разработке "портрета" пользователя, без продумывания регламентов и методов применения программы потребительская ценность приложения снижается. Функционально хорошие программы не всегда в состоянии удовлетворить пользователя с его запросами, обусловленными особенностями его конкретной деятельности. Иногда приемлемый для одних целей интерфейс оказывается неудобным для других из-за смещений между назначением и применением программы (соответствующие примеры удачных и неудачных интерфейсных решений можно найти в [8]).

Проблема разработки интерфейса, соответствующего потребностям пользователей, состоит в том, что заранее определить истинные потребности получается далеко не всегда. Для решения этой проблемы мы предлагаем воспользоваться специальным приемом, отвечающем рассмотрению системы как совокупности активных и/или пассивных *элементов*. В каждый момент элемент находится в одном из заранее определенных *состояний,* характеризующих возможное *поведение элемента*, т.е. его способность выполнять определенные *действия*, в том числе *установку* и *разрыв связей* элементов, а также их создание и удаление элементов и доступность их действий для выполнения. Все это соответствует хорошо известному шаблону проектирования программных систем MVC (Model, View, Controller) [9]. Его основное требование сводится к необходимости при проектировании разграничивать три компонента системы:

- *Модель вычислений* (Model) – определяет и фиксирует требуемую функциональность разрабатываемой системы как множество допустимых последовательностей состояний элементов модели в целом;
- *Представление* (View) – определяет и фиксирует формы задания и предъявления перерабатываемых данных и другой информации элементов системы, необходимую для реализации модели;
- *Управление* (Controller), — определяет и фиксирует действия и воздействия элементов, которые при функционировании модели активизируют фрагменты вычислений в зависимости от поступающих для управления данных и приводят к изменению состояния модели вычислений в целом.

Информация о состоянии элементов доступна для принятия решений о том, какие воздействия нужно осуществить для перевода системы из текущего состояния в желаемое. Она передается как со стороны модели, так от управления в представление для внешнего использования. Взаимодействие пользователя с приложением осуществляется через представление. Из этого следует необходимость учета того, что для разных типов пользователей могут требоваться различные представления.

При разработке представлений необходимо понимать, что каждый пользователь понимает модель и управление приложения по-своему. Это связано с тем, что он встраивает приложение в свою деятельность в качестве инструмента. А поскольку эту деятельность пользователя и связанные с ней другие деятельности предсказать невозможно, не стоит надеяться на то, что пользовательское понимание модели и управления будут совпадать с тем, что предлагает программа приложения. Задача каждого из представлений, предлагаемых разным типам пользователей, состоит в том, чтобы построить соответствие между моделью и управлением приложения и пользователей. Имея ввиду необходимость поддержки этого соответствия при работе приложения как основы его полезности, или, что то же, юзабилити программной системы в целом, *задача конструирования интерфейса разбивается на две части:*

- Разработка *средств взаимодействия системы* с пользователем в виде программных интерфейсов, которые обеспечивают осуществимость выполнения всей

функциональности приложения. Комплект таких средств называется *абстрактным интерфейсом*.

- Разработка визуальных и/или иных *средств предъявления информации* для пользователя и задания воздействий на систему, которые реализуются как обращение к элементам абстрактного интерфейса. Комплект таких средств называется *конкретным интерфейсом*.

Понятия абстрактного и конкретного интерфейсов в точности соответствуют понятиям абстрактного и конкретного синтаксисов из области конструирования компиляторов (см. [10]): и в том, и в другом случае абстрактное является инвариантом множества конкретного, приводящего к одним и тем действиям модели и управления. Аналогия распространяется и на разделение задачи конструирования интерфейса на две части: в архитектуре компиляторов ему соответствуют оперирование семантической информацией и синтаксический анализ.

На рисунке 1 представлена схема, иллюстрирующая задачу конструирования интерфейса, согласованного с шаблоном проектирования MVC. Шаблон естественным образом задает «водораздел» между абстрактным (относящимся к вычислениям) и конкретным (связанным с деятельностью пользователя) уровнями приложения, к которым относятся две части интерфейсной задачи. На схеме они разделены жирной линией. Модель и управление с точки зрения пользователя обозначены заштрихованными блоками U-Model и U-Control, соответственно.

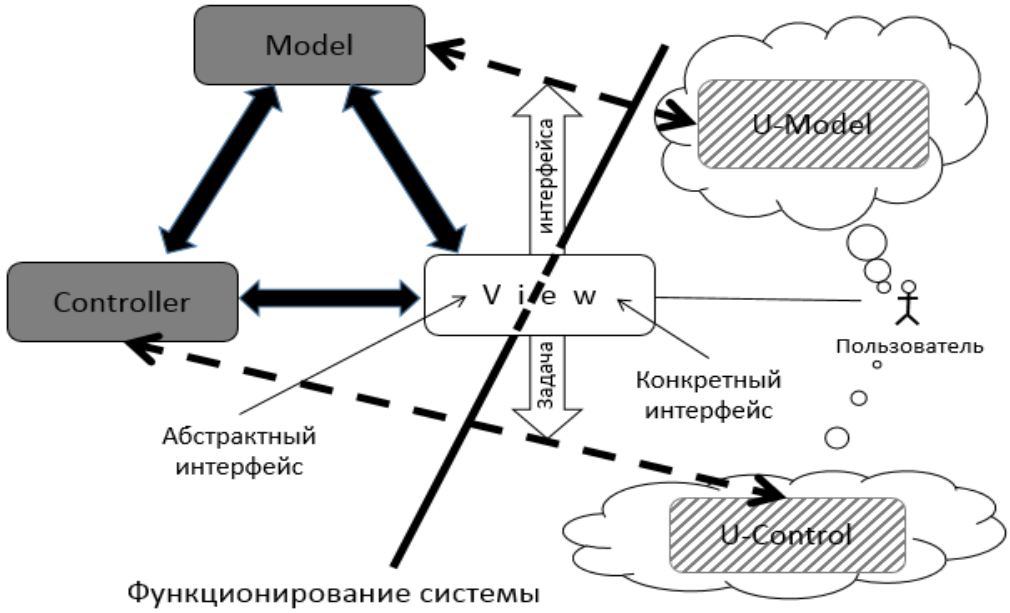


**Рисунок 1**: Модель MVC, абстрактный и конкретный интерфейсы

## 4. Деятельность пользователя и разработка интерфейса приложения

Два аспекта задачи конструирования интерфейса, которые мы только что определили, требуют различных методов решения. Разработка конкретного интерфейса исходит из таких понятий, как дизайн экрана, эргономичность, психологические ограничения, использование стандартов и сложившейся практики оформления взаимодействий с программной системой. При построении абстрактного интерфейса ситуация иная. Здесь главным является обеспечение автоматизируемой приложением деятельности максимально возможной поддержкой. Понятно, что это только потенциальная возможность, которая может быть начисто уничтожена плохим конкретным интерфейсом. Подтверждение тому можно найти на каждом шагу. И это довольно поучительно. Так, примеры, представленные Дж. Раскиным в [1], позволили сформулировать ряд универсальных принципов разработки интерфейсов с учетом когнитивных особенностей

восприятия информации. К сожалению, этот автор обсуждает действия пользователя, не связывая интерфейс с архитектурой, и не учитывая различия типов пользователей. Подобные ошибки, рассуждая об интерфейсах, допускает и Дж. Сполски в своей замечательной серии статей про программное обеспечение [2]. В результате разумные рекомендации обоих авторов для интерфейсных решений зачастую оказываются несовместимыми, а порою прямо противоположными. В свое время нам также не удалось избежать подобного недостатка: анализ эргономических и психологических ограничений, а также влияния привычек и сложившихся стихийно стандартов на качество программной разработки, представленный в работе [8], мог быть более точным, если бы он выполнялся раздельно для конкретного и абстрактного интерфейсов.

Понятно, что в связи с понятиями абстрактного и конкретного интерфейсов есть необходимость развить информацию, представленную в указанных публикациях, а также других более поздних работах. В последующем изложении мы обсуждаем в основном вопросы конструирования абстрактных интерфейсов и лишь частично касаемся разработки конкретных интерфейсов. Для этого есть две причины. Во-первых, разработка абстрактного интерфейса *всегда должна предшествовать* решению вопросов о конкретизации пользовательских средств и методов конструируемой программной системы. Можно утверждать, что в качественно проработанном программном проекте абстрактный интерфейс становится базой для проектируемых средств конкретного интерфейса. Вторая причина связана с различием стиля работ, выполняемых в рамках конструирования двух интерфейсных аспектов: абстрактный интерфейс является более технологичной деятельностью, чем конкретный, который можно с полным на то основанием считать искусством. Исходя из этих причин мы рассматриваем методы развития проектом с позиции методологической теории деятельности.

## 4.1 Общие положения деятельностного подхода

Понятие деятельности — ключевое для проектирования программных систем, и это тема отдельного рассмотрения. Здесь мы рассмотрим лишь те ее аспекты, которые имеют отношение к разработке интерфейсов. И первое, на что стоит обратить внимание, — это *структура деятельности*: из каких элементов она состоит? Естественное ограничение при ответе на этот вопрос — рассматривать только целенаправленные деятельности. Как следствие, появляются следующие *элементы деятельности* (см. рис.2):

- *Цель* — для чего организуется деятельность, т.е. какие продукты и с каким уровнем качества предписано получить при ее выполнении;
- *Результат* — что фактически продуцируется в ходе выполнения деятельности. Результат не всегда соответствует цели. Если он включает все, что предписано целью, деятельность считается успешной, иначе — неуспешной. В обоих случаях при выполнении деятельности не только те продукты, которые соответствует цели (например, опыт разработчиков, отходы и др.);
- *Субъект* — тот, кто выполняет деятельность. Это необязательно индивидуум, поскольку деятельность может выполняться автоматически, и необязательно реальный индивидуум, т.к. деятельность может быть коллективной, и тогда удобно говорить о виртуальном субъекте. Не любой субъект пригоден для выполнения деятельности, а значит, он должен обладать вполне определенными атрибутами (например, соответствующей квалификацией), которые необходимы для выполнения деятельности;
- *Материалы и ресурсы* — то, из чего продуцируются результаты деятельности. Материалы могут быть как материальные, так и информационные, а ресурсы — расходуемыми или не ограниченные (считаются такими);
- *Средства и инструменты* — с помощью чего продуцируются результаты деятельности;
- *Методы* — указывают на то, как выполнять деятельность для получения целевых результатов, потребляя ее материалы и ресурсы.

Деятельность может находиться в одном из ее *состояний*, которые можно рассматривать в качестве характеристики выполнения деятельности. На множестве состояний определено

бинарное отношение возможности перехода из одного состояния в другое. Тем самым задается граф состояний. Переход между состояниями (если он возможен) происходит в результате *действий* субъекта, использующего элементы деятельности. Допускаются изменения состояний и без участия субъекта (пример — конкурентное использование ресурса, общего для разных деятельностей).

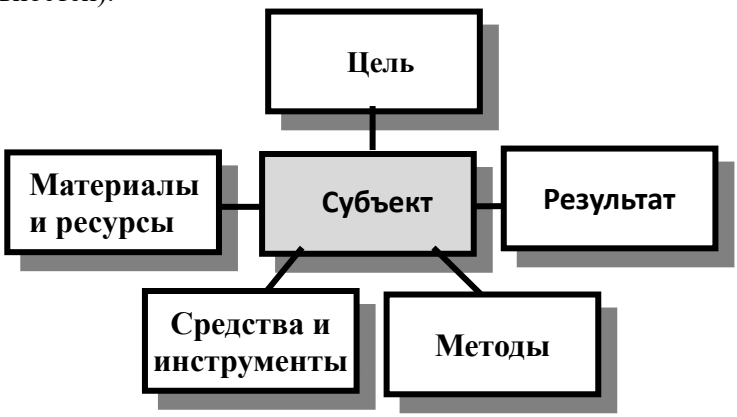


**Рисунок 2**: Элементы деятельности

В графе состояний выделены *начальносе состояние*, в котором деятельность активизируется, и множество *заключительных состояний*, в каждом из которых деятельность может завершиться. Некоторые из заключительных состояний определяются как *целевые* — считается, что в них цели деятельности достигнуты.

Последовательность состояний, которая выстраивается в результате действий субъекта, называется *операционным маршрутом* деятельности. Операционный маршрут называется *полным*, если он ведет от начального состояния в заключительное. Полный операционный маршрут, приводящий к целевому состоянию, считается *успешным*. В отличие от так называемых *сценариев*, которые для попадания в целевое состояние предписывают субъекту определенные последовательности действий, операционные маршруты выстраиваются субъектом, т.е. допускают свободный выбор очередного состояния, что более точно отражает реальность целенаправленных деятельностей.

Деятельность всегда выполняется в некотором *окружении*, из которого поставляются ее элементы, и в которое передаются ее результаты в качестве элементов других деятельностей из окружения. Связанные этими отношениями деятельности могут группироваться, и такие группы можно трактовать как самостоятельные деятельности со своими окружениями и элементами. Таким образом образуются *система деятельностей*, в которой каждая из деятельностей составляется из других деятельностей, актуализирующих элементы групп.

Системы деятельностей образуются на основе отношения использования продуктов и результатов, передаваемых от одной деятельности другой. Эти и связанные с ними понятия уточняются следующим образом:

- *Результат деятельности* — все, что производится в ней, независимо от того, используется это или нет в какой-либо другой деятельности (в последнем случае использование рассматривается вне изучаемой системы);
- *Продукт деятельности* $Д_п$ — та часть результата $Д_п$, которая *используется* в другой деятельности $Д_и$ в качестве какого-то ее элемента. Множество таких деятельностей $A = \{Д_{и1}, \ldots, Д_{ип}\}$ очень велико. В частности, оно содержит все варианты использования всех пользователей продукта;
- *Целевой (основной) продукт деятельности* $Д_п$ — то, что заявлено в $Д_п$ как ее цель, т.е. обозначены некие деятельности $= \{Д_{иi}, i \in \{1, \ldots, n\} \mid Д_{иi} \in A\}$, о которых предполагается, что они будут использовать продукты $Д_п$);
- *Побочный продукт* $Д_п$—та часть результата, не являющаяся целевым продуктом, для которой есть использующая его деятельность $Д_и \in A \setminus \{Д_{и1} \cup \ldots \cup Д_{ип}\}\ i \in \{1, \ldots, n\}$.
- *Сопутствующий продукт* — тот продукт (возможно, другой деятельности), без которого использование целевого продукта затруднительно (в разной степени).

Рис. 3 иллюстрирует введенные понятия. Светлые стрелки указывают на блоки, раскрывающие результаты деятельностей, которые предоставляют целевые и побочные продукты другим деятельностям (они выделенных овалами). Темные стрелки отражают использование продуктов.

Автоматизация деятельности — это подмена каких-то ее элементов программными аналогами. Такая подмена неизбежно влечет изменение и других элементов за исключением ее цели — новый инструмент требует новых методов работы с ним, другой квалификации субъекта и пр. Для эффективного (во всех смыслах) выполнения субъектом деятельности необходимо, чтобы его новые операционные маршруты были бы сходными с ранее реализуемыми маршрутами, в частности, они не должны ломать привычные стандарты взаимодействия с системой. Это справедливо и при конструировании новой деятельности, которая предполагает достижение новых целей. В обоих случаях аналогия помогает освоению автоматизированной деятельности.

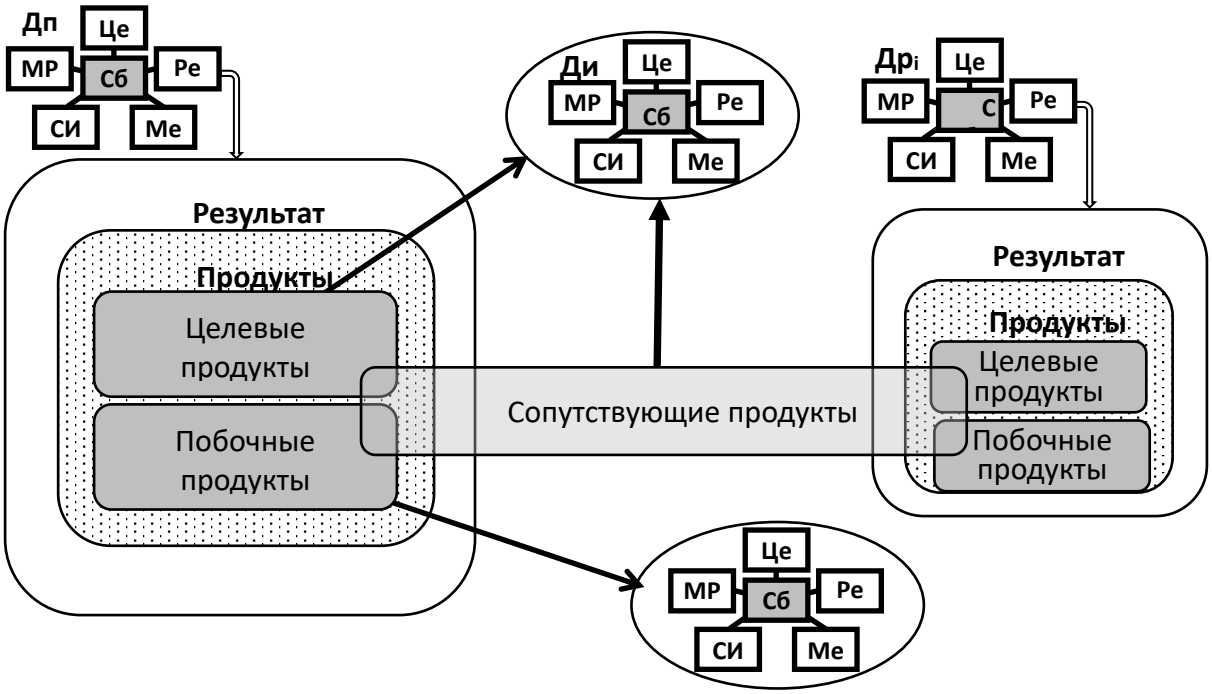


**Рисунок 3**: Результат и продукты деятельности

Сходство нового операционного маршрута с ранее освоенными пользователям, указывает на то, что деятельность, которую реализует этот маршрут, разбивается на фрагментные деятельности, и некоторые из них соответствуют сформировавшимся у пользователя поведенческим или иным стереотипам. Когда стереотипы отвечают эргономическим и когнитивным ограничениям, производительность труда пользователя повышается. Это наблюдение распространяется и на решение задач конкретного интерфейса. Для разработки абстрактного интерфейса оно может служить рекомендацией для выбора подходящих шаблонов в качестве атомарных единиц действий, так называемых виджетов — примитивов графического интерфейса пользователя, имеющий стандартный внешний вид и выполняющий стандартные действия [11].

## 4.2 Конструирование абстрактного интерфейса

Предыдущее обсуждение показывает, что для разработки абстрактного интерфейса необходим *анализ операционных маршрутов*, при котором решается, *что* должно предъявляться пользователю в качестве информации и для выбора вариантов действий, но не *как* это предъявляется: показывается, рассказывается и др. Последнее — задача анализа и

формирования представлений конкретного интерфейса, которую нужно решать после того, как приняты решения на абстрактном уровне.

Вариантов операционных маршрутов слишком много, чтобы пытаться охватить их все. Вместо этого целесообразно строить типовые сценарии пользовательской деятельности, которые ограничивают рассмотрение последовательностями состояний деятельности при движении к цели. В отличие от сценариев, предлагаемых пользователю в качестве руководства, аналитические сценарии для каждого состояния сопровождаются информацией о тех состояниях, в которые можно перейти в принципе. С точки зрения отдельного сценария большая часть переходов в такие состояния — это ошибки. Однако не исключено, что другие сценарии организуют последовательности состояний, для которых данные переходы правомерны. Полезное состояние не будет упущено не только, когда другой сценарий уже входит в число типовых, но и случаях, когда его полезность выяснится в дальнейшем, и придется модернизировать систему.

В результате анализа операционного маршрута разработчики должны для каждого состояния определить:

- Какие элементы деятельности используются в пользовательском действии, как изменяются элементы деятельности;
- Какой результат продуцируется при переходе в новое состояния и в каких деятельностях используются продукты;
- В какой программной поддержке нуждаются эти действия.

На основе информации об операционных маршрутах выполняется следующий шаг анализа — *проверка функциональной полноты*, т.е. соответствие предлагаемых средств полному набору автоматизируемых функций, необходимых для пользовательских действий. Полнота проверяется исходя из текущего понимания потребностей пользователя, или, что то же, на основе всех действий субъекта на всех проанализированных операционных маршрутах. Если в процессе разработки программного продукта или даже в ходе его эксплуатации выясняется нарушение функциональной полноты, то комплект предлагаемых средств расширяется, и проверка повторяется.

Из функционально полного набор средств выделяются *базовые элементы* модели вычислений и требуемого управления абстрактного уровня. Набор этих элементов должен быть достаточен для реализации всех автоматизируемых функций, что означает требование *реализационной полноты*. Это достигается путем унификации и комбинирования элементов, а при необходимости, их декомпозиции. Реализация базовых элементов осуществляется в рамках принятых для проекта архитектурных решений и требований эффективности с использованием как общесистемных, так и специальных программных средств. В результате работ, связанных с обеспечением реализационной полноты, в архитектуре фиксируется достигнутый уровень функциональных возможностей. В терминах шаблона MVC это означает спецификацию модели, управления и реализационной части представления. Последнее есть основа конструирования конкретного интерфейса — вторая часть представления.

Наряду с функциональной и реализационной полнотой имеет смысл рассматривать *интерфейсную полноту*, понимая ее как задание языка управления поведением модели, адекватного восприятию программной системы пользователем. Если задача определения абстрактного интерфейса решена, то определена и семантика такого языка. И тогда можно подбирать варианты возможного интерфейса, учитывая эргономические и когнитивные ограничения, о которых убедительно говорит Дж. Раскин [1]. Самый главный аспект интерфейсной полноты — поддержка всех сторон автоматизируемой деятельности, в частности, в интерфейсе должны быть распознаваемы все элементы этой деятельности.

В связи с предложенной методикой конструирования абстрактного интерфейса необходимо подчеркнуть следующее. Разработчик программной системы не должен полагать, что его решения об операционных маршрутах окажутся единственно правильными, а выполненная им проверка функциональной и реализационная полноты станет гарантией этих качеств для пользователя. Практика показывает, что действительно нужные для пользователей программные системы – это продукты, которые постоянно развиваются для удовлетворения пользовательских потребностей. При переходе к разработке конкретного интерфейса, для каждого варианта предлагаемого пользователям и удовлетворяющего ограничениям нужна

проверка того, достигнута ли поддержка всех сторон автоматизируемой деятельности. И первым шагом в этой проверке является выяснение его соответствия абстрактному интерфейсу. В то же время, среди критериев качества не последнее место занимает точность: среди предоставляемых элементов изображений, звуковых и иных сигналов не должно быть ничего лишнего.

Представленная только что схема применения принципов полноты нуждается в дополнении, когда результаты проверки полноты и точности неудовлетворительны и необходима ревизия проекта. В этом случае следует вернуться к этапу, который стал причиной дефекта. Уточненная схема поддержки конструирования интерфейса представлена на рис. 4.

## 5. Заключение

В данной публикации представлены лишь те аспекты разработки программных систем, которые затрагивают конструирование интерфейсов. Как показывает анализ многих программных проектов, очень часто поддержка деятельности в целом подменяется реализацией лишь требуемых функций без заботы о том, как и для чего пользователь активизирует эти функции. Эта позиция приемлема, когда в деятельности пользователя используется, к примеру, лишь вычислительный результат, получение которого не требует управления. Если же программа предлагается как инструмент деятельности, необходимо понимание всех аспектов этой деятельности, т.е. комплексный системный подход к организации предъявляемых средств.

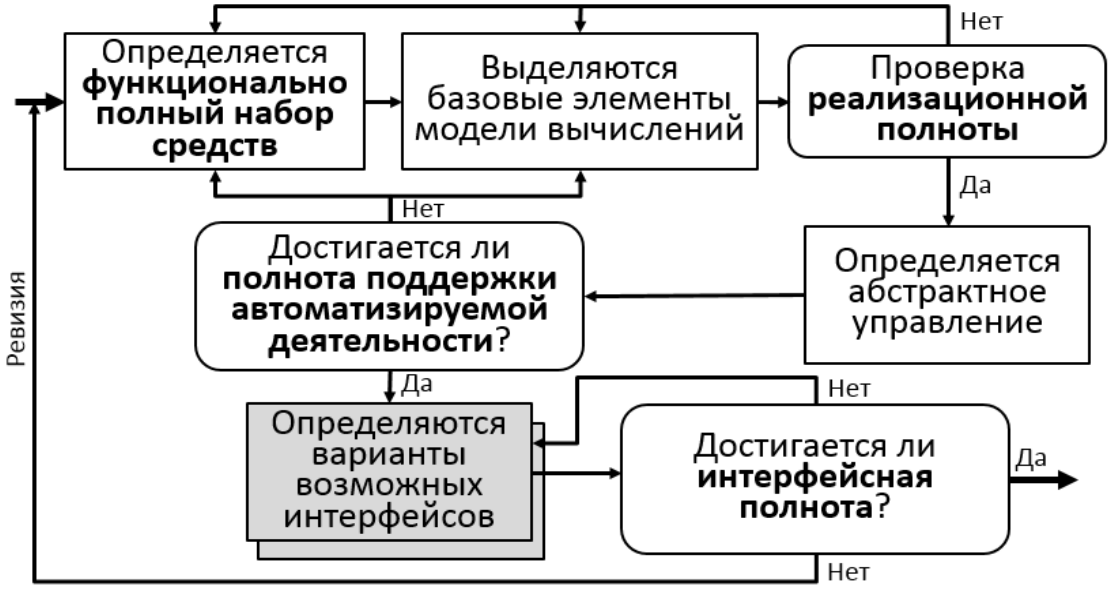


**Рисунок 4**: Схема применения принципа полноты

Комплексность автоматизации пользовательской деятельности предполагает, что разработчики при конструировании программ не ограничиваются учетом эргономических, когнитивных и психологических ограничений, а стремятся к достижению соответствия предлагаемых средств структуре деятельности, понимают, что автоматизируемая деятельность есть часть большой системы деятельностей, игнорирование которой приводит необходимости приспосабливаться к особенностям программы. На это уже довольно давно указывал Г.П.Щедровицкий в исследованиях по общей теории деятельности и ее приложению к проектировочной деятельности. Его статью [12] можно рассматривать как методологическую основу проведенного в настоящей работе анализа и выдвигаемых принципов конструирования интерфейсов.

Рассмотрение задачи конструирования интерфейсов с позиций теории деятельности приводит к тому, что хорошо себя зарекомендовавший шаблон проектирования MVC должен

быть дополнен понятиями абстрактного и конкретного интерфейсов, на основе которых предлагается особый подход к проектированию и реализации интерфейсов. Одним из наиболее важных соглашений этого подхода является принцип полноты поддержки автоматизируемой деятельности, который естественным образом проявляется в функциональной, реализационной и интерфейсной полноте.

Подход, который мы предлагаем для конструирования не только интерфейсов, но и других составляющих разработки программных проектов в ряде аспектов перекликается с тем, что предлагается в документе PMBOK [13] (Project Management Body of Knowledge). Однако нельзя не отметить принципиальное отличие: PMBOK ориентирован на процессное представление проектной деятельности, в котором нет места субъектам, разграничению продукта и результата. В результате проектные деятельности и их части оказываются черными ящиками со входами и выходами, разбавленными инструментами, которые влияют на выполнение процесса. По этой причине концепция деятельностого представления проектов представляется более перспективной тем более, когда речь идет о конструировании интерфейсов.

Более гибким по сравнению с PMBOK, хотя одновременно и более специализированным является стандарт SWEBOK [14] (Software Engineering Body of Knowledge), претендующий на роль Руководства к своду знаний по программной инженерии. Тем не менее, и к этому стандарту можно предъявить аналогичные претензии.

Подводя итог нашему исследованию проблемы адекватной разработки программных интерфейсов, можно сделать вывод о том, что эта тематика развивается недостаточно интенсивно и пока почти не выходит за рамки экспериментов и случайных решений.

## 6. Благодарности

Исследования выполнены в рамках государственного задания ИВМиМГ СО РАН FWNM-2025-0005 и представлены на 7th International Workshop on Information, Computation, and Control Systems for Distributed Environments (ICCS-DE 2025), Июль 7–11, 2025, Иркутск, Россия.

## 7. Список использованных источников

[1] Raskin J. The Humane Interface: New Directions for Designing Interactive System // Addison-Wesley Professional,2000.–256 p. ISBN 0201379376 (ISBN13: 9780201379372)
[2] Перевод на русский: Раскин Дж. Интерфейс: новые направления в проектировании компьютерных систем. // Символ-Плюс. 2005. — 272 с. ISBN 5-93286-030-8
[3] Spolsky J. Joel on Software. // URL: https://m-i-uznetsov.livejournal.com/147976
[4] Перевод на русский: Джоэл о программировании, Символ-Плюс, 2006,
[5] ISBN 5-93286-063-4
[6] Интегрированная_среда_разработки. https://ru.wikipedia.org/wiki/Интегрированная_среда_разработки
[7] Фаронов В. В. Turbo Pascal. Наиболее полное руководство. // BHV-Санкт-Петербург. 2007. — 1054 с. ISBN 5-94157-295-6, CD.
[8] Бутюгин Д.С., Гурьева Я.Л., Ильин В.П., Перевозкин Д.В., Скопин И.Н. Функциональность и технологии алгебраических решателей в библиотеке Krylov. // Вестник ЮУрГУ, 2013, т. 2, №3. — с. 92 – 105.
[9] Обзор новой версии Google Chrome 26. // URL: http://www.comss.ru/page.php?id=1394
[10] YouTube.// URL: http://www.youtube.com/
[11] Скопин И.Н. Разработка интерфейсов программных систем. // В сб. Системная информатика. Вып. 6/Новосибирск: Наука. Сибирская издательская фирма. 1997. с.34 – 96.
[12] Burbeck S. Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC). // URL: http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html
[13] Ахо А.В., Лам М.С., Сети Р., Ульман Д.Д. Компиляторы. Принципы, технологии и инструментарий. // Вильямс. 2014. —1184 с. ISBN 978-5-8459-1932-8.

[14] Что такое виджеты // URL: http://widgetok.ru/2009/01/what-is-widgets/
[15] Щедровицкий Г. П. Автоматизация проектирования и задачи развития проектировочной деятельности. // Разработка и внедрение автоматизированных систем в проектировании (теория и методология). — М.: Стройиздат, 1975. — С. 3 – 177.
[16] Руководство к своду знаний по управлению проектами (Руководство PMBOK®). — Project Management Institute, Inc, 2008. — 465 с.
[17] Guide to the software engineering body of knowledge: 2004 version," Library of Congress Online Catalog, 2005, http://lccn.loc.gov/2005921729. Retrieved 16 July 2013.