Изучение требований, предъявляемых к безопасной и комфортной работе инженеров-программистов.

Знакомство со спецификой этапов жизненного цикла программного продукта.

Изучение принципов работы программ для имитации речевого поведения человека при общении с одним или многими собеседниками.

Изучение языка разметки искусственного интеллекта  AIML (Artificial Intelligence Markup Language) с целью анализа его возможностей для построения интеллектуального чат-бота.

Изучение существующих фреймворков (API) для построения чат-ботов на базе распространенных текстовых менеджеров  (Telegram, Facebook, Skype и др).

Проектирование структуры программного комплекса и базы данных для хранения информации в соответствии с темой дипломной работы.

Разработка программного модуля для запроса информации из базы данных заданной предметной области на основе текстового пользовательского запроса.

Формирование информационной базы для выполнения дипломной работы и тестирования предполагаемых к созданию программных продуктов.

Создание прототипа программного продукта в соответствии с темой дипломной работы.

Методология проектирования информационных систем описывает процесс создания и сопровождения систем в виде жизненного цикла информационной системы, представляя его как некоторую последовательность стадий и выполняемых на них процессов. Для каждого этапа определяются состав и последовательность выполняемых работ, получаемые результаты, методы и средства, необходимые для выполнения работ, роли и ответственность участников и т.д. Такое формальное описание жизненного цикла информационной системы позволяет спланировать и организовать процесс коллективной разработки и обеспечить управление этим процессом.

Жизненный цикл информационной системы можно представить как ряд событий, происходящих с системой в процессе ее создания и использования.

Модель жизненного цикла отражает различные состояния системы, начиная с момента возникновения необходимости в данной информационной системе и заканчивая моментом ее полного выхода из употребления. Модель жизненного цикла – это структура, содержащая процессы, действия и задачи, которые осуществляются в ходе разработки, функционирования и сопровождения программного продукта в течение всей жизни системы, от определения требований до завершения ее использования.

В зависимости от потребностей проекта выбирается соответствующий подход к циклу разработки. В процессе создания программного обеспечения используются разные виды жизненных циклов.

Каскадная стратегия (однократный проход, водопадная или классическая модель) подразумевает линейную последовательность выполнения стадий создания информационной системы (рис.3.1). Другими словами, переход с одной стадии на следующую происходит только после того, как будет полностью завершена работа на текущей. Каждый этап завершается выпуском полного комплекта документации, достаточной для того, чтобы разработка могла быть продолжена командой специалистов на следующем этапе.



Рис. 2. Каскадная схема разработки ПО

К положительным сторонам применения каскадного подхода можно отнести то, что на каждом этапе формируется законченный набор проектной документации, отвечающий критериям полноты и согласованности. Выполняемые в логичной последовательности этапы работ позволяют планировать сроки завершения всех работ и соответствующие затраты.

Каскадный подход хорошо зарекомендовал себя при построении информационных систем, для которых в самом начале разработки можно достаточно точно и полно сформулировать все требования, с тем чтобы предоставить разработчикам свободу реализовать их как можно лучше с технической точки зрения. В эту категорию попадают сложные расчетные системы, системы реального времени и другие подобные задачи. Однако, в процессе использования этого подхода обнаружился ряд его недостатков, вызванных прежде всего тем, что реальный процесс создания программного обеспечения (ПО) никогда полностью не укладывался в такую жесткую схему.

В процессе создания ПО постоянно возникала потребность в возврате к предыдущим этапам и уточнении или пересмотре ранее принятых решений. В результате реальный процесс создания ПО принимал вид, представленный на рис. 3.

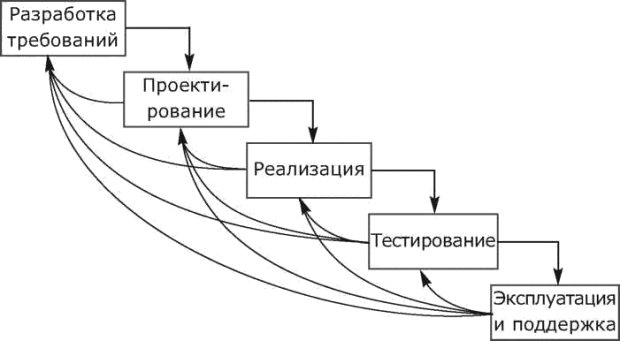


Рис. 3. Реальный процесс разработки ПО по каскадной схеме

Так выглядит поэтапная модель с промежуточным контролем, где разработка ИС ведется итерациями с циклами обратной связи между этапами. Межэтапные корректировки позволяют учитывать реально существующее взаимовлияние результатов разработки на различных этапах. Время жизни каждого из этапов растягивается на весь период разработки.

Основным недостатком такого подхода является существенное запаздывание с получением результатов. Согласование результатов с пользователями производится только в точках, планируемых после завершения каждого этапа работ, требования к ИС сохраняются в виде технического задания на все время ее создания. Таким образом, пользователи могут внести свои замечания только после того, как работа над системой будет полностью завершена. В случае неточного изложения требований или их изменения в течение длительного периода создания ПО, пользователи получают систему, не удовлетворяющую их потребностям. Модели автоматизируемого объекта могут устареть одновременно с их утверждением.

Для преодоления перечисленных проблем была предложена спиральная модель ЖЦ, изображенная на рисунке 4. На этапах анализа и проектирования реализуемость технических решений и степень удовлетворения потребностей заказчика проверяется путем создания прототипов. Каждый виток спирали соответствует созданию работоспособного фрагмента или версии системы. Это позволяет уточнить требования, цели и характеристики проекта, определить качество разработки, спланировать работы следующего витка спирали. Таким образом углубляются и последовательно конкретизируются

детали проекта и в результате выбирается обоснованный вариант, который удовлетворяет действительным требованиям заказчика и доводится до реализации.

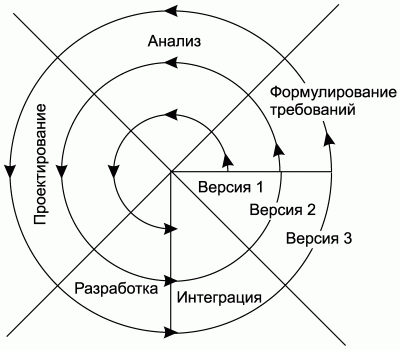


Рисунок 4 – Спиральная модель ЖЦ ИС

Итеративная разработка отражает объективно существующий спиральный цикл создания сложных систем. Она позволяет переходить на следующий этап, не дожидаясь полного завершения работы на текущем и решить главную задачу – как можно быстрее показать пользователям системы работоспособный продукт, тем самым активизируя процесс уточнения и дополнения требований.

Основная проблема спирального цикла – определение момента перехода на следующий этап. Для ее решения вводятся временные ограничения на каждый из этапов жизненного цикла, и переход осуществляется в соответствии с планом, даже если не вся запланированная работа закончена. Планирование производится на основе статистических данных, полученных в предыдущих проектах, и личного опыта разработчиков.

Agile

Гибкая методология разработки (англ. *Agile software development*, *agile*-методы) – серия подходов к разработке программного обеспечения, ориентированных на использование интерактивной разработки, динамическое формирование требований и обеспечение их реализации в результате постоянного взаимодействия внутри самоорганизующихся рабочих групп, состоящих из специалистов различного профиля. Существует несколько методик, относящихся к классу гибких методологий разработки, в частности экстремальное программирование, *DSDM*, *Scrum*, *FDD*.

Большинство гибких методологий нацелены на минимизацию рисков путём сведения разработки к серии коротких циклов, называемых итерациями, которые обычно длятся две-три недели. Каждая итерация сама по себе выглядит как программный проект в миниатюре и включает все задачи, необходимые для выдачи мини-прироста по функциональности: планирование, анализ требований, проектирование, программирование, тестирование и документирование. Хотя отдельная итерация, как правило, недостаточна для выпуска новой версии продукта, подразумевается, что гибкий программный проект готов к выпуску в конце каждой итерации. По окончании каждой итерации команда выполняет переоценку приоритетов разработки.

Методы *Agile* – это такие гибкие методологии, как *Lean Development* («Бережливая разработка ПО»), *Scrum* и др. Они были разработаны еще в начале 2000-х как альтернатива малоэффективным традиционным *IT* методам.

Практически все *аgile*-команды сконцентрированы в одном офисе (*bullpen*). Офис включает *product* *owner* – заказчика, который и определяет требования к продукту. В качестве заказчика может выступать бизнес-аналитик, менеджер проекта или клиент. Кроме того, в офис могут входить и дизайнеры интерфейса, тестировщики, технические писатели. То есть методы *Agile* направлены в первую очередь на непосредственное общение.

Основной метрикой *agile*-методов является рабочий продукт. Отдавая предпочтение непосредственному общению, *agile*-методы уменьшают объём письменной документации по сравнению с другими методами.

Основные идеи:

* люди и взаимодействие важнее процессов и инструментов;
* работающий продукт важнее исчерпывающей документации;
* сотрудничество с заказчиком важнее согласования условий контракта;
* готовность к изменениям важнее следования первоначальному плану.

Принципы *Agile*:

1. Удовлетворение клиента за счёт ранней и бесперебойной поставки ценного программного обеспечения;
2. Приветствие изменений требований даже в конце разработки (это может повысить конкурентоспособность полученного продукта);
3. Частая поставка рабочего программного обеспечения (каждый месяц или неделю или ещё чаще);
4. Тесное, ежедневное общение заказчика с разработчиками на протяжении всего проекта;
5. Проектом занимаются мотивированные личности, которые обеспечены нужными условиями работы, поддержкой и доверием;
6. Рекомендуемый метод передачи информации – личный разговор (лицом к лицу);
7. Работающее программное обеспечение – лучший измеритель прогресса;
8. Постоянное внимание улучшению технического мастерства и удобному дизайну;
9. Простота – искусство не делать лишней работы;
10. Лучшие технические требования, дизайн и архитектура получаются у самоорганизованной команды;
11. Постоянная адаптация к изменяющимся обстоятельствам.

Главные преимущества *Agile*:

* Качество продукта. Вовлечение заказчика в процесс каждой итерации дает возможность корректировать процесс, что неизменно повышает качество.
* Высокая скорость разработки. Итерация длится не более 3-х недель, к концу этого срока обязательно есть результат.
* Минимизация рисков. Крупный проект дает возможность заказчику оплатить несколько итераций и в ходе работы понять, что он вовремя получит именно то, что хочет и за приемлемую цену. Водопадные модели (с применением спецификаций и технических заданий) таких возможностей не дают.
* Заказчик всегда имеет возможность наблюдать за ходом разработки, корректировать фунуциональность проекта, тестировать или запускать его, даже может остановить его в любой момент.

Вопросы применения систем виртуального общения на основе искусственного интеллекта исследуют на протяжении многих лет. На сегодняшний день проблема виртуального общения актуальна из-за быстрого доступа к информации, возможности одновременной работы в системе многих пользователей, обмена информацией, взаимодействия с целью решения любых вопросов, поддержки обучения, коммуникации с клиентами и партнерами по бизнесу, проведения аналитических исследований, сбора необходимой информации, повышения квалификации и других преимуществ.

Основными вопросами в создании систем общения являются разработка модели общения, модели участника общения, развитие средств, в первую очередь, семантических и прагматических, описания окружающей среды (модели языка, модели пользователя, модели окружающей среды, модели системы общения). Поэтому для решения этих вопросов необходимо определение принципов работы, особенностей имитации речевого поведения человека в процессе общения, разработка модели общения, написания чат-бота.

Среди программ-собеседников есть программы, созданные на основе искусственного интеллекта. При разработке таких программ необходимо знать психологию, а также принципы построения фраз человеческой речи. Более того, если правильно определить языковые ограничения и предметную область, то существующими методами можно получить системы, пригодные для общения. С точки зрения теории языка и общения необходима разработка семантического описания структур текстов и предложений. С точки зрения модели окружающей среды основным ограничением является отсутствие средств для представления динамически меня

Для необходимо определить, что такое бот. Бот – это некоторый программный комплекс для решения каких-либо задач, обладающий признаками искусственного интеллекта. К ним относятся: способность принимать решения на основе эмпирически полученных данных, понимать семантику данных, проводить анализ и выдавать оптимальное решение. Все слышали о ботах в компьютерных играх. Это небольшие библиотеки, встроенные или настраиваемые. Они анализируют математическую модель игры и получают решение, которое становится их стратегией. Другие боты – это виртуальные пользователи. Программа сама перемещает мышь по экрану, нажимает клавиши и выполняет прочие функции, присущие биологическому организму, сидящему перед монитором. Ещё одна разновидность ботов: чат-боты.

Чат-бот – это специальный бот, который позволяет вести диалог с человеком путём вычленения информации на основе определённых шаблонов поиска и приведения её к стандартизированному виду. При этом стандарт в каждом чат-боте задаётся по-разному. Семантический разбор естественного языка считается достаточно сложной задачей. Чтобы её облегчить, были придуманы шаблоны семантического анализа. Проще говоря, это некоторые алгоритмы, применяемые к тексту, которые возвращают наличие или отсутствие в нём определённого признака.

Чат-боты в основном используются для общения с людьми в чатах, на форумах и пр. Однако некоторая их категория применяется для анализа текста и выявления в нём информации с последующим помещением её в реляционные БД.

Принцип работы чат-бота заключается в реализации этапов: бот принимает входящие сообщения, анализирует их и отсылает результат выполнения и/или выполняет команду [1]. Следовательно, общение в чат-ботах осуществляется путем ввода сообщений и вывода ответа (мнения) собеседника. Здесь возможны два вида ведения разговора: обычная беседа или обсуждение важного вопроса. Но, в отличие от разговора людей, программа не обладает гибким интеллектом, поэтому большинство виртуальных собеседников запрограммированы на ведение несложной беседы. Такие программы относят к классу программ с естественным языковым интерфейсом [1]. Обработка естественного языка человека, особенно разговорного стиля, является проблемой, касающейся искусственного интеллекта. Проблема создания программ собеседников на базе искусственного интеллекта, которые могут моделировать интеллектуальную деятельность человека, на сегодняшний день остается открытым.

К сожалению, современные виртуальные собеседники лишь частично решают вопрос имитации разговора человека. Основу их функционирования составляет база знаний. В простейшем случае она содержит наборы возможных вопросов пользователя и соответствующих ответов на них. Наиболее распространенные методы выбора ответов в данном случае следующие: реакция на ключевые слова; совпадение фразы пользователя с той, которая есть в базе знаний; также программа может учитывать порядок слов. Программы-собеседники не могут использовать фразы, насыщенные местоимениями. В таких случаях программы анализируют предыдущие фразы пользователя и выбирают наиболее приемлемый ответ. Также проблематичным может быть подбор слов-синонимов.

На сегодняшний день разработано большое количество ботов. Среди них можно выделить наиболее распространенные: *A*.*L*.*I*.*C*.*E*., *ChatMaster*, *Electronic* *Brain*, *ELIZA*, *George*, *NAI*, *SkypeTalk* и другие.

Так, британская компьютерная программа *George* победила в *Loebner* *Prize* – конкурсе чат-ботов на наилучшее соответствие языку человека, который проводят ежегодно.

*SkypeTalk* ведет диалог на украинском языке. Возможности и знания можно расширить, поскольку основная часть написана на специальном диалекте *XML*, что достаточно гибко позволяет переписывать бот под свои потребности и обновлять базу. Версию *XML*, используемую в *A*.*L*.*I*.*C*.E*.,* распространяется под лицензией *GNU* *GPL*, поэтому существует много свободных реализаций этого языка [2], а также программ, использующих этот язык (интерпретаторов). Наиболее активно используемая версия интерпретатора, написанная на языке *Java*. Существует также формальная спецификация языка и *XML*-схема, данная консорциумом *W3C*.

Уникальные возможности Интернет такие, как скорость, оперативность, доступность коммуникации между пользователями – позволяют использовать сеть как средство общения и создавать интерактивные формы общения: чаты, форумы, телеконференции, электронную почту и другие. На смену реальным собеседникам приходят программы искусственного интеллекта, такие как чаты, консультанты, помощники, развлекательные программы и другие. Но, в отличие от разговора людей, программа не обладает гибким умственным интеллектом.

К сожалению, современные виртуальные собеседники лишь частично решают вопрос имитации разговора человека. Словарный запас большинства виртуальных собеседников ограничен, кроме этого, у них отсутствует эмоциональная окраска, тембр голоса и пр., поэтому большинство виртуальных собеседников запрограммированы на ведение несложной беседы.

Обработка естественного языка человека, особенно разговорного стиля, является проблемой, касающейся искусственного интеллекта. В основу функционирования виртуальных собеседников положена база знаний.

Уникальные возможности Интернет такие, как скорость, оперативность, доступность коммуникации между пользователями – позволяют использовать сеть как средство общения и создавать интерактивные формы общения: чаты, форумы, телеконференции, электронную почту и другие. На смену реальным собеседникам приходят программы искусственного интеллекта, такие как чаты, консультанты, помощники, развлекательные программы и другие. Но, в отличие от разговора людей, программа не обладает гибким умственным интеллектом. К сожалению, современные виртуальные собеседники лишь частично решают вопрос имитации разговора человека. Словарный запас большинства виртуальных собеседников ограничен, кроме этого, у них отсутствует эмоциональная окраска, тембр голоса и пр., поэтому большинство виртуальных собеседников запрограммированы на ведение несложной беседы. Обработка естественного языка человека, особенно разговорного стиля, является проблемой, касающейся искусственного интеллекта. В основу функционирования виртуальных собеседников положена база знаний. В простейшем случае она содержит наборы возможных вопросов пользователя и соответствующих ответов на них. Некоторые программы могут учиться, а именно: пополнять словарный запас, учитывать определенные особенности языка, стиля общения. Но проблема создания программ собеседников на базе искусственного интеллекта, которые могут моделировать интеллектуальную деятельность человека, на сегодняшний день остается открытой. Несмотря на преимущества, виртуальные собеседники в настоящее время не могут пройти тест Тьюринга на соответствие интеллекта компьютера человеческому интеллекту.

К сожалению, на современном этапе своего развития виртуальное общение не может заменить реальное, но компьютерные технологии постоянно совершенствуются, поэтому, возможно, через какое-то время виртуальные собеседники смогут заменить реальное общение.

На сегодняшний день остается актуальным создание программ имитирующих общение человека. Простейшей моделью общения является база вопросов и ответов к ним [1]. В данном случае возникает проблема описания базы знаний и реализация программы интерпретатора. Язык разметки базы знаний может включать в себя паттерны вопросов и соответствующие им шаблоны ответов, также предысторию диалогов к ним и название соответствующей темы общения.

*AIML* (*Artificial* *Intelligence* *Markup* *Language*) – язык разметки для искусственного интеллекта. *AIML* – диалект *XML* для создания виртуальных собеседников.

Файл *AIML* – содержит набор категорий (*category*). Категория открывается тегом *<category>* и закрывается тегом *</category>.* Каждому тегу открытия должен соответствовать тег закрытия.

Категория (*category*) включает паттерны (*pattern*, обращения) и темплейты (*template*, ответные реплики). Паттерн пишется заглавными буквами.

Для определения произвольного слова или группы слов используется знак \*

Примеры паттернов:

1. <pattern> ПРИВЕТ*</pattern>*
2. *<pattern>* ПРИВЕТ *\*</pattern>*
3. *<pattern> \** ПРИВЕТ *\*</pattern>*
4. *<pattern> \** ПРИВЕТ*</pattern>*

Паттерн 1 подойдет только для входящей реплики содержащей единственное слово «Привет» – и ни для какой другой.

Паттерн 2 подойдет только для входящей реплики, начинающейся со слова «Привет» и содержащей еще какие-нибудь слова после слова «Привет», например, «Привет Бот!».

Паттерн 3 подойдет только для входящей реплики, начинающейся с 1 или нескольких слов, за которым следует слово «Привет» и содержащей еще какие-нибудь слова после слова «Привет», например, «Эй ты привет красавчик!».

Паттерн 4 подойдет только для входящей реплики, начинающейся с одного или нескольких слов, за которым следует слово «Привет», например, «Бот, привет!».

Темплейт может включать одну реплику, например:

*<template>* Добрый день!*</template>*

Или несколько реплик. В этом случае необходимо использовать дополнительный тег *<random>.* Пример:

*<template>*

*<random>*

*<li>* Добрый день! *</li>*

*<li>* Здравствуйте *</li>*

*<li>* Приветик… *</li>*

*</random>*

*</template>*

В этом случае в качестве ответной реплики будет произвольно выбрана одна из реплик, находящаяся между тегами *<li> </li>.*

Рандом можно комбинировать с обычной репликой. Например:

*<template>*

*<random>*

*<li>* Добрый день! *</li>*

*<li>* Здравствуйте *</li>*

*<li>* Приветик… *</li>*

*</random>*

Как дела?

*</template>*

В этом случае бот ответит «Добрый день! Как дела?» или «Здравствуйте Как дела?» или «Приветик… Как дела?»