Содержание

[Введение 2](#_Toc319340687)

[1 Прототипирование архитектуры ПО 3](#_Toc319340688)

[1.1 Прототипирование ПО 3](#_Toc319340689)

[1.2 Понятие прототипа архитектуры ПО 7](#_Toc319340690)

[1.3 Использование прототипов архитектуры ПО 7](#_Toc319340691)

[1.4 Требования к прототипам архитектуры ПО 7](#_Toc319340692)

[1.5 Инструменты прототипирования архитектуры ПО 8](#_Toc319340693)

[1.6 Выводы 9](#_Toc319340694)

# Введение

Современные методы и практики быстрой разработки программного обеспечения нацелены на миницизацию рисков, путем сведения процесса к серии коротких циклов, каждый их которых пердставляет программный проект в миниатюре и включает все задачи, необходимые для выдачи минимального прироста по функциональности. При таком подходе особенно важно представлять всю систему целиком на ранних этапах разработки. Для решения подобных задач применяются прототипы программных систем, которые позволяют вести разработку проекта методом непрерывной интеграции.

Настоящая работа представляет собой исследование современных решений в области прототипирования программного обеспечения, оценку их эффективности и применимости согласно выдвинутой модели требований, а также выводы о необходимости появления нового класса инструментов прототипирования, в виду неготовности существующих решений удовлетворять ранее выдвинутым требованиям.

Кроме того, в работе детально представлена предлагаемая автором модель системы описания и тестирования прототипов архитектуры программного обеспечения и рассмотрена ее реализация с точки зрения современных технологий программирования. Основополагающая идея предлагаемого подхода заключается в использовании унифицированного языка в процессе описания прототипов, а также свойств конечных автоматов в процессе тестирования.

# 1 Прототипирование архитектуры ПО

## 1.1 Прототипирование ПО

Конструирование или проектирование программного обеспечения – это непростой процесс, требующий много усилий и внимания. Хорошо спроектированное приложение может в дальнейшем сэкономить много времени и сил его разработчикам. Кроме того проектирование является едва ли не самой крупной частью процесса разработки и в значительной степени влияет на успешность всего процесса разработки. Кроме того процесс проектирования является своего рода центральной частью разработки: до него происходит этап выработки требований к разрабатываемому программному обеспечению и начальной разработки его архитектуры, после него – тестирование и доработка приложения.

Результатом этапа конструирования является готовый исходный код программы. Спецификации требовании и проектная документация могут устареть, но исходный код актуален всегда, и именно поэтому он должен быть максимально качественным. Часто процесс конструирования также называют «кодированием» или «программированием». Конструирование — единственный процесс, который выполняется во всех случаях.

Идеальный программный проект до начала конструирования проходит стадии тщательной выработки требований и проектирования архитектуры. После конструирования в идеале должно быть выполнено исчерпывающее тестирование системы. Однако в реальных проектах разработчики часто пропускают этапы выработки требований и проектирования, начиная прямо с конструирования программы. Тестирование также часто выпадает из расписания из-за огромного числа ошибок и недостатка времени. Отсюда можно сделать вывод, что повышение эффективности конструирования программного обеспечения позволяет оптимизировать любой проект, каким бы несовершенным он ни был. Кроме того качество конструирования ПО - это один из основополагающих факторов оценки качества всего ПО в целом.

В процессе конструирования условно можно выделить следующие этапы:

* детальное проектирование – продумывание архитектуры приложения, отрисовка диаграмм (интерфейсов, классов и т.д.),
* кодирование – написание программного кода в соответствии с результатами, полученными на этапе детального проектирования,
* отладка – этап конструирования, на котором локализуются и устраняются ошибки в коде программы,
* интеграция – процесс объединения отдельных компонентов в единую систему,
* тестирование – процесс выявления ошибок программы с применением различных подходов. Он позволяет получить представление о качестве спроектированного приложения и исправить ошибки, которые не были выявлены и исправлены на этапе отладки.

Если в результате этапа конструирования мы получаем готовый программный продукт, то процесс прототипирования позволяет в результате получить макет готовой системы. Такой макет можно проверить на пригодность предлагаемых для применения концепции, архитектурных и технологических решений, а так же предоставить на ранних этапах разработки заказчику.

Прототипирование можно назвать ускоренной версией проектирования, поскольку в этом случае к основными стадиями можно отнести следующие:

* определение начальных требовании,
* разработка первого варианта прототипа системы на основе требовании,
* изучение прототипа и получение обратной связи о необходимых изменениях и дополнениях,
* переработка и улучшение прототипа (с учётом полученных замечаний и предложений изменяются как спецификации, так и прототип).

Зачастую результат, полученный в процессе прототипирования, может и не стать частью готовой системы. Но, тем не менее, он может служить ещё одним шагом на пути к созданию прототипа финальной версии разрабатываемого продукта.

Выделяют различные подходы к прототипированию. Но, в общем, принято выделять два основных, принципиально различных подхода – это быстрое и эволюционное прототипирование. Эволюционное прототипирование заключается в последовательном создании макетов системы, которые будут все ближе и ближе к реальному продукту. Несомненное преимущество такого подхода в том, что на каждой шаге мы имеем рабочую систему, пусть и не располагающую всеми необходимыми нам функциями, но уже более приближенную к финальной версии, нежели предыдущая версия системы. Такой подход очень удобен в ситуации, когда все требования к системе ещё не определены, и будут определяться в процессе разработки. Однако при использовании быстрого прототипирования заранее предполагается, что создаваемый макет на каком-то этапе будет оставлен и не войдёт в готовую систему. Безусловно, преимуществом такого подхода является скорость – в ответ на требования заказчика сразу проектируется каркас системы. Этот каркас системы отдаётся заказчику, требования вновь уточняются или изменяются и вновь происходит создание каркаса. Стоимость внесения изменений и создания нового каркаса очень низкая, поскольку на этом этапе не нужно писать код системы, а создаётся только её каркас. К преимуществам использования прототипирования как такового можно отнести уменьшение времени разработки и стоимости системы за счёт улучшения спецификации, а также вовлечение пользователей или заказчиков в процесс разработки.

Но у прототипирования так же есть и недостатки:

* недостаточный анализ (акцентирование внимания разработчиков на ограниченном прототипе может отвлечь их от анализа требовании на итоговую систему),
* чрезмерное время на создание прототипа (если разработчики проектируют слишком сложную систему и тратят много времени, то все преимущества от использования прототипирования теряются),
* смешение представлений пользователей или заказчиков о прототипе и готовой системе (есть вероятность, что они могут потерять отличие между прототипом и основой будущей системы и разочароваться в возможностях разработчиков).

На этапе работы над каркасом приложения удобно пользоваться различными вспомогательными средствами, среди которых можно выделить универсальный язык графического описания для объектного моделирования UML. С его помощью можно получить прототип системы в виде различных схем и диаграмм, которые с разных сторон отразят особенности разрабатываемой архитектуры приложения. Полученные диаграммы проверяются только на правильность и соответствие стандартам UML и не гарантируют правильность относительно требований, предъявленных к разрабатываемой системе.

Разрабатываемое приложение должно позволять пользователю составить каркас системы, используя специализированный набор инструментов, затем протестировать разработанный каркас на соответствие предъявленным требованиям, и по результатам тестирования позволить выполнить дополнительные модификации прототипа или же выполнить кодегенерацию на основе созданного прототипа. На данный момент сложно найти программный продукт, позволяющий выполнить этот набор манипуляции.

Можно создавать прототипы:

* архитектуры,
* новой функциональной возможности уже существующей системы,
* структуры илисодержания внешних данных системы,
* инструментальных средств или компонентов,
* рабочих характеристик,
* дизайна пользовательского интерфейса.

## 1.2 Понятие прототипа архитектуры ПО

Протипы архитектуры создаются, чтобы смоделировать будущую систему в целом. Ни один из отдельных модулей в прототипе не должен быть особенно функциональным.

## 1.3 Использование прототипов архитектуры ПО

в использщовании можно написать и про ЖЦ и про скрам и про ажайл

Когда разработчики сталкиваются с разработкой чего-то нового и еще не существующего, они в первую очередь подвержены большому риску выбрать неверный способ и пойти по неправильному пути. А поскольку заказчики и пользователи ранее не сталкивались с подобного типа системами, то и требования их могут быть неточными и расплывчатыми. Кроме того, сами разработчики вынуждены будут использовать средства алгоритмы, методики или библиотеки, с которыми они не знакомы. Таким образом, получается, что разработчики сталкиваются с большим количеством неизвестных.

Самый часто применяемый выход из этой ситуации – это составление предельно подробных спецификаций системы. Написание большого количества документации, которые будут четко регламентировать каждое требование к системе, связывать каждое неизвестное и ограничивать рабочую среду.

Однако, существуют и другие способы решения такой проблемы. Одним из которых является «стрельба трассирующими». Характерной особенностью данного метода является то

## 1.4 Требования к прототипам архитектуры ПО

Цель работы с прототипом – исследование определенных характеристик и аспектов конечной версии системы. При построении прототипа можно пренебречь деталями и особенностями системы, которые в данный момент не важны.

Детали, которые можно не учитывать при работе над прототипом:

* корректность – там, где это приемлимо, можно использовать фиктивные данные,
* завершенность – прототип может функционировать лишь в ограниченном смысле, возможно лишь с одним заданным фрагментом данных и одним пунктом меню,
* надежность – процедура проверки ошибок, вероятно, будет неполной или будет отсутствовать полностью. При отклонении от определенного пути прототип может выйти из строя,
* стиль – прототип программы не имеет большого значения для комментариев или документации.

Готовый прототип архитектуры системы должен дать ответы на многие вопросы. Например:

* четко ли определены обязанности основных компонентов,
* являются ли эти обязанности приемлимыми для компонентов,
* четко ли определена совместная работа основных компонентов,
* сведено ли к минимуму связывание между компонентами,
* можно ли выделить потенциальные источники дублиования,
* можно ли применять определения интерфейсов и ограничения,
* обладает ли каждый из модулей путем доступа к данным, требуемым ему в ходе выполнения? Может ли он получить его в случае необходимости.

## 1.5 Инструменты прототипирования архитектуры ПО

Поскольку большинство прототипов создается с целью моделирования рассматриваемой системы в целом, то полученный прототип есть не что иное, как одноразовая программа, необходимая для того, чтобы получить ответы на определенный ряд вопросов. В прототипах опущены ненужные детали и подробности, что позволяет в центре рассмотрения иметь лишь определенные аспекты системы. Для создания прототипа не нужно писать программу – он может быть составлен даже на обычном листе бумаги или доске. Главная цель составления такого прототипа – это получить понимание того, как система будет выглядеть в собранном виде, опуская детали. С этой точки зрения может показаться удобным создание прототипов посредством языков очень высокого уровня, а точнее языков более высокого уровня по сравнению с языком, используемым при написании системы. К таким языкам, например, можно отнести Perl и Python. Язык сценариев высокого уровня позволяет опустить многие детали (например, указание типов данных), но при этом создавать функциональный фрагмент программы. Такие языки также позволят при необходимости соеденить низкоуровневые фрагменты в новые сочетания. В итоге, используя такой подход, можно быстро собрать существующие компоненты в новые конфигурации и посмотреть, как они работают.

## 1.6 Выводы

Согласно приведенным выше рассуждениям, можно сделать вывод о том, что основополагающая проблема эксплуатации современных систем мониторинга заключается в отсутствии на рынке целого класса комбинированных систем, одновременно объединяющих в себе преимущества как распределенных, так и расширяемых систем мониторинга. Кроме того, современные тенденции развития облачных и кластерных решений в области суперкомпьютерных технологий, лишь подтверждают необходимость в появлении подобных инструментов мониторинга.

Таким образом, рассмотренные выше проблемы эксплуатации систем мониторинга, позволяют сделать вывод о неготовности существующих решений комплексно выполнять выдвинутые к ним требования.

Авторами предлагается проект распределенной системы мониторинга и диспетчеризации процессов гетерогенной среды, которая позволяет обеспечить выполнение перечисленных требований. Основополагающая идея предлагаемого в проекте подхода заключается в использовании механизма разработки и исполнения дополнительных модулей в процессе решения задач мониторинга, а также свойств распределенных систем в процессе эксплуатации.