DOI: 10.22184/1992-4178.2023.226.5.108.115

Анализ архитектуры виртуальных прототипов аппаратуры на SystemC

Д. Гаврилова[1], Т. Мадумаров[2]

УДК 621.3:004.4 | ВАК 2.2.2

Виртуальные прототипы встраиваемых систем ускоряют их создание за счет параллельной разработки аппаратной и программной частей системы. В больших, быстроразвивающихся проектах трудно уследить за возникающими ошибками и проблемами с интеграцией компонентов. В статье проведен анализ архитектуры виртуального прототипа для выявления проблемных компонентов с целью их последующей доработки. На ранних стадиях такой подход упрощает разработку программного обеспечения в будущем.

На сегодняшний день одним из важнейших разделов мировой науки и техники является создание встраиваемых аппаратно-программных систем, служащих для управления и обработки информации во всех областях жизнедеятельности человека: в космосе, в воздухе, на воде и под водой, на земле. Функционирование таких систем обеспечивается их программным обеспечением (ПО), разработка, отладка и испытания которого – трудоемкий, дорогой и длительный процесс. Кроме этого, необходимо программное обеспечение для наладки и тестирования собственно аппаратной части системы.

Аппаратура, для которой разрабатывается ПО, появляется далеко не сразу, и поэтому для ускорения разработки, отладки и тестирования ПО создается модель аппаратной части, на которой можно отлаживать и тестировать ПО. Такую модель в дальнейшем будем называть *виртуальным прототипом аппаратуры* [1] (далее – виртуальный прототип). Использование виртуальных прототипов позволяет начать разработку и отладку ПО раньше, чем будет готов физический прототип.

Для проведения процесса тестирования и наладки прибора необходимо также и стендовое оборудование, предназначенное для симуляции сигналов из внешней среды и измерения параметров выходных сигналов тестируемой системы. Виртуальный прототип аппаратуры, помимо модели вычислителя, включает и модель тестового стенда.

[1] МИЭТ, Институт МПСУ им. Л. Н. Преснухина, студент-бакалавр; АО НИИ «Субмикрон», старший техник-программист, gavrilova.spk@gmail.com.

[2] АО НИИ «Субмикрон», руководитель группы, madrook00@gmail.com.

Часто для разработки виртуальных прототипов встраиваемых систем используется язык программирования (далее – ЯП) C++ и библиотека SystemC [2, 3]. Данная библиотека предназначена для моделирования аппаратуры на различных уровнях абстракции. Самым низким уровнем, на котором поддерживается моделирование, является уровень регистровых передач (Register Transfer Level, RTL), а самым высоким – уровень транзакций (Transaction Level Modeling, TLM). Поскольку быстродействие является важным параметром виртуального прототипа, был выбран именно TLM-уровень, на котором время моделируется на уровне обменов, а не тактов.

Архитектура виртуального прототипа получилась сложной из-за необходимости поддерживать моделирование широкой номенклатуры аппаратуры. Было принято решение провести анализ устойчивости архитектуры к изменениям при помощи метода, который был введен Робертом Мартином в книге «Чистая архитектура» [4].

При анализе архитектуры главным элементом является компонент. Он представляет собой элементарную единицу, которую можно развертывать в составе ПО. Правильное проектирование дает компоненту возможность независимого развертывания и модификации. Метод состоит в том, чтобы для каждого компонента ПО измерить две метрики: *устойчивость*, которая отражает сложность модификации данного компонента, и *абстрактность*, которая отражает возможность расширения компонента. Для установки связи между устойчивостью и абстрактностью был введен принцип устойчивости абстракций (Stable Abstractions Principle, SAP). Он гласит, что для поддержания гибкости необходимо, чтобы абстрактные компоненты были устойчивыми.

Для анализа архитектуры виртуальных прототипов на SystemC данный метод нуждается в адаптации, поскольку он был разработан для использования

с объектно-ориентированными ЯП, а C++ является мультипарадигменным.

Цель работы – найти компоненты, нарушающие SAP, которые помешают в будущем расширить возможности виртуального прототипа. Для этого необходимо формализовать архитектуру в виде UML-диаграмм, а также провести анализ архитектуры по методу Роберта Мартина [4].

Актуальность исследования определяется необходимостью быстро разрабатывать виртуальные прототипы для большого количества различных вычислителей. Для каждого из них модификация виртуальных прототипов – это частая задача, поэтому многие компоненты будут подвергаться изменениям и расширению. Поскольку ошибки в архитектуре являются очень дорогостоящими для исправления, актуально проводить анализ архитектуры виртуальных прототипов (далее – просто архитектура, если не указано другое) на ранних этапах проектирования и по мере усложнения архитектуры.

ИСПОЛЬЗУЕМЫЕ МЕТОДИКИ АНАЛИЗА АРХИТЕКТУРЫ

Компоненты зависят друг от друга. В коде это отражается, например, в виде #include-директивы, когда сущность, определенная в одном компоненте, используется в другом. При рассмотрении зависимостей отдельного компонента будем называть *исходящей* зависимостью каждую сущность, определенную в других компонентах и используемую в данном. *Входящей* зависимостью будем называть сущность, определенную в данном компоненте и используемую в других.

Устойчивые компоненты представляют собой набор сущностей, которые неизменны с течением времени. Такие компоненты, как правило, имеют множество входящих зависимостей и мало зависимы от других компонентов. Именно зависимость других компонентов делает их устойчивыми.

Согласно SAP, чем больше в компоненте ПО абстракций, тем проще расширить его функционал. Чем больше компонентов от него зависит, тем сложнее его изменить. Если все устойчивые компоненты являются абстрактными, то добавление нового поведения реже ведет к изменению компонентов.

Пусть In – это число входящих зависимостей, то есть число классов, определенных в компоненте и используемых в зависимых компонентах, а Out – число классов, определенных в других компонентах и используемых в анализируемом. Тогда неустойчивость компонента I определяется следующим образом:

$$I = \frac{Out}{In + Out}.$$

Эта метрика принимает значения в диапазоне от 0 до 1, где 0 означает максимальную устойчивость компонента, а 1 – максимальную неустойчивость.

Другой метрикой является *абстрактность* (A), которая зависит от количества абстрактных классов и интерфейсов. Абстрактные классы и интерфейсы через полиморфизм позволяют организовывать взаимозаменяемость компонентов. Любые компоненты, реализующие заданный интерфейс, являются взаимозаменяемыми. Это позволяет расширять поведение приложения путем внедрения зависимостей, что не приводит к изменению компонентов. Для расчета абстрактности (A) необходимо найти отношение числа абстрактных классов и интерфейсов (N_a) к общему количеству классов (N_{all}) компонента:

$$A = \frac{N_a}{N_{all}}.$$

Значение метрики находится в пределах 0 и 1, где 0 означает отсутствие абстрактных классов, а 1 означает, что все классы в компоненте абстрактные.

Для анализа архитектуры программного обеспечения предлагается оценить связь между абстрактностью и устойчивостью. На графике (рис. 1) точке с координатами (0,1) соответствует компонент, обладающий максимальной абстрактностью и устойчивостью, а точке с координатами (1, 0) – обладающий максимальной неустойчивостью и конкретностью. Прямая проходящая через эти точки называется главной последовательностью [4].

Компоненты, лежащие ближе к главной последовательности, лучше всего удовлетворяют SAP. Чем дальше компонент от главной последовательности, тем больше вероятность создания проблем в будущем. Конкретный

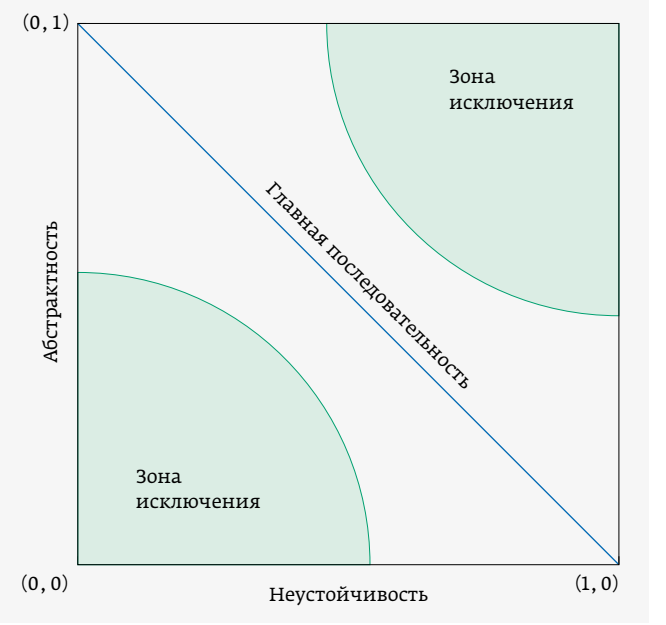

Рис. 1. График зависимости абстрактности от неустойчивости

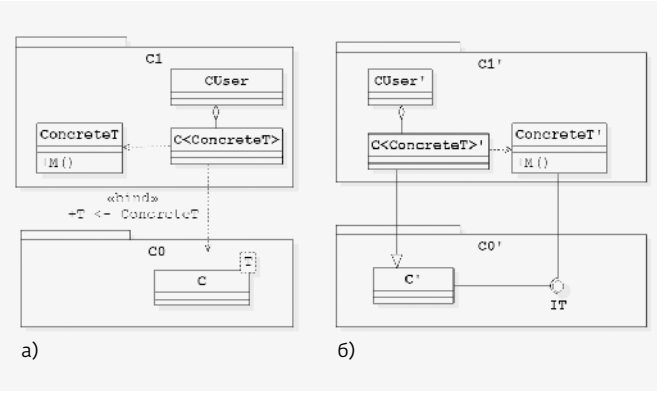

Рис. 2. UML-диаграмма абстракции при помощи шаблонов классов (а) и интерфейсов (б)

и устойчивый компонент будет сложно изменять, а неустойчивый и абстрактный компонент чаще всего бесполезен. Области вокруг этих точек называются зонами исключения. Положение точек в этих зонах нежелательно и приводит к проблемам в процессе разработки. Для оценки компонента необходимо определить расстояние до главной последовательности, D:

$$D=|A+I-1|,$$

где A – абстрактность, I – неустойчивость.

Оно принимает значение от 0 до 1, где 0 означает, что компонент на главной последовательности, а 1 – значительно удален от главной последовательности.

Для наглядной демонстрации архитектуры проекта мы будем пользоваться универсальным языком моделирования UML [5].

Поскольку в данной статье анализируется архитектура приложения, разработанного на ЯП C++, необходимо как-то учитывать возможность абстрагирования кода при помощи шаблонного метапрограммирования. Допустим, имеется шаблонный класс C<T>, который использует набор методов M из параметра шаблона T. С точки зрения абстрактности это эквивалентно тому, что класс T реализует гипотетический интерфейс IT, который реализует набор методов M.

Из рис. 2 видно, что замена компонентов C0 и C1 на C0' и C1' не нарушает принцип инверсии зависимости, а, соответственно, с точки зрения расчета метрики абстракции, данные схемы эквивалентны и класс C<T> можно считать абстрактным. При этом полную специализацию шаблона класса, а также шаблоны классов с параметрами-данными будем считать конкретными.

Поскольку ЯП C++ является мультипарадигменным, организовать полиморфизм в нем можно другими способами, помимо абстрактных классов. Одним из способов организации полиморфизма, помимо шаблонного метапрограммирования, является передача указателей на функции. При разработке виртуальных прототипов мы использовали модуль functional из стандартной библиотеки. Зависимость от подобной функции эквивалентна зависимости от интерфейса с одной функцией, но является более выразительной и занимает меньше места (см. пример 1).

Пример 2 демонстрирует организацию инъекции зависимости через абстрактный класс (через объект класса std::function из модуля functional).

```
class IInterface {
public:
virtual int interfaceMethod(const std::string& str) = 0;
};

class Dependent {
public:
int callDependency() {
return m_dep->interfaceMethod(m_name);
}
...
private:
IInterface* m_dep = nullptr;
std::string m_name;
};
```

Пример 1.

```
class Dependent {
public:
int callDependency() {
return m_dep(m_name);
}
...
private:
std::function<int (const std::string&)> m_dep;
std::string m_name;
};
```

Пример 2.

www.monolit.by

МОНОЛИТ

ВИТЕБСКИЙ ЗАВОД РАДИОДЕТАЛЕЙ

МНОГОСЛОЙНЫЕ КЕРАМИЧЕСКИЕ КОНДЕНСАТОРЫ

ИМПОРТОЗАМЕЩАЮЩАЯ ПРОДУКЦИЯ
для высоконадёжной аппаратуры

ТЕРМОРЕЗИСТОРЫ
с положительным температурным коэффициентом сопротивления

РЕГИСТРЫ НАГРЕВАТЕЛЬНЫЕ

Республика Беларусь, 210101,
г. Витебск, ул. М. Горького, д. 145
Телефон: +375 (212) 36-45-05 (приемная)
Факс: +375 (212) 36-44-07
E-mail: info@monolit.by

Конструкторско-технологический отдел:
Телефон: +375 (212) 36-44-21
E-mail: kto@monolit.by

Отдел маркетинга и сбыта
Маркетинг:
Телефон: +375 (212) 36-44-52
E-mail: monolmarket@mail.ru
Сбыт:
Телефон: +375 (212) 36-45-34;
+375 (212) 36-45-42
E-mail: monosbet@mail.ru

СПЕЦЭЛЕКТРОНКОМПЛЕКТ
КОМПЛЕКТНЫЕ ПОСТАВКИ ЭЛЕКТРОННЫХ КОМПОНЕНТОВ

ЭКСКЛЮЗИВНЫЙ ДИЛЕР НА ТЕРРИТОРИИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Акционерное общество «СПЕЦЭЛЕКТРОНКОМПЛЕКТ»

www.monolit.by

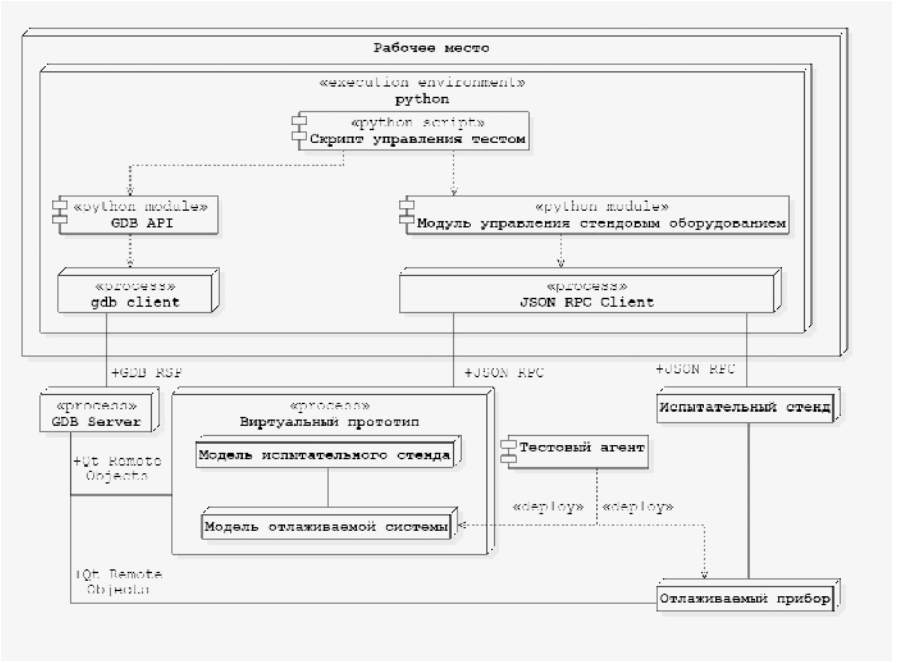

Рис. 3. UML-диаграмма развертывания системы тестирования

В таком случае мы будем увеличивать количество абстрактных сущностей компонента на 1, поскольку оба фрагмента эквивалентны с точки зрения инверсии зависимостей.

Внутренние классы не будем учитывать при подсчете абстрактности, поскольку их назначение – функциональная декомпозиция.

АРХИТЕКТУРА ВИРТУАЛЬНОГО ПРОТОТИПА АППАРАТУРЫ

Диаграмма развертывания системы тестирования изображена на рис. 3. Из диаграммы видно, что виртуальный прототип должен реализовывать механизмы удаленного вызова процедур (Remote Procedure Call, RPC). RPC – это технологии, позволяющие программам вызывать функции или процедуры на удаленных узлах или в другом адресном пространстве на том же узле. Для поддержки отладки ПО, развернутого на виртуальном прототипе, при помощи отладчика GDB используется механизм RPC, входящий в состав библиотеки Qt Remote Objects [6].

Для работы со стендовым оборудованием был использован протокол JSON-RPC [7]. Это легковесный формат для обеспечения RPC, который не требует зависимости от такого тяжелого набора программных модулей, как Qt. Помимо этого, и клиент, и сервер могут быть разработаны с использованием различных ЯП, что позволит интегрировать данную систему в любое окружение и даст возможность заказчику использовать для управления тестированием любой современный скриптовый ЯП.

Для анализа архитектуры рассмотрим абстрактный виртуальный прототип аппаратуры на платформе MIPS32 (рис. 4). Данный виртуальный прототип служит для отладки и тестирования компонентов и в качестве демонстрационной задачи, в нем нет особой практической потребности, однако, он имеет в своем составе все необходимые пакеты и поэтому подходит для архитектурного анализа.

Поскольку на базе одних и тех же компонентов могут быть разработаны различные виртуальные прототипы, необходима библиотека управления инверсией зависимостей. Поскольку существующие библиотеки такого рода для ЯП C++ достаточно тяжеловесны, была разработана библиотека tiny_ioc, которая не зависит от boost.

Библиотеки sm.vp.ip содержат наборы классов, из которых в основном приложении при помощи библиотеки tiny_ioc можно будет составить виртуальный прототип.

Для моделирования обменов со стендовым оборудованием используется библиотека sm_qt_json_rpc, которая реализует RPC-сервер.

Для реализации синхронизации между RPC-сервером и средой SystemC была разработана библиотека submic_system_c, которая содержит примитивы синхронизации между симуляцией SystemC и средой исполнения Qt.

Такая архитектура виртуального прототипа позволяет, во-первых, быстро добавлять поддержку новой виртуальной платформы за счет использования инъекции

Осциллографы высокого разрешения

АКИП-4140

Основные возможности и преимущества

- Разрядность АЦП 12 бит
- Полоса пропускания: 100 / 200 / 350 / 500 МГц, расширяемая электронным ключом
- Максимальная частота дискретизации 2 ГГц
- Максимальный объем памяти 200 МБ
- Высокая скорость сбора данных: до 100.000 осц./сек (до 500.000 осц./сек в режиме сегментированной развертки)
- Сегментированная память: до 80.000 сегментов (с отображением межсегментного времени)
- Режим "Поисковая машина" и Режим HISTORY
- Режим БПФ на интервале 2М точек
- Статистическая обработка результатов измерений
- Синхронизация и декодирование: I^2C, SPI, UART/RS232, CAN, LIN – стандартно, I^2S, MIL-1553, FlexRay, CAN FD, Manchester (только декодирование) – опция
- Осциллограф смешанных сигналов (16 цифровых каналов) – опция
- Генератор сигналов (функциональный + СПФ – опция); частота дискретизации 125 Мвыб/с
- Программная опция ПКЭ - полный набор измерений и анализа электрической

prist.ru

У точности есть имя: ПРИСТ

111141, **г. Москва**, ул. Плеханова 15а; тел.: +7 (495) 777-5591; факс: +7 (495) 640-3023
196006, **г. Санкт-Петербург**, ул. Цветочная, д. 18, лит. В, офис 202; тел./факс: +7 (812) 677-7508
620089, **г. Екатеринбург**, ул. Цвиллинга, д. 58, офис 1; тел./факс: +7 (343) 317-3999; ek@prist.ru

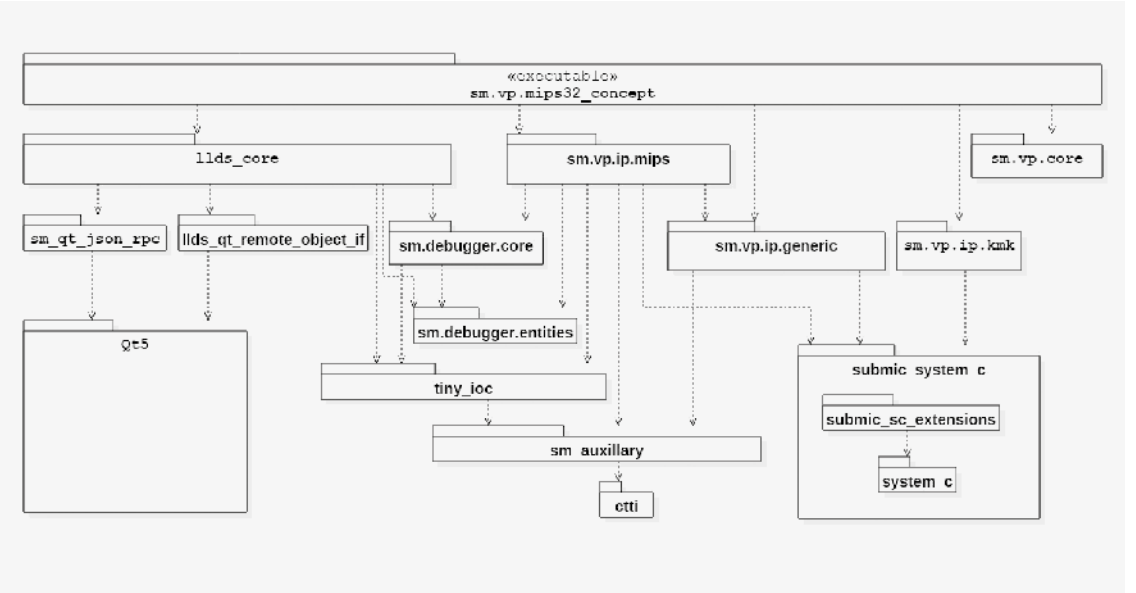

Рис. 4. UML-диаграмма пакетов виртуального прототипа

зависимости при помощи пакета tiny_ioc. Во-вторых, обеспечивает гибкость при изменении требований к моделируемому прибору за счет использования инверсии зависимостей. В-третьих, обеспечивает необходимые механизмы удаленного вызова процедур.

РЕЗУЛЬТАТЫ АНАЛИЗА

Для определения значения метрик не было найдено подходящих инструментов подсчета зависимостей классов от других классов, а также подсчета абстрактных классов. Метрики были измерены вручную: просматривались проекты и их классы. Прежде всего было необходимо просмотреть файлы компонентов и отметить зависимости классов от классов других компонентов, а также абстрактность классов. На основе этих данных созданы таблицы Excel с расчетом метрик в каждом компоненте. Такой подход является трудозатратным, поэтому необходимо найти решение для автоматизации подсчета метрик.

Данные, полученные при расчете метрик, указаны в табл. 1. На основании этих данных можно построить график зависимости абстрактности от неустойчивости (рис. 5). На основании полученной информации можно сделать следующие выводы.

Компоненты sm.debugger.core, tiny_ioc и sm_auxiliary находятся в зоне главной последовательности. Можно заключить, что модули спроектированы в соответствии

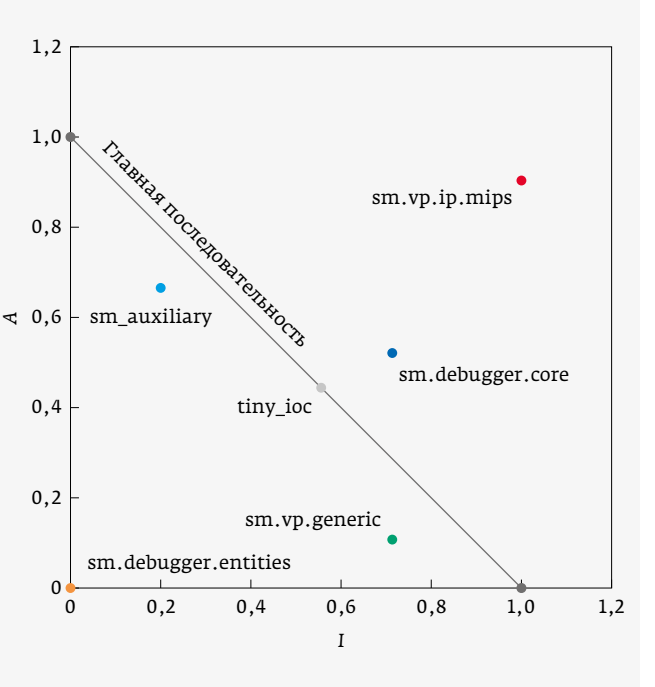

Рис. 5. Диаграмма рассеяния компонентов

Таблица 1. Просчитанные метрики компонентов

Компонент	Абстрактность	Неустойчивость	Расстояние до главной последовательности
sm_auxiliary	0,67	0,20	0,13
tiny_ioc	0,44	0,56	0,00
sm.debugger.entities	0,00	0,00	1,00
sm.debugger.core	0,52	0,71	0,24
sm.vp.generic	0,11	0,71	0,17
sm.vp.ip.mips	0,91	1,00	0,91

с правилами чистой архитектуры и возникновение с течением времени проблем от компонентов маловероятно.

Компонент sm.vp.generic находится в зоне главной последовательности, но, так как система не готова полностью и предполагается его использование другими компонентами, можно прогнозировать его смещение влево, ближе к зоне исключения. Этот компонент требует внимания при будущем использовании.

Компонент sm.vp.ip.mips находится на достаточно далеком расстоянии от главной последовательности и входит в зону исключений. В этой области компоненты близки к состоянию бесполезности. Так как в нынешней системе не достает всех компонентов, можно предполагать сдвиг точки левее, к более устойчивому положению. Этот компонент пока можно сохранить в своем состоянии до появления полных данных.

Компонент sm.debugger.entities находится в максимально отдаленной точке и требует большего внимания. В этой зоне находятся компоненты, которые с высокой долей вероятности доставят много трудностей в последующих изменениях. Его устойчивость не может быть изменена, и поэтому необходимо увеличить его абстрактность.

* * *

В результате проделанной работы методика анализа архитектуры при помощи метрик абстрактности и устойчивости была частично адаптирована для анализа архитектуры ПО, разработанного с использованием ЯП C++, использующего шаблонное метапрограммирование и объекты-функторы для инъекции зависимостей. С помощью данной методики был проведен анализ архитектуры виртуального прототипа встраиваемой системы на архитектуре MIPS32.

Выявление на ранних стадиях проекта потенциально опасных компонентов позволяет сейчас решить множество проблем, которые должны возникнуть в будущем. В настоящее время доступна не вся информация, так что некоторые компоненты являются благоприятными, но в будущем могут попасть в зону исключения. Необходимо помнить об этом во время разработки и заранее исправлять несостоятельные компоненты. Такой подход увеличивает временные затраты сейчас, но экономит время в будущем, избавляя от доработок и проблем с интеграцией компонентов.

Была выявлена необходимость разработки автоматизированной программы для анализа архитектуры на основе UML-диаграмм и исходного кода во избежание трудоемкого ручного расчета метрик.

ЛИТЕРАТУРА

1. **Кечиев Л. Н.** Электрофизические основы конструирования электронной аппаратуры. Инженерное пособие. М.: Грифон, 2020. 480 с.
2. **Мадумаров Т. А., Стефанцов А. В.** Проблемы внедрения SystemC в качестве каркаса для виртуального прототипирования бортовой аппаратуры и стендов для разработки БПО // НАНОИНДУСТРИЯ. 2018. № 9. С. 184–185.
3. IEEE Standard for Standard SystemC® Language Reference Manual. – Design Automation Standards Committee. 2011.
4. **Мартин Р.** Чистая архитектура. Искусство разработки программного обеспечения. – СПб: Питер, 2018. 352 с.
5. **Розенберг Д., Скотт К.** Применение объектного моделирования с использованием UML и анализ прецедентов / Пер. с англ. М.: ДМК Пресс, 2022. 160 с. ил. (Серия «Объектно- ориентированные технологии в программировании»).
6. Qt 5.13 Documentation // The QT Company URL: https://doc.qt.io/archives/qt-5.13/index.html (дата обращения: 25.03.2023).
7. JSON-RPC 2.0 Specification // JSON-RPC URL: https://www.jsonrpc.org/specification (дата обращения: 25.03.2023).