Логика обновления. Перед тем как отрисовывать объект, необходимо учесть прошедшее время и рассчитать его положение в пространстве, а также другие изменения, которые могли произойти за это время. Для разных объектов реализуется совершенно разная логика поведения. Однако можно выделить общие абстрактные зависимости. Например, объекты должны реагировать на столкновения с другими объектами. Для таких целей разработан CollisionController, который обрабатывает такие события.

Логика отрисовки. Разные объекты необходимо отрисовывать по-разному, учитывая тени, альфа-канал, материал, постобработку, а также их комбинации. В силу этого логика отрисовки вынесена как отдельная сущность, общие черты которой можно выявить и составить эффективную иерархию наследования.

Итак, обобщая все изложенное выше, получаем следующую структуру. Разработанное ядро CoreX содержит в себе информацию о текущей загруженной сцене, а также хранилища для всего контента [1, 2]. Сцена представляется собой набор объектов (комбинаций сущностей), которые «знают», как себя вести за все время своего жизненного цикла. Предлагаемая конструкция позволяет достаточно просто генерировать многообразие объектов, используя абстрактные типы поведения и отрисовки, связывая их с текстурами, мешами и произвольным шейдером для материала. Кроме того, поддерживается этап постобработки, информация для которого может быть сгенерирована в процессе отрисовки конкретных объектов.

Список использованных источников:

1. Ломакин, Г.А. Об одном подходе к построению графических приложений / Г.А. Ломакин // Наука-2011 : сб науч. ст. В 2ч. Ч. 2 / БГУ ; рекол.: В.И. Малюгин (отв. ред.) [и др.]. – Минск : БГУ, 2011. – С. 89-91.

2. Ломакин, Г.А. О разработке веб-среды, поддерживающей создание 3D-приложений / Г.А. Ломакин // Веб-программирование и Интернет-технологии WebConf2012: материалы 2-й Междунар. науч.-практ. конф., 5–7 июня 2012 г., С. 233-234.инск. – Минск Изд. центр БГУ, 2012. – С. 164-165.

ПОСТРОЕНИЕ ИНФРАСТРУКТУРЫ СИСТЕМЫ АВТОМАТИЗАЦИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Мойса Д. С

Серебряная Л.В — к.т.н, доцент

В докладе рассматривается способ построения инфраструктуры поддержки жизненного цикла программного обеспечения, а также необходимые организационные изменения для внедрения такого способа работы с инфраструктурой.

В современных процессах разработки очень важное место занимает инфраструктура поддержки и автоматизации этих процессов, а также сама среда работы продукта. Наиболее часто используемые системы: контроля версий, сборки, непрерывной интеграции, развертывания, тестирования и т. д. Очень важно, чтобы такие системы работали бесперебойно, так как нарушение их работы может негативно повлиять на процесс разработки в целом, а иногда и полностью остановить его. Наиболее частые факторы риска: ошибки конфигурации, неработоспособность систем в течение некоторого времени или их полная неработоспособность.

Ошибки в таких системах часто связаны с человеческим фактором. Чаще всего их настройка и поддержка происходит в ручном режиме, что влечет за собой высокую вероятность внесения ошибки. Также нередки ситуации, когда при смене ключевого сотрудника не происходит передача необходимых знаний новому работнику, и информация о необходимой конфигурации остается потерянной.

При серьезных сбоях возможно полное разрушение системы. В этом случае единственным выходом является настройка этой системы с нуля. Все это время процесс разработки, поддерживаемый поврежденной системой, будет нарушен и могут возникать сложности вплоть до полной остановки разработки. Таким образом, надежность инфраструктуры поддержки процессов разработки является важной и актуальной задачей.

Наиболее часто применяемыми мерами для сохранения и быстрого восстановления систем являются резервные копии, однако восстановление из такой копии может также занять длительное время. Кроме того, копия может быть устаревшей, либо содержать только данные, не включающие информацию о конфигурации.

Представление инфраструктуры в виде кода — это процесс написания сценариев от создания виртуальной машины и установки операционной системы до установки и настройки программного обеспечения, связей между экземплярами и других параметров конфигурации. Среда, описанная сценариями, позволяет применять одну и ту же конфигурацию к одному узлу или нескольким узлам. Автоматизация инфраструктуры вносит гибкость как в процесс разработки, так и в эксплуатацию, потому что любой уполномоченный член команды может изменять сценарии с применением к инфраструктуре надлежащих методов разработки, таких как автоматизированное тестирование и управление версиями. Это значит, что к описанию инфраструктуры можно применять те же метрики эффективности и качества, что и к коду приложений.

Важным моментом в применении данного принципа является то, что необходимо обеспечить порядок применения изменений кода инфраструктуры к самой инфраструктуре. Для минимизации рисков необходимо организовать полный цикл работы с кодом инфраструктуры:

- версионирование кода инфраструктуры – исходный код хранится в системе контроля версий;
- ревизия и обсуждение кода ключевыми членами команды перед помещением в репозиторий (код-ревью);
- модульное тестирование (где возможно) – позволяет убедиться, что отдельные модули и подмодули после изменения кода возвращают ожидаемые результаты;
- автоматизированное функциональное тестирование кода инфраструктуры;
- автоматизированное развертывание – инфраструктура разворачивает и тестирует саму себя;
- мониторинг развернутой инфраструктуры и контроль за процессом извне;

Таким образом, организация вышеописанного процесса позволяет добиться следующих преимуществ:

1. Быстрое внесение и применение изменений. Любое изменение – это изменение кода, которое, проходя через все системы автоматически тестируется и применяется.
2. Возможность быстрого развертывания всей инфраструктуры или ее части с нуля.
3. Самодокументируемость конфигурации инфраструктуры.
4. Версионность инфраструктуры.
5. Повышение надежности.

Следовательно, предложенный подход позволяет существенно упростить настройку и поддержку инфраструктуры жизненного цикла программного обеспечения, а также заметно снизить риски, связанные с ней.

Список использованных источников:

1. Morris K. Infrastructure as Code — O'Reilly, 2016.
2. Taylor M. Learning Chef. A Guide to Configuration Management and Automation — O'Reilly, 2014

МЕТОД ОБНАРУЖЕНИЯ УЯЗВИМОСТЕЙ К SQL-ИНЪЕКЦИЯМ В WEB-ПРИЛОЖЕНИЯХ

Белорусский государственный университет информатики и радиоэлектроники, г. Минск, Республика Беларусь

Оношко Д.Е.

Бахтизин В.В. — к.т.н., профессор

Широкое использование web-приложений и повышение их сложности для решения новых задач поднимают значимость оценки их качества на новый уровень. Обнаружение уязвимостей к SQL-инъекциям, являющихся распространённым видом уязвимостей web-приложений, представляет собой важную часть такой оценки.

По состоянию на 2013 год по данным OWASP наиболее распространённым классом уязвимостей существующих приложений являются уязвимости-инъекции [1]. В web-приложениях наиболее частым видом инъекций являются SQL-инъекции. Ошибки, допущенные разработчиками при разработке алгоритмов обработки данных, позволяют злоумышленнику изменить поведение web-приложения путём формирования запросов, использующих эти ошибки, и являются причиной уязвимости web-приложений к SQL-инъекциям.

Большинство методов обнаружения уязвимостей к SQL-инъекциям основывается на анализе поведения web-приложения в условиях эксплуатации, т.е. на динамическом анализе. Однако такие методы по своей сути являются частным случаем функционального тестирования, и, следовательно, их эффективность определяется выбором тестовых данных, для которых производится анализ поведения, что в свою очередь означает невозможность обнаружения всех возможных ошибок данного класса. Статический анализ, т.е. анализ исходных кодов без запуска web-приложения, напротив, позволяет, последовательно анализируя отдельные части web-приложения, формально доказать корректность алгоритмов обработки данных либо выявить ошибки в них, а значит, получить ответ на вопрос о наличии в web-приложении эксплуатируемых и потенциальных уязвимостей к SQL-инъекциям. Вместе с тем, являясь рутинной процедурой, подобный анализ может быть автоматизирован посредством программного средства контроля качества кода, основанного на абстрактной интерпретации исходных кодов web-приложения.

Предлагаемый метод обнаружения уязвимостей основан на рассмотрении web-приложения как множества процедур $Q = \{P_1, P_2, \ldots, P_N\}$, вызывающих друг друга с некоторыми параметрами. Обмен данными между процедурами может реализовываться в различных языках программирования по-разному, поэтому при использовании метода предлагается приводить все возможные взаимодействия к одному из двух видов формальных параметров: in-параметры (данные, передаваемые в процедуру) и out-параметры (данные, возвращаемые из процедуры). Формальным параметрам процедур назначаются оценки, в простейшем случае — бинарного характера: «опасный» (U) или «безопасный» (S), причём U < S.