Приветствую вас, читателя, на конечной остановке этого путешествия в статьи разных мыслителей. В данном выступлении автор затрагивает темы сложности, интерактивности, визуализации, сохранения обратной совместимости, а главное изменения подходов к созданию программ. Свой рассказ о программах, живых и мертвых, начинает с воспоминаний о молодости: «В начале была стопка проткнутых перфокарт, и было это сложно, и сложность проявляла себя тупым образом». Это был ужасно медленным процессом – проверить результаты работы программы можно было только через 45 минут. Кроме того, программы всегда начинались с пустого листа. Время шло, а программисты так и не могли менять программы во время их выполнения, ошибки обрушивали программы, а компиляция и запуск были мало между с собой связаны. Оказывается, большинство программистов остаются заложниками старых методов программирования и пишут свой код точно так же и по сей день. После восхваления функциональных языков автор переходит к проблеме представлении программ. Он обнаруживает, как многие вещи, что изначально были удобны, стали рудиментами бородатых годов: бессмертный 80-столбцовый и раскладка vi. Но такие, казалось бы, случайные и не связанные с настоящим решения всё ещё накладывают свой отпечаток на современный стиль программирования. Несуразное прошлое крепко держит нас за горло, но Джек Рашер готов стать нашим спасителем и предложит своё решение. Этот решение — интерактивное программирование (Лисп из 60-х) и визуализация кода и вывода программ (Oracle Designer из 80-х).

По моему мнению, этот выступление является очень годным с «провидческой» точки зрения, но со стороны прагматики — очень слабым. Те технологии, которых он привёл в качестве положительных примеров на самом деле уже испытаны и в большем мере отклонены из-за значимых слабостей: Lisp и Clojure страдает от безумного использования макросов и скорости повсеместной динамической типизации. Smalltalk — медленный, модель «мутабельного всего» склонена к ошибкам, а редактирование кода на основе изображений уступает статическим системам контроля версий. Erlang в задачах, связанных с процессором, медленнее этого семестра, так как архитектура без общего доступа плохо использует память компьютера. У Julia приличный синтаксис, но она все еще слишком медленная из-за динамических типов. И в конце, визуальное программирование обладает поносной масштабируемостью (доклад Андерса Хейлсберга об истории его языков — они пробовали этого ещё в 90-х годах). В конечном счете проблема заключается в том, что обеспечить максимальную скорость работы кода, оставаясь при этом полностью интроспектируемым, просто невозможно. Чтобы его обеспечить, некоторые вещи действительно должны быть распакованными, статическими частями, а не живыми, визуализируемыми объектами. Вот почему в более практичных языках есть разделение между сборками отладки и сборками выпуска. Не стоит забыть и про тесты: они обеспечивают еще более быстрый цикл обратной связи без необходимости запуска каких-либо пакетных заданий.

Итак, по моему скромному мнению самым близким к идеалам из этого доклада предметом является такая система, как например JVM или CLR, где: программы компилируются в байт-код и запускаются виртуальной машиной с настраиваемыми параметрами; код можно динамически загружать или генерировать, сохраняя при этом скорость благодаря JIT-компилятору; некоторые типы примитивны и распакованы из-за скорости, но ссылочные типы могут быть полностью изучены во время выполнения, а в этом случае рефлексия вывозит; встроены богатые исключения для обнаружения ошибок даже в рабочей среде; запущенные процессы можно удаленно отлаживать и профилировать при условии, что они были запущены с соответствующими параметрами виртуальной машины; тестировать и имитировать сложный код очень просто. Так что да, мне бы хотелось работать со средой на основе JVM, которая могла бы визуализировать работающую программу и обрабатывать динамическую перезагрузку кода волей-неволей. Но переходить к Smalltalk или Lisp — уже перебор.