Khi xem bài nói chuyện The Future of Programming của Bret Victor, tôi đã rất bất ngờ bởi cách ông nhập vai một kỹ sư từ năm 1973 và phủ nhận các giả định, những điều mà ta vốn xem là hiển nhiên trong lập trình hiện đại, để dự đoán cho một tương lai của ngành lập trình. Ông ấy đã đem đến một luận điểm rất hay – sự hoài nghi và bó buộc lối tư duy của con người chính là sự cản trở cho sự phát triển của các công nghệ mới.

Bret là một kỹ sư ở năm 1973, thời điểm mà máy tính còn thô sơ, đã mường tượng ra một tương lai nơi chúng ta không còn cần viết mã như trước. Ông đưa ra khái niệm **"thao tác trực tiếp với dữ liệu"**, một ý tưởng mà ông xem như bước đột phá, lấy ví dụ từ Sketchpad của Ivan Sutherland. Đó là hệ thống cho phép người dùng vẽ và áp đặt các ràng buộc mà không cần viết bất cứ dòng mã nào. Nhìn vào thực tế ngày nay, tôi nhận thấy ý tưởng này đã phần nào trở thành hiện thực trong các công cụ thiết kế đồ họa, ví dụ như Figma. Tuy nhiên, tôi không nghĩ rằng con người có thể tránh khỏi việc phải viết mã nguồn. Những công cụ này chỉ giải quyết bề nổi; khi cần một logic phức tạp cho những dự án mới mẻ (thậm chí là những công nghệ chưa xuất hiện), ta vẫn phải quay về với code.

Kế tiếp, Bret đề cập đến **lập trình định hướng mục tiêu và ràng buộc**, nơi mà lập trình viên chỉ cần mô tả điều mình muốn, còn hệ thống sẽ tự tìm cách thực hiện. Prolog là một ví dụ ông nhắc đến, cũng như khái niệm về constraint programming. Rất thú vị theo khía cạnh khoa học, nhưng xét về mặt tâm lý, có một trở ngại lớn: Con người luôn muốn có quyền kiểm soát mọi thứ. Liệu các lập trình viên, bản chất là con người, vốn sợ mất quyền kiểm soát - có để cho một hệ thống “toàn quyền nghĩ thay" mình? Điều này đòi hỏi một sự tin tưởng lớn vào công cụ, thứ mà có thể bị can thiệp bởi nhiều yếu tố.

Bret đã nói về việc **biểu diễn thông tin dưới dạng không gian.** Trải nghiệm sẽ trở nên trực quan và dễ hiểu hơn. Nhưng sẽ không hiệu quả đối với mọi trường hợp. Việc tìm kiếm lỗi trong một hệ thống đồ họa có thể phức tạp hơn debug mã nguồn thông thường. Tôi đồng ý với Bret rằng chúng ta cần nhiều hơn việc viết mã trong text file, nhưng text có một sức mạnh riêng: nó dễ lưu trữ, dễ tìm kiếm, và dễ kiểm soát bằng các công cụ như Git. Có thể tương lai sẽ có sự kết hợp giữa hai mô hình này thay vì loại trừ lẫn nhau.

**Lập trình song song và mô hình phân tán** là phần mà Bret Victor cho rằng sẽ thay đổi căn bản cách ta thiết kế phần mềm. Ông phê phán sâu sắc mô hình thread và lock, và nhấn mạnh rằng nếu sau 40 năm ta vẫn dùng thread để quản lý concurrency thì "chúng ta đã thất bại". Thật buồn cười là, sau đúng 40 năm, phần lớn phần mềm phổ biến vẫn dùng thread và lock! Nhưng điều đó không có nghĩa là ông sai. Actor Model, CSP, hay hệ sinh thái microservices đã mở ra một kỷ nguyên mới cho lập trình song song và phân tán, chỉ là chúng chưa thay thế hoàn toàn những gì cũ kỹ. Và tôi nghĩ, có lẽ điều này cũng không cần thiết – chúng có thể cùng tồn tại, phục vụ cho những mục đích khác nhau.

Dù không hoàn toàn đồng tính với những luận điểm trên của Bret Victor, nhưng thông điệp của ông về **nguy cơ trở thành nạn nhân của tư duy giáo điều:** "Nguy hiểm nhất là khi bạn nghĩ rằng bạn biết mình đang làm gì" – thật sự rất chính xác. Là lập trình viên, tôi cũng từng tự hào khi nắm vững một ngôn ngữ nào đó, và chính sự tự hào đó đã khiến tôi trì trệ khi cần học một ngôn ngữ mới. Thật tuyệt vời, nếu không có những người như Bret Victor, sẵn sàng chất vấn, sẵn sàng tưởng tượng một thế giới khác, thì có lẽ lập trình đã không tiến xa như bây giờ.

Когда я посмотрел доклад The Future of Programming Бретта Виктора, я был очень удивлён тем, как он стал инженером в 1973 года и поставил под сомнение предположения, которые мы привыкли считать самоочевидными в современном программировании, чтобы спрогнозировать будущее этой области. Он выдвинул очень сильный аргумент — именно сомнения и ограниченность человеческого мышления являются препятствием для развития новых технологий.

Бретт в роли инженера 1973 года, когда компьютеры ещё были примитивными, представил себе будущее, в котором нам больше не нужно писать код так, как мы привыкли. Он выдвинул концепцию «непосредственного взаимодействия с данными», которую считает прорывной. В качестве примера он привёл Sketchpad Ивана Сазерленда — систему, которая позволяла пользователям рисовать и накладывать ограничения, не написав ни одной строки кода. Если посмотреть на современность, можно сказать, что эта идея частично воплощена в таких графических инструментах, как Figma. Однако я не думаю, что человечество сможет полностью отказаться от написания исходного кода. Эти инструменты лишь решают задачи на поверхностном уровне; когда возникает необходимость реализовать сложную бизнес-логику или инновационные проекты (а иногда даже технологии, которых ещё не существует), нам всё равно приходится возвращаться к написанию кода.

Далее Бретт рассматривает программирование, ориентированное на цели и ограничения, где разработчик должен только описать, чего он хочет достичь, а система сама найдёт способ, как это сделать. Он упоминает Prolog, а также концепцию constraint programming. Это очень интересно с научной точки зрения, но с психологической стороны существует большая проблема: человек всегда хочет контролировать происходящее. Разработчики, как и все люди, боятся потерять контроль. Могут ли они доверить системе «думать за себя»? Это требует высокого уровня доверия к инструментам, которые могут быть изменены из-за множества факторов (например, хакеры, вирусы).

Бретт также говорит о пространственном представлении информации. Пользовательский опыт при этом становится более наглядным и понятным. Однако это не всегда эффективно. По моему опыту, поиск ошибок в таких графических системах может быть даже сложнее, чем отладка обычного исходного кода. Я согласен с Бреттом, что нам нужно больше, чем просто писать текст в файлах, но при этом текст имеет свои неоспоримые преимущества: его легко хранить, искать и контролировать с помощью таких инструментов, как Git. Думаю, в будущем нас ждёт не исключение одного подхода другим, а скорее их интеграция.

Параллельное программирование и распределённые системы — вот то, что, по мнению Бретта Виктора, должно кардинально изменить проектирование программного обеспечения. Он резко критикует модель потоков и блокировок, подчёркивая, что если через 40 лет мы всё ещё будем использовать потоки для управления конкурентностью, значит, «мы потерпели неудачу». Забавно, что прошло ровно 40 лет, а большая часть популярных программ всё ещё работает на потоках и блокировках! Однако это не делает его утверждение ложным. Модель акторов (Actor Model), CSP и экосистема микросервисов действительно открыли новую эру параллельного и распределённого программирования. Просто эти подходы ещё не полностью вытеснили старые методы. И, по моему мнению, это даже не обязательно — они могут сосуществовать, служа разным целям.

Хотя я не полностью согласен со всеми аргументами Бретта Виктора, его основная мысль о риске стать жертвой догматического мышления абсолютно точна. «Самое опасное — думать, что ты знаешь, что делаешь» — это высказывание действительно заставляет задуматься. Будучи программистом, я и сам когда-то гордился тем, что владею каким-то языком программирования, и именно эта гордость мешала мне осваивать новые технологии. Было бы замечательно, если бы не было таких людей, как Бретт Виктор, которые готовы ставить под сомнение общепринятое и воображать совершенно другой мир — тогда, возможно, программирование так бы и не продвинулось вперёд.