

Mô hình tư duy(Mental Model)

- Giới thiệu
- Mô hình tư duy là gì
- Làm thế nào để đạt được mô hình tư duy chung

Giới thiệu

Chúng ta sẽ cùng khám phá

1. Mô hình tư duy & tầm quan trọng của chúng trong thiết kế phần mềm có tính đến chiến lược kinh doanh

- Module đầu tiên giải thích khái niệm mô hình tư duy, lý do cần xây dựng mô hình tư duy chia sẻ
- Cách thiết lập mô hình tư duy vững chắc

2. Đào sâu Strategic Domain-Driven Design

- Các khái niệm: Subdomain (Tên miền phụ), Bounded Context (Ngữ cảnh giới hạn)
- Phát triển Ngôn ngữ phổ quát - yếu tố then chốt cho giao tiếp nhóm hiệu quả
- Khám phá 2 biến thể Event Storming: Big Picture (Toàn cảnh) và Design Level (Mức độ thiết kế)
- Cách dẫn xuất Bounded Context từ các phiên thảo luận

3. Nguyên tắc Hexagonal & Clean Architecture kết hợp Tactical DDD

- Triển khai kiến trúc thông qua các mẫu thiết kế chiến thuật như Aggregate (Tập hợp), Value Object (Đối tượng giá trị)
- Xây dựng nền tảng vững chắc cho quy trình thiết kế và phát triển phần mềm

4. Kiểm thử Use Case Clean Architecture

- Đa dạng phương pháp Test-Driven Development (TDD) và Refactoring

Mỗi module được thiết kế nhằm giúp bạn thấu hiểu và ứng dụng Domain-Driven Design một cách sâu sắc.

Mô hình tư duy là gì

(Mô hình tư duy(Mental Model)là cách mà một người **tưởng tượng, hiểu và lý giải cách một hệ thống, quy trình, hoặc một lĩnh vực hoạt động.**

Nói đơn giản, đó là **bức tranh trong đầu mỗi người** về thế giới thật — và bức tranh này có thể **khác nhau giữa từng người.**

Trong bối cảnh Domain-Driven Design (DDD):

- Mô hình tư duy là cách mà **các bên liên quan (domain expert, developer, khách hàng)** hiểu về domain (lĩnh vực nghiệp vụ).
- Nếu mỗi người có một Mô hình tư duy khác nhau, dễ dẫn đến **hiểu nhầm, thiết kế sai,** hoặc **phần mềm không đáp ứng đúng nhu cầu thực tế.**

Ví dụ

Vai trò	Mental Model về “Lô hàng”
Nhân viên kho	Một lần xe chở hàng vào kho = 1 lô hàng
Kế toán	Một đơn hàng có thể nhập thành nhiều lô
Lập trình viên (dev)	Một dòng trong bảng <code>Shipments</code> trong DB

Nếu không trao đổi để thống nhất, thì phần mềm sẽ **thiết kế sai với thực tế.**

Tại sao Mental Models quan trọng trong DDD?

1. Mô hình tư duy & sự đồng bộ trong thiết kế

- Việc xây dựng **mô hình tư duy (mental model)** chung giúp **đồng bộ hóa thiết kế phần mềm** với lĩnh vực nghiệp vụ.
- Trong thực tế, chúng ta thường gặp tình huống phải **diễn đạt lại ý tưởng** hoặc tiếp cận từ góc nhìn khác để được hiểu đúng – điều này phơi bày **sự phức tạp của giao tiếp.**
- Trong DDD, **khác biệt về mô hình tư duy** giữa các bên liên quan có thể dẫn đến phần mềm **lệch chuẩn nhu cầu nghiệp vụ.**

2. Vai trò then chốt của mô hình tư duy

- Cách nó **làm cầu nối** giữa **góc nhìn kỹ thuật** và **yêu cầu nghiệp vụ**
- Hiểu rõ **động lực giao tiếp** thông qua mô hình tư duy chung giúp phát triển phần mềm **sát với mục tiêu nghiệp vụ và nhu cầu người dùng.**

3. Thách thức trong giao tiếp

- **Truyền đạt sai** dẫn đến **hiểu nhầm yêu cầu**, **sai lệch mục tiêu**, và cuối cùng là phần mềm **không đáp ứng nhu cầu**.
- Thành công phụ thuộc vào **kinh nghiệm chia sẻ** giữa các thành viên. Mỗi tương tác góp phần xây dựng **mô hình tư duy chung**, giúp hiểu thông tin sâu sắc hơn.
- Trong môi trường đa ngành (như phát triển phần mềm), mô hình tư duy chung **thu hẹp khoảng cách** giữa đội ngũ kỹ thuật và Các bên liên quan(stakeholders).

4. Rào cản từ phụ thuộc vào tài liệu

- **Tài liệu chi tiết** không thể thay thế **giao tiếp trực tiếp**. Nếu người đọc không chia sẻ mô hình tư duy với tác giả, dễ xảy ra hiểu lầm.
- Ở các tập đoàn lớn, **khoảng cách giữa phòng kinh doanh và kỹ thuật** càng trầm trọng – không chỉ về địa lý mà còn về **tư duy**.

5. Xung đột giữa mục tiêu nghiệp vụ và kỹ thuật

- **Business stakeholders** tập trung vào ROI, trải nghiệm khách hàng.
- **Kỹ sư** quan tâm đến kiến trúc, công nghệ.
- Sự khác biệt này gây khó khăn trong việc **chuyển đổi nhu cầu kinh doanh thành giải pháp kỹ thuật**.

6. Hạn chế của mô hình truyền thống (Waterfall)

- Phân tách rõ ràng các giai đoạn (requirements, design, dev, test) làm **trầm trọng hóa khoảng cách**.
- Business analyst đóng vai trò "phiên dịch", nhưng việc **chuyển đổi tầng tầng lớp lớp** dễ gây hiểu sai.
- Giả định rằng **kỹ sư có thể tự hiểu vấn đề nghiệp vụ phức tạp** là sai lầm.

7. Hậu quả của thiếu đồng bộ

- **Lệch mục tiêu**: Dev tập trung vào công nghệ mới thay vì nhu cầu nghiệp vụ → phần mềm không giải quyết đúng vấn đề.
- **Tổn tài nguyên**, sản phẩm lỗi, **phải làm lại**.
- **Mất niềm tin**: Stakeholder siết kiểm soát → đội ngũ tập trung vào "hoàn thành task" thay vì chất lượng.
- **Bỏ qua testing/refactor** → rủi ro hệ thống sụp đổ.

8. Giải pháp then chốt

- **Mô hình tư duy chung** là chìa khóa.
- **Giao tiếp trực tiếp**, minh bạch, xây dựng niềm tin.
- **Tránh phụ thuộc quá mức vào tài liệu** – chỉ giao tiếp mới phá vỡ rào cản.

9. Kết luận

- **Cân bằng mô hình tư duy** ngăn thất bại dự án, đảm bảo phần mềm **linh hoạt, đúng nhu cầu, dễ bảo trì**.

Câu hỏi mở: Làm thế nào để đạt được sự đồng bộ này?

Tác giả: Đỗ Ngọc Tú
Công Ty Phần Mềm VHTSoft

Làm thế nào để đạt được mô hình tư duy chung

Việc đạt được mô hình tư duy chung(shared mental model) là **một trong những mục tiêu quan trọng nhất trong DDD** — và cũng là một trong những **khó khăn lớn nhất** nếu không làm đúng cách.

Những rào cản lớn để đạt được

Rào cản	Tác động
Thiếu giao tiếp giữa dev và domain expert	Không hiểu đúng nghiệp vụ
Ngôn ngữ không thống nhất	Dễ gây nhầm lẫn khi thảo luận hoặc code
Developer chỉ tập trung vào kỹ thuật	Bỏ qua logic nghiệp vụ, hiểu sai bản chất
Domain expert không quen mô hình hóa	Không diễn đạt đúng ý hoặc bỏ sót ý quan trọng
Áp lực deadline → code trước, hiểu sau	Dẫn đến phần mềm xa rời thực tế
Không ghi chú/tài liệu hoá rõ ràng	Không lưu giữ tri thức đã chia sẻ

Agile - Giải pháp thay thế cho mô hình truyền thống

Tuyên ngôn Agile đề xuất chuyển dịch:

- Từ **phát triển dựa trên tài liệu** sang **cách tiếp cận tập trung vào hợp tác**
- Thoát khỏi tư duy **tổ chức phân mảnh (siloeed organization)**
- Xây dựng **niềm tin** và **chia sẻ kinh nghiệm**

2 giá trị cốt lõi của Agile giải quyết thách thức giao tiếp

1. **Ưu tiên tương tác giữa con người** hơn quy trình hoặc công cụ
2. **Hợp tác với khách hàng** thay vì đàm phán hợp đồng cứng nhắc
→ Cả hai nhấn mạnh **làm việc chặt chẽ với khách hàng** để xây dựng **mô hình tư duy chung**

Cách Agile xây dựng mô hình tư duy chung

- **Giao tiếp mở và thường xuyên** giữa kỹ sư và khách hàng để giải quyết bất đồng
- **Phương pháp lặp (iterative)**: Như Scrum hay XP (Extreme Programming) tập trung vào:
 - Phản hồi liên tục từ khách hàng/đội ngũ
 - Kế hoạch linh hoạt thay vì chi tiết cứng nhắc
- **Học tập tương tác**: Hiểu sâu vấn đề của người dùng thông qua:
 - Thảo luận trực tiếp
 - Mockup/prototype đơn giản→ Kích thích trao đổi và điều chỉnh sớm

Vai trò của Prototyping

- **Prototype song song**: Thử nghiệm nhiều giải pháp cùng lúc để tìm phương án tối ưu
- **Tiến hóa từ mockup thô → sản phẩm tinh chỉnh**:
 - Mỗi vòng lặp làm sâu sắc thêm hiểu biết về vấn đề
 - Kế hoạch luôn giữ tính linh hoạt

Cơ chế phản hồi trong Agile

1. **Lập trình cặp (Pair Programming)/Mob Programming**:
 - Kết hợp góc nhìn đa dạng → Hiểu biết tập thể
 - Khuyến khích **khách hàng tham gia trực tiếp** để giảm hiểu lầm
2. **Phát triển hướng kiểm thử (TDD)**:
 - Viết test trước code → Đảm bảo phần mềm đáp ứng nghiệp vụ
 - Test nên viết từ góc độ **yêu cầu kinh doanh**, không chỉ kỹ thuật
3. **Stand-up Meeting**:
 - Giải quyết bất đồng ngay lập tức
 - Duy trì sự đồng bộ trong team
4. **Planning Meeting**:
 - Sự tham gia của khách hàng → Xác nhận mục tiêu chung

Kết quả đạt được

- Phần mềm **linh hoạt, đúng nhu cầu** nhờ:
 - Giao tiếp liên tục
 - Phản hồi theo vòng lặp
- **Tài liệu sống động** (living documentation) thay vì tĩnh

Kết nối giữa Agile và Domain-Driven Design (DDD)

- **DDD bổ sung cho Agile** bằng:
 - **Ubiquitous Language**: Ngôn ngữ chung giữa kỹ thuật & nghiệp vụ
 - **Bounded Context**: Xác định phạm vi rõ ràng
 - **Event Storming**: Công cụ trực quan hóa bài toán nghiệp vụ

