責任、模組與契約：從檔案分類到系統邊界

Responsibility, Modules, and Contracts: From File Classification to System Boundaries

v0.1 / 2026-07-27 / Neo.K with Aletheia / 初版完成

https://thisoneisneok.com/html/papers/pu-1-04.html

SECTION

Responsibility, Modules, and Contracts: From File Classification to System Boundaries

論文編號： PU-1-04

系列：「程式宇宙書系」第 1 冊《系統性程式設計》

作者： Neo.K with Aletheia

機構： EveMissLab／一言諾科技有限公司

版本： v0.1

日期： 2026 年 7 月 27 日

SECTION

摘要

前兩篇分別建立了問題世界模型，以及資料、狀態、事件、行動的四元動力結構。本篇處理系統設計的下一個核心問題：當世界中已有實體、狀態、事件與行動後，究竟由誰負責維持它們？哪些狀態屬於哪個模組？模組之間應交換命令、事件、查詢、資料還是證據？跨邊界失敗由誰承擔？契約又應包含哪些內容？

本文提出：

> 【Module
> ≠
> Folder

資料夾是檔案組織；套件是語言或建置單位；程序是執行單位；服務是可被網路或程序邊界存取的運作單位；部署單元是發布與資源配置單位。模組則是對一組問題世界狀態、規則與失敗承擔一致責任的語意邊界。

本文將模組定義為：

> 【mathcal M
> =
> <
> W_M,
> S_M,
> I_M,
> A_M,
> E_M,
> C_M,
> F_M,
> O_M,
> D_M
> >】

其中：

- W_M：模組負責的問題世界切片；
- S_M：模組擁有的權威狀態；
- I_M：模組必須維持的不變量；
- A_M：模組接受的行動與命令；
- E_M：模組發布的事件；
- C_M：模組對外契約；
- F_M：模組承擔的失敗、補償與恢復；
- O_M：可觀測性與證據；
- D_M：外部依賴與非責任。

本文提出「權威狀態單一責任原則」：

> 每一項重要世界狀態，必須具有唯一可識別的權威責任者；它可以被複製、快取與投影，但不能由多個模組在沒有明示協議下共同直接改寫。

形式上：

> ∀ s∈ S_(critical),
>
> ∃ ! M:
> Owner(M,s)

此處的唯一不是要求單一機器、單一資料庫或單一程序，而是要求單一可追責的語意責任。分散式複本可以很多，但誰決定狀態合法、誰維持不變量、誰處理衝突，必須清楚。

本文將責任定義為：

> mathcal R_s
> =
> <
> Purpose,
> StateOwnership,
> DecisionAuthority,
> Obligation,
> FailureOwnership,
> Evidence,
> NonResponsibility
> >

責任不只是「這個類別做什麼」，而是對目的、狀態、決策權、義務、失敗、證據與非責任的整體承諾。

本文進一步提出完整契約模型：

> 【mathcal K_(AB)
> =
> <
> Intent,
> Input,

契約不等於函式簽章、API schema 或 JSON 格式。型別只能描述資料形狀，完整契約還必須說明：

- 為何呼叫；
- 呼叫前世界必須如何；
- 誰有權要求；
- 成功會改變什麼；
- 會產生哪些事件；
- 失敗如何分類；
- 是否可重試與冪等；
- 時間與期限；
- 何種證據算完成；
- 契約版本與相容；
- 誰能修改契約及如何申訴。

本文區分跨模組互動的五種基本形式：

- Query：要求資料投影，不應改變世界；
- Command／Action：要求另一模組執行世界變更；
- Event：宣告某項已發生事實；
- Evidence：提供完成、驗證或責任證明；
- Shared State：多方直接存取同一狀態，僅在責任與一致性協議明確時使用。

本文主張，共享資料庫不是免費整合。當多個模組直接修改同一資料表時，實際上形成隱藏共同責任：

> 【Shared Write
> =
> Shared Invariant Responsibility】

如果團隊無法共同說明不變量、衝突、回復與版本，共享寫入便只是邊界消失。

本文亦分析同步呼叫、非同步命令、事件發布、共享狀態與人工工作流。不同形式不只是技術選擇，而是對時間、責任、可用性與失敗傳播方式的選擇。同步呼叫讓依賴立即可見，卻容易形成時間與故障耦合；非同步互動提高解耦與恢復能力，卻需要顯式處理延遲、重複、順序與最終一致；事件發布降低發布者對訂閱者的依賴，卻不能用事件逃避對結果的責任。

本文提出「契約邊界測試」：一個候選模組若無法獨立回答以下問題，就尚未形成真正邊界：

- 它擁有什麼狀態？
- 它維持什麼不變量？
- 它接受哪些行動？
- 它發布哪些事件？
- 它承擔哪些失敗？
- 它提供什麼證據？
- 它依賴誰？
- 它明確不保證什麼？

本文使用訂單與付款、帳號與權限、網站部署、AI Agent 工具層、研究工作流與多團隊平台等案例，說明責任如何決定模組，而不是由檔案、類別、微服務數量或組織圖自動決定。

本文最後提出可證偽研究綱領，包括共享寫入事故率、狀態擁有者辨識率、契約完整度、跨邊界失敗傳播、同步耦合成本、事件濫用率、模組交接時間、責任圖與依賴圖差異、契約版本破壞率及模組邊界與團隊邊界的關係。

本文為下一篇〈可視化作為外部結構記憶〉建立組織骨架：當責任、狀態與契約被明確後，系統才能被不同角色、不同尺度與不同時間共同看見，而不再依賴少數工程師腦內的隱性架構。

關鍵詞： 模組、責任、契約、狀態擁有權、系統邊界、共享資料、同步耦合、事件驅動、軟體架構

SECTION

Abstract

This paper defines modules as semantic boundaries of responsibility rather than folders, packages, processes, services, or deployment units.

A module is modeled as:

> mathcal M
> =
> <
> W_M,
> S_M,
> I_M,
> A_M,
> E_M,
> C_M,
> F_M,
> O_M,
> D_M
> >

Each critical state must have one identifiable authoritative responsibility, even if it is replicated across multiple machines.

The paper develops a full contract model that includes intent, input, preconditions, authorization, effects, events, failures, retries, time, evidence, versioning, and governance. It distinguishes queries, commands, events, evidence, and shared state, and analyzes synchronous, asynchronous, event-driven, shared-state, and human-mediated interactions.

The paper argues that module boundaries should be derived from state ownership, invariants, decisions, failures, and evidence rather than file structure or deployment topology.

Keywords: modules, responsibility, contracts, state ownership, system boundaries, event-driven architecture

很多專案以下列方式拆分：

```
controllers/
services/
models/
utils/
```

repositories/
或拆成：

user-service
order-service
payment-service
notification-service
但名稱相似不代表責任清楚。

一個名為 order-service 的服務可能：

- 直接改付款資料；
- 直接扣庫存；
- 直接寄信；
- 直接修改會員狀態；
- 同時維護多個互不相干不變量。

  反之，一項真正完整的「訂單責任」也可能分散在多個資料夾、程序與部署單元中。

  因此，模組不能由檔案或部署形式定義。

  它必須由責任定義。

SECTION

2.1 模組不等於資料夾

資料夾只表示檔案被放在一起。

SECTION

2.2 模組不等於套件

套件是語言、建置或發布機制。

SECTION

2.3 模組不等於類別

單一類別通常只承擔局部行為或資料結構。

SECTION

2.4 模組不等於程序

程序是 Runtime 隔離與資源單位。

SECTION

2.5 模組不等於服務

服務可以承載一個或多個模組，也可能只是一個技術接口。

SECTION

2.6 模組不等於部署單元

同一模組可以被多個部署實例承載；多個模組也可以暫時共同部署。

因此：

> 【Semantic Boundary
> ≠
> Physical Deployment Boundary】

本文定義：

> 【mathcal M
> =
> <
> W_M,
> S_M,
> I_M,
> A_M,
> E_M,
> C_M,
> F_M,

SECTION

3.1 問題世界切片 W_M

模組負責理解哪一部分世界。

SECTION

3.2 權威狀態 S_M

哪些狀態由此模組決定與維護。

SECTION

3.3 不變量 I_M

無論如何轉移都必須成立的條件。

SECTION

3.4 接受行動 A_M

外部主體可以要求此模組做什麼。

SECTION

3.5 發布事件 E_M

模組會對外宣告哪些已發生事實。

SECTION

3.6 對外契約 C_M

其他模組如何合法互動。

SECTION

3.7 失敗責任 F_M

哪些失敗由本模組重試、補償、回復或升級。

SECTION

3.8 觀測與證據 O_M

如何證明模組做了什麼、目前如何、為何失敗。

SECTION

3.9 相依與非責任 D_M

模組依賴誰，以及不保證什麼。

本文定義：

> mathcal R_s
> =
> <
> Purpose,
> StateOwnership,
> DecisionAuthority,
> Obligation,
> FailureOwnership,
> Evidence,
> NonResponsibility
> >

SECTION

4.1 目的

模組為何存在，試圖維持或改變哪一部分世界。

SECTION

4.2 狀態擁有權

哪些權威狀態由此模組維護。

SECTION

4.3 決策權

哪些世界判定只有此模組能正式作出。

SECTION

4.4 義務

模組收到合法行動後必須提供什麼回應、狀態與證據。

SECTION

4.5 失敗擁有權

失敗發生後，誰負責：

- 重試；
- 回復；
- 補償；
- 升級；
- 通知；
- 人類接管。

SECTION

4.6 證據

模組如何證明其決策與效果。

SECTION

4.7 非責任

模組明確不保證什麼。

非責任不是卸責，而是防止邊界外效果被虛假承諾。

本文提出：

> ∀ s∈ S_(critical),
>
> ∃ ! M:
> Owner(M,s)

SECTION

5.1 唯一責任不等於單一副本

一項狀態可以具有：

- 主副本；
- 唯讀複本；
- 快取；
- 搜尋索引；
- 分析投影；
- 離線副本。

但必須知道哪個模組有權判定其合法狀態。

SECTION

5.2 單一責任不等於單一人

責任可由團隊、制度與 Runtime 共同承擔。

重點是責任位置可識別。

SECTION

5.3 多模組直接寫入

若多個模組直接寫入同一狀態，則必須共同維持所有相關不變量。

這通常意味著它們尚未真正分離。

SECTION

5.4 狀態變更應經由擁有者

外部模組不應直接修改另一模組的權威狀態，而應提出行動：

> M_A
> xrightarrowCommand
> M_B
> xrightarrowDecision
> E_B

SECTION

5.5 讀取與寫入不同

其他模組可以讀取投影，但必須知道：

- 是否過時；

- 更新延遲；
- 可否用於決策；
- 是否需要再次向擁有者確認。

SECTION

6.1 不變量

不變量是模組必須始終維持的世界條件。

例如：

- 同一訂單不得被重複結算；
- 未授權主體不能修改權限；
- 發布版本必須對應唯一建置產物；
- Agent 不得超出委任權限。

SECTION

6.2 凝聚力

若一組狀態與行動共同維持同一組不變量，它們通常應位於相同責任邊界。

SECTION

6.3 切斷不變量

若將一項必須原子維持的不變量拆到多個模組，系統會產生：

- 雙重寫入；
- 補償；
- 同步鎖定；

- 分散式交易；
- 競爭；
- 不一致窗口。

SECTION

6.4 過度聚合

反之，把沒有共同不變量的責任放在同一模組，會造成：

- 修改互相影響；
- 測試範圍擴張；
- 權限過大；
- 部署耦合；
- 團隊知識過載。

SECTION

6.5 模組邊界問題

因此模組設計不是追求最大或最小，而是尋找：

> 【High Internal Responsibility Coherence
> +
> Explicit External Contracts】

定義責任圖：

> G_R
> =
> (
> M,
> K,
> O,

其中：

- M：模組節點；
- K：契約邊；
- O：狀態與決策擁有關係；
- D：相依與失敗傳播。

SECTION

7.1 責任圖不同於呼叫圖

呼叫圖描述誰呼叫誰。

責任圖描述：

- 誰擁有狀態；
- 誰作出決策；
- 誰承擔失敗；
- 誰提供證據。

SECTION

7.2 呼叫方向不等於責任方向

前端呼叫付款模組，不代表前端對付款合法性負責。

SECTION

7.3 資料流不等於權威流

資料從 A 流向 B，不代表狀態所有權也轉移。

SECTION

7.4 失敗傳播

責任圖應標示：

- 哪些失敗可局部吸收；
- 哪些需要上游重試；
- 哪些需要補償；
- 哪些需要人工介入。

SECTION

8.1 Query

Query 要求資料投影：

> Q
> :
> M_A
> →
> M_B

原則上不應改變問題世界。

Query 契約需說明：

- 資料是否權威；
- 新鮮度；
- 分頁；
- 權限；
- 遺漏；

- 不確定性。

SECTION

8.2 Command／Action

要求另一模組執行狀態變更。

命令以祈使語意命名：

AuthorizePayment
CreateAccount
DeployRelease
RevokeAgentPermission
被要求者可以接受、拒絕、延遲或進入人工審核。

SECTION

8.3 Event

宣告已發生事實：

PaymentAuthorized
AccountCreated
ReleaseDeployed
AgentPermissionRevoked
事件發布者對事件真實性負責，但不應命令所有訂閱者如何反應。

SECTION

8.4 Evidence

證據支援：

- 完成；
- 驗證；
- 稽核；
- 外部效果；

- 責任判定。

  證據不是普通資料附件，而是契約完成判準的一部分。

SECTION

8.5 Shared State

多個模組直接讀寫同一狀態。

共享狀態需要明示：

- 權威；
- 鎖定；
- 衝突；
- 不變量；
- 遷移；
- 稽核；
- 失敗回復。

  否則共享只是隱藏共同責任。

  本文提出：

> 【Shared Write
> =
> Shared Invariant Responsibility】

SECTION

9.1 表面便利

共享資料庫能快速：

- JOIN；
- 直接更新；
- 建立報表；
- 跨模組交易。

SECTION

9.2 隱藏代價

多個模組直接寫入同一表會使：

- schema 成為隱藏公共 API；
- 修改需要多方協調；
- 不變量分散；
- 權限模糊；
- 刪除與遷移困難；
- 測試必須跨模組。

SECTION

9.3 唯讀共享

唯讀分析與投影可以共享，但要標示：

- 新鮮度；
- 權威來源；
- 不可反向寫入；
- 敏感欄位；
- 版本。

SECTION

9.4 合法共享狀態

在以下條件下，共享狀態可能合理：

- 真正共同責任；
- 同一交易邊界；
- 明示一致性協議；
- 單一治理機制；
- 失敗與回復共同設計。

  函式簽章可能描述：

  authorizePayment(orderId: string, amount: number): Promise<Result>
  但它沒有完整說明：

- 為何可呼叫；
- 誰能呼叫；
- 訂單必須處於何種狀態；
- 金額以何者為權威；
- 超時後能否重試；
- 成功是否代表銀行結算；
- 會發布哪些事件；
- 需要保留什麼證據；
- 版本如何演化。

  所以型別只描述契約的一部分。

  本文定義：

> 【mathcal K_(AB)
> =
> <
> Intent,
> Input,
> Precondition,
> Authorization,
> Effect,
> Event,
> Failure,
> Retry,
> Time,
> Evidence,
> Version,
> Governance
> >】

SECTION

11.1 Intent

為何存在此契約，它服務哪個問題世界目的。

SECTION

11.2 Input

需要哪些資料、身份與上下文。

SECTION

11.3 Precondition

呼叫前世界必須滿足何種狀態。

SECTION

11.4 Authorization

哪些主體可在何種情境下使用契約。

SECTION

11.5 Effect

成功後會改變哪些權威狀態與外部世界。

SECTION

11.6 Event

成功、拒絕、部分成功或補償後會發布哪些事件。

SECTION

11.7 Failure

失敗類型及其語意：

- 驗證失敗；
- 權限拒絕；
- 暫時不可用；
- 永久拒絕；
- 部分成功；
- 結果未知。

SECTION

11.8 Retry

是否可重試、使用何種冪等身份、何時停止。

SECTION

11.9 Time

期限、超時、生效時間、資料新鮮度與最大延遲。

SECTION

11.10 Evidence

什麼證據足以證明契約已履行。

SECTION

11.11 Version

schema、語意、規則與相容策略。

SECTION

11.12 Governance

誰能修改契約、如何審查、通知、異議與淘汰。

SECTION

12.1 結構契約

描述資料形狀、型別與必要欄位。

SECTION

12.2 語意契約

描述欄位與操作在問題世界中的意義。

例如 status = paid 究竟表示：

- 付款請求已送出；
- 付款已授權；
- 款項已結算；
- 訂單可出貨；

  必須明確。

SECTION

12.3 操作契約

描述：

- 延遲；
- 可用性；
- 重試；
- 順序；
- 冪等；
- 配額；
- 失敗模式。

SECTION

12.4 治理契約

描述：

- 所有者；

- 版本；
- 棄用；
- 法律與權利；
- 稽核；
- 申訴；
- 人類接管。

只有四層同時成立，契約才不會只停留在資料格式。

SECTION

13.1 同步請求—回應

上游等待下游立即回覆：

> M_A
> xrightarrowrequest
> M_B
> xrightarrowresponse
> M_A

SECTION

13.2 優點

- 控制流清楚；
- 即時結果；
- 錯誤容易返回；
- 適合短時判定；
- 開發與除錯較直接。

SECTION

13.3 代價

- 時間耦合；
- 故障傳播；
- 延遲累積；
- 呼叫鏈放大；
- 下游不可用使上游停滯。

SECTION

13.4 同步不代表一致完成

下游回覆成功，可能只表示其內部階段完成。

上游仍須理解回覆的證據層級。

SECTION

13.5 適用情境

同步較適合：

- 需要立即拒絕；
- 短時間可完成；
- 呼叫者必須根據結果繼續；
- 依賴數量有限；
- 失敗語意可清楚返回。

SECTION

14.1 形式

> M_A
> xrightarrowCommand
> Queue
> xrightarrow
> M_B

發送者不等待最終完成。

SECTION

14.2 優點

- 時間解耦；
- 緩衝尖峰；
- 下游可恢復處理；
- 適合長時工作；
- 可建立重試與死信流程。

SECTION

14.3 代價

- 完成延遲；
- 重複；
- 亂序；
- 追蹤困難；

- 使用者需要中間狀態；
- 結果可能最終失敗。

SECTION

14.4 接受不等於完成

命令入列只代表：

accepted-for-processing
不代表世界效果已完成。

SECTION

14.5 命令所有權

發送者對「為何要求」負責。

接收者對「是否接受、如何執行與維持不變量」負責。

SECTION

15.1 形式

M_A
xrightarrowEvent
{
M_B,
M_C,
M_D
}

SECTION

15.2 發布者責任

發布者必須保證：

- 事件確實成立；
- 身份可去重；
- 語意與版本清楚；
- 重要證據可取得；
- 發布與狀態一致。

SECTION

15.3 訂閱者責任

訂閱者決定：

- 是否處理；
- 如何處理；
- 是否重試；
- 如何維持自己的不變量。

SECTION

15.4 事件不應偽裝成命令

若事件名稱是：

SendEmailNow
UpdateInventoryImmediately
它其實仍在要求特定下游行動。

真正事件應描述已發生事實。

SECTION

15.5 事件不能逃避責任

發布事件後，原模組不能宣稱：

我已經發事件了，
後面發生什麼不關我的事。
若原目的包含完整工作流，它仍需追蹤整體責任與完成證據。

SECTION

16.1 人類也是契約參與者

部分決策需要：

- 專業判斷；
- 正當程序；
- 不可代理選擇；
- 例外；
- 高風險批准。

SECTION

16.2 人工節點不是空白

應正式建模：

- 任務；
- 角色；
- 期限；
- 所需證據；
- 可選結果；
- 未回應；
- 委任；

- 升級。

SECTION

16.3 人工超時

人類未回覆不是成功，也不一定是拒絕。

需保存：

awaiting-human
human-timeout
escalated
withdrawn

SECTION

16.4 人機契約

系統應讓人類知道：

- 為何需要判斷；
- 系統已知什麼；
- 未知什麼；
- 決策後果；
- 是否可撤回。

SECTION

17.1 契約具有時間

需要說明：

- 何時生效；
- 何時過期；
- 最長處理時間；

- 資料新鮮度；
- 回應與最終完成差異。

SECTION

17.2 超時不是單一失敗

超時可能表示：

- 尚未開始；
- 正在處理；
- 已成功但回覆遺失；
- 下游不可用；
- 結果未知。

SECTION

17.3 結果未知

unknown-outcome 應成為正式狀態，避免盲目重試。

SECTION

17.4 服務水準不是世界保證

「99.9% 可用」不代表每一項世界效果都正確完成。

SECTION

18.1 結構相容不等於語意相容

新增可選欄位可能結構相容，但若欄位改變決策語意，仍可能破壞契約。

SECTION

18.2 契約版本

需版本化：

- schema；
- 語意；
- 規則；
- 事件；
- 錯誤；
- 權限；
- 時間承諾。

SECTION

18.3 生產者與消費者相容

可分別檢查：

- 生產者是否仍產生消費者可理解資料；
- 消費者是否能忽略未知欄位；
- 舊事件是否仍可重播；
- 新規則是否錯誤套用舊歷史。

SECTION

18.4 棄用

契約棄用需要：

- 宣告；
- 期限；
- 替代方案；
- 遷移工具；
- 影響分析；
- 退出條件。

SECTION

19.1 失敗不是自動屬於下游

上游選擇依賴下游，因此不能把所有結果責任全部推給下游。

SECTION

19.2 三層責任

跨邊界失敗至少包含：

- 局部處理責任：哪個模組應修復自己的失敗；
- 流程協調責任：誰追蹤整體工作流；
- 使用者結果責任：誰向受影響主體解釋、補救與結案。

SECTION

19.3 失敗吸收

模組可在邊界內吸收：

- 暫時重試；
- 快取失效；

- 局部降級；
- 備援切換。

若失敗影響上游目的，則需對外發布或返回明確狀態。

SECTION

19.4 失敗升級

需要升級的情境包括：

- 重試耗盡；
- 結果未知；
- 部分成功；
- 不可逆效果；
- 契約違反；
- 高風險權限問題。

SECTION

19.5 失敗所有者

每一重大失敗應具有：

```
failure:
 local_owner: "payment-module"
 workflow_owner: "order-module"
 user_resolution_owner: "support-process"
```

SECTION

20.1 委任

模組可以將局部執行委任給另一模組、外部服務或 Agent。

SECTION

20.2 原始責任

若 A 對使用者承諾結果，再把工作委任給 B：

> A
> xrightarrowdelegate
> B

A 仍需對：

- 委任選擇；
- 追蹤；
- 證據；
- 替代；
- 補救；

  負責。

SECTION

20.3 接受責任

B 則對自己接受的契約、狀態與執行合法性負責。

SECTION

20.4 責任鏈

> Commitment
> →
> Delegation
> →
> Execution

任何斷點都應有責任位置。

本文提出九步模組拆分法。

SECTION

第一步：列出問題世界責任

不要先列技術層，而是列：

- 哪些世界結果必須被維持；
- 哪些決策需要一致語意。

SECTION

第二步：找出權威狀態

為每項重要狀態指定候選擁有者。

SECTION

第三步：找出不變量群

將共同維持同一不變量的狀態與行動聚集。

SECTION

第四步：列出接受行動

哪些外部意圖能進入此邊界。

SECTION

第五步：列出發布事件

哪些已發生事實對外具有意義。

SECTION

第六步：列出失敗與補償

誰負責重試、回復、升級與補救。

SECTION

第七步：建立契約

將輸入、前置條件、權限、效果、證據與時間明示。

SECTION

第八步：選擇物理部署

在語意邊界清楚後，再決定：

- 同程序；
- 不同套件；
- 不同服務；
- 不同資料庫；
- 不同團隊。

SECTION

第九步：持續以事故與變更校準

若跨邊界修改頻繁、契約大量洩漏或不變量持續被破壞，邊界需重新調整。

一個候選模組應能回答：

- 它為何存在？
- 它擁有什麼狀態？

- 它維持什麼不變量？
- 它接受哪些行動？
- 它發布哪些事件？
- 它拒絕哪些行動？
- 它如何證明成功？
- 它承擔哪些失敗？
- 它依賴哪些外部模組？
- 它明確不保證什麼？
- 契約如何版本化？
- 誰能修改其規則？

若無法回答，多半只是檔案集合或技術組件，尚未形成完整責任模組。

SECTION

23.1 過小模組

症狀：

- 一次功能需要穿越大量邊界；
- 事件與命令爆炸；
- 大量同步呼叫；
- 每次變更需協調多個團隊；
- 狀態不變量被切碎。

SECTION

23.2 過大模組

症狀：

- 內部責任互不相關；
- 權限過寬；
- 部署牽動所有功能；
- 單一團隊無法理解；
- 測試與修改成本持續增加。

SECTION

23.3 評估原則

模組大小應依：

- 不變量；
- 狀態生命週期；
- 決策語意；
- 失敗責任；
- 變更方向；

判定，而不是依程式碼行數。

SECTION

24.1 組織影響架構

團隊溝通結構會影響系統邊界。

SECTION

24.2 不應直接等同

組織圖可能因預算與人事改變，問題世界責任不應隨意破碎。

SECTION

24.3 團隊擁有權

每個模組應有明確維護團隊，但一個團隊可維護多個相近模組。

SECTION

24.4 跨團隊契約

跨團隊邊界需要更強：

- 文件；
- 版本；
- 可觀測；
- 相容測試；
- 事故責任；
- 變更通知。

SECTION

24.5 平台團隊

平台團隊應提供共通能力，但不應奪取所有領域狀態與決策權。

SECTION

25.1 訂單模組

擁有：

- 訂單生命週期；

- 客戶購買承諾；
- 履約進度；
- 取消與爭議。

SECTION

25.2 付款模組

擁有：

- 付款嘗試；
- 授權；
- 結算主張；
- 退款；
- 金流證據。

SECTION

25.3 互動

訂單模組提出：

AuthorizePayment
付款模組發布：

PaymentAuthorized
PaymentDeclined
PaymentOutcomeUnknown

SECTION

25.4 責任分離

付款模組不能直接把訂單設為 completed。

訂單模組也不能直接修改付款授權狀態。

SECTION

25.5 流程責任

訂單模組若向客戶承諾購買流程，仍需追蹤付款事件並處理結果，而不能只說「付款是另一個服務」。

SECTION

26.1 帳號模組

擁有：

- 帳號身份；
- 啟用；
- 暫停；
- 關閉；
- 驗證關係。

SECTION

26.2 權限模組

擁有：

- 角色；
- 委任；
- 資源存取；
- 權限期限；
- 撤銷。

SECTION

26.3 邊界

刪除帳號不應由權限模組直接完成。

權限模組可在帳號停用事件後撤銷存取。

SECTION

26.4 契約

必須定義帳號停用與權限撤銷之間：

- 延遲；
- 順序；
- 緊急封鎖；
- 證據；
- 失敗升級。

SECTION

27.1 Release 模組

擁有版本、建置產物與發布候選。

SECTION

27.2 Deployment 模組

擁有部署工作流、環境狀態、健康檢查與回滾。

SECTION

27.3 Traffic 模組

擁有流量切換與路由狀態。

SECTION

27.4 契約鏈

ApproveRelease
→ ReleaseApproved
→ DeployRelease
→ DeploymentVerified
→ ShiftTraffic
→ TrafficShifted

SECTION

27.5 完成責任

部署模組可證明服務啟動與健康檢查，但產品流程是否符合目的，仍需外部驗收。

SECTION

28.1 Agent 核心

擁有：

- 目標；
- 計畫；
- 記憶；
- 任務狀態；
- 人類委任。

SECTION

28.2 工具模組

各自擁有：

- 工具輸入驗證；
- 外部操作；
- 冪等；
- 證據；
- 局部錯誤。

SECTION

28.3 權限模組

獨立維持：

- 工具白名單；
- 作用域；
- 時間；
- 預算；
- 人類批准。

SECTION

28.4 工具結果

工具回傳資料不應直接被 Agent 當成世界完成事件，需依契約轉換成證據與狀態。

SECTION

28.5 不可逆操作

發布、付款、刪除與寄送等工具需要更高層契約與批准。

SECTION

29.1 文獻模組

擁有來源、引用與閱讀證據。

SECTION

29.2 命題模組

擁有定義、命題、依賴與證明狀態。

SECTION

29.3 實驗模組

擁有執行、參數、產物與結果證據。

SECTION

29.4 出版模組

擁有文件版本、格式、發布與外部識別。

SECTION

29.5 責任分離

文件發布不應直接把命題標記為已證明。

實驗結果也不應直接修改論文結論，而應發布證據事件供命題模組判定。

平台可能包含：

- 身份；
- 內容；

- 搜尋；
- 推薦；
- 計費；
- 通知；
- 稽核。

若所有團隊直接共享使用者、內容與權限資料表，會形成巨大隱藏共同模組。

更成熟的結構是：

- 各責任擁有權威狀態；
- 對外提供投影與契約；
- 跨域工作流明示協調者；
- 共同平台只提供不含領域決策的基礎能力。
- 資料夾即模組： 檔案放在一起就被視為責任一致。
- 服務即邊界： 建立網路服務便假定語意邊界已成立。
- 部署即責任： 同時部署的內容被迫共享所有責任。
- 類別即領域： 單一類別名稱被當成完整問題世界模組。
- 狀態無主： 重要狀態沒有唯一可識別的權威責任者。
- 多方直接寫入： 多模組任意修改同一權威狀態。
- 共享資料庫免費論： 忽略 schema、權限、不變量與遷移耦合。
- 唯讀投影僭位： 快取或查詢投影被拿來作權威判定。
- 契約等於 schema： 只有資料形狀，沒有語意、時間與失敗。
- 成功語意模糊： 回傳成功卻不說完成到哪一層。
- 命令事件混用： 以事件名稱要求下游執行特定命令。

- 事件逃責： 發布事件後便放棄整體流程責任。
- 同步鏈蔓延： 服務間長鏈呼叫造成故障與延遲放大。
- 非同步即解耦： 改用佇列卻未處理重複、亂序與追蹤。
- 超時即失敗： 不保留結果未知狀態而盲目重試。
- 委任即卸責： 上游把工作交給下游後不再追蹤承諾。
- 失敗無流程所有者： 各模組只處理局部錯誤，無人負責使用者結果。
- 補償責任不明： 部分成功後無模組決定是否補償。
- 契約永久不變： 沒有版本、棄用、遷移與治理。
- 結構相容迷思： JSON 可解析就宣稱語意相容。
- 模組過小： 不變量被切碎，跨邊界協調成本爆炸。
- 模組過大： 無關責任集中，權限與修改範圍失控。
- 組織圖即架構： 人事分工被直接固化為世界邊界。
- 平台奪取領域： 共通平台集中所有狀態與決策權。
- 人工流程空白化： 人類節點沒有期限、證據、狀態與責任。
- 非責任未聲明： 模組被迫對無法控制的外部效果作承諾。

SECTION

32.1 狀態擁有者辨識率

測量團隊成員能否一致回答每項關鍵狀態由哪個模組權威維護。

SECTION

32.2 共享寫入事故率

統計多模組直接寫入同一資料結構造成的：

- 不變量破壞；
- 資料競爭；
- 遷移失敗；
- 權限越界；
- 責任爭議。

SECTION

32.3 契約完整度

對契約十二元結構評分：

$$C_K = (|\text{declared contract dimensions}|)/(12)$$

研究完整度與事故率、整合時間、交接成本的關係。

SECTION

32.4 結構—語意相容差異

統計 schema 驗證通過但實際語意破壞的版本事件。

SECTION

32.5 同步耦合成本

測量呼叫鏈深度與：

- 延遲；
- 故障傳播；

- 部署協調；
- 重試放大；

之間的關係。

SECTION

32.6 非同步追蹤完整率

檢查非同步工作流是否能從原始意圖追蹤到最終事件、狀態與證據。

SECTION

32.7 事件濫用率

統計名為事件、實際要求特定下游行動的接口比例。

SECTION

32.8 失敗所有權覆蓋

檢查每類重大失敗是否具有：

- 局部所有者；
- 流程所有者；
- 使用者結案所有者。

SECTION

32.9 模組交接時間

比較責任與契約明確、以及只依賴檔案結構的專案，新成員理解與接手時間。

SECTION

32.10 責任圖—依賴圖差異

比較呼叫圖、部署圖與責任圖，研究哪些隱藏責任無法從技術拓撲看出。

SECTION

32.11 邊界變更率

記錄事故與需求變更中，有多少需要重新調整模組責任，而非只修改內部實作。

SECTION

32.12 契約版本破壞率

追蹤每次契約更新對既有消費者造成的結構、語意、時間與治理破壞。

SECTION

32.13 模組大小與變更擴散

研究模組內責任凝聚度、跨模組契約數量與修改影響範圍的關係。

SECTION

32.14 團隊—模組對齊效應

比較團隊與責任模組較一致、及高度交錯的組織，在交付、事故與知識集中上的差異。

- 模組是語意責任邊界，不是檔案、套件、程序、服務或部署單元。
- 模組必須能說明其問題世界、狀態、不變量、行動、事件、契約、失敗與證據。
- 每項關鍵狀態都需要唯一可識別的權威責任者。
- 單一責任不要求單一機器或單一資料副本。
- 外部模組應透過行動要求狀態擁有者修改權威狀態。
- 共同維持同一不變量的狀態與行動，通常應位於相同責任邊界。
- 呼叫圖、資料流圖與部署圖不能取代責任圖。

- Query、Command、Event、Evidence 與 Shared State 具有不同語意。
- Query 原則上不應改變問題世界。
- Command 是世界變更請求，Event 是已發生事實。

11.

> 【Shared Write
> =
> Shared Invariant Responsibility】

- 契約不等於函式簽章或資料 schema。
- 完整契約必須包含意圖、條件、授權、效果、失敗、時間、證據、版本與治理。
- 同步互動造成時間與故障耦合；非同步互動要求顯式處理延遲、重複與最終狀態。
- 發布事件不能消除原始承諾與流程責任。
- 人類工作流必須具備正式狀態、證據、期限與升級。
- 超時可能代表結果未知，而非單純失敗。
- 委任局部執行，不會自動消除原始責任。
- 跨邊界失敗需要局部、流程與使用者結果三層所有者。
- 模組大小應依不變量、狀態生命週期、決策語意與變更方向決定。
- 團隊邊界可以承接模組，但不應任意取代問題世界責任。

22.

> 【架構的核心不是把程式拆開，】

> 【而是讓每一項世界責任都有清楚的擁有者，】

> 【並讓責任之間只透過可理解、可驗證、】

> 【可演化的契約連接。】

SECTION

34.1 承接 PU-1-01

PU-1-01 將程式定義為可執行問題模型。

本篇進一步說明：程式的世界責任不能全部集中在一份來源碼中，而必須形成可識別模組。

SECTION

34.2 承接 PU-1-02

PU-1-02 建立實體、關係、規則與系統邊界。

本篇把世界邊界轉化為：

- 狀態擁有權；
- 決策權；
- 契約；
- 失敗責任；
- 非保證。

SECTION

34.3 承接 PU-1-03

PU-1-03 區分資料、狀態、事件與行動。

本篇決定：

- 哪個模組擁有狀態；
- 哪個模組接受行動；

- 哪個模組發布事件；
- 哪個資料只是跨邊界投影；
- 哪個證據足以完成契約。

SECTION

34.4 銜接 PU-1-05

下一篇將研究如何把責任、狀態、事件、契約、依賴與失敗投影成多種可視結構，讓：

- 使用者；
- 設計者；
- 工程師；
- 維護者；
- 稽核者；
- AI Agent；

能在不同尺度理解同一系統，而不把所有知識鎖在少數人腦中。

程式碼可以依任何方式切割。

可以按語言、檔案類型、功能、框架、部署或團隊分類。

但只有其中一部分切割能形成真正模組。

真正模組必須承擔世界：

- 擁有某些權威狀態；
- 維持某些不變量；
- 作出某些合法判定；
- 接受某些行動；
- 宣告某些事件；

- 承受某些失敗；
- 提供某些證據；
- 明示某些非責任。

因此，模組不是容器，而是承諾。

契約也不是資料格式，而是模組間對世界變化的可驗證承諾。

本文將系統架構收束為：

> 【System Architecture
> =
> Responsibility Modules
> +
> State Ownership
> +
> Explicit Contracts
> +
> Failure Ownership】

一個成熟架構不應只回答：

有哪些服務？
它們如何呼叫？
部署在哪裡？

還必須回答：

誰決定這項狀態？
誰維持這條不變量？
誰可以要求改變？
成功究竟代表什麼？
失敗由誰重試、補償與結案？
哪些資料只是投影？
契約改變時誰受影響？
哪一部分世界沒有任何模組負責？

如果這些問題沒有答案，即使系統擁有漂亮的資料夾、微服務、API Gateway 與事件匯流排，也仍可能只是被技術拓撲切碎的責任混合物。

本文的最終命題是：

> 【好的模組，使責任聚合；】

> 【好的契約，使邊界可穿越；】

> 【好的架構，使世界能改變，】

> 【卻不讓責任在改變中消失。】

```
module:
 module_id: "payment"
 purpose:
  - "maintain payment attempts, authorization, settlement claims, and refunds"
 owned_state:
  - "payment-attempt"
  - "payment-authorization"
  - "refund"
 invariants:
  - "same payment action cannot produce duplicate charge"
  - "refund cannot exceed captured amount"
 accepts:
  commands:
   - "AuthorizePayment"
   - "RefundPayment"
 emits:
  events:
   - "PaymentAuthorized"
   - "PaymentDeclined"
   - "PaymentOutcomeUnknown"
   - "PaymentRefunded"
 failure_ownership:
  local:
   - "provider timeout retry"
   - "duplicate callback deduplication"
  escalates:
   - "unknown settlement outcome"
 evidence:
  - "provider receipt"
  - "idempotency record"
 non_responsibilities:
  - "order fulfillment"
  - "physical delivery"
```

```
contract:
 contract_id: "authorize-payment-v2"
 provider: "payment-module"
 consumer: "order-module"
 intent: "request payment authorization for an order"
 input:
  order_id: "string"
  amount: "money"
  payment_method_ref: "token"
  idempotency_key: "string"
 preconditions:
  - "order is payment-pending"
  - "amount matches authoritative order amount"
 authorization:
  allowed_callers:
   - "order-module"
 effects:
  authoritative:
   - "create or reuse payment attempt"
  external:
   - "request authorization from provider"
 events:
  success:
   - "PaymentAuthorized"
  rejection:
   - "PaymentDeclined"
  uncertain:
   - "PaymentOutcomeUnknown"
 retry:
  safe: true
  identity: "idempotency_key"
 time:
  request_timeout: "10s"
  final_outcome_deadline: "24h"
 evidence:
  required:
   - "provider receipt"
 version:
  schema: "2.0"
  semantics: "2.1"
 governance:
  owner: "payments-team"
  deprecation_notice: "90d"
responsibility_edge:
 from: "order-module"
```

```
to: "payment-module"
interaction: "command"
purpose: "authorize payment"
contract: "authorize-payment-v2"
upstream_responsibility:
 - "customer purchase workflow"
 - "order completion tracking"
downstream_responsibility:
 - "payment legality"
 - "provider interaction"
 - "payment evidence"
failure:
 local_owner: "payment-module"
 workflow_owner: "order-module"
 user_resolution_owner: "support-process"
```

問題　必須回答

為何存在　問題世界目的

擁有什麼　權威狀態

保護什麼　不變量

接受什麼　Query／Command

宣告什麼　Event

如何證明　Evidence

如何失敗　Retry／Compensation／Escalation

依賴什麼　外部模組與服務

不保證什麼　Non-responsibility

如何演化　Version／Governance

- PU-1-01　程式不等於程式碼：可執行問題模型的基本定義
- PU-1-02　問題世界建模：實體、關係、規則與系統邊界
- PU-1-03　資料—狀態—事件—行動：程式系統的四元動力結構
- PU-1-04　責任、模組與契約：從檔案分類到系統邊界
- PU-1-05　可視化作為外部結構記憶：多角色、多尺度的程式理解

- PU-1-06　失敗也是程式：驗證、可觀測、恢復與長期維護

SECTION

Neo.K／EveMissLab 相關理論

- Neo.K with Aletheia，《程式不等於程式碼：可執行問題模型的基本定義》，2026。
- Neo.K with Aletheia，《問題世界建模：實體、關係、規則與系統邊界》，2026。
- Neo.K with Aletheia，《資料—狀態—事件—行動：程式系統的四元動力結構》，2026。
- Neo.K with Aletheia，《數位宇宙作為人工存在域：實體、狀態、規則與歷史》，2026。
- Neo.K with Aletheia，《Agent Runtime：能力規劃、工具調用與可恢復執行》，2026。
- Neo.K with Aletheia，《人類可見狀態：意圖程式系統的稽核、解釋與可逆治理》，2026。

SECTION

一般理論背景

- Parnas, D. L., “On the Criteria To Be Used in Decomposing Systems into Modules,” 1972.
- Meyer, B., Object-Oriented Software Construction, 1997.
- Evans, E., Domain-Driven Design, 2003.
- Fowler, M., Patterns of Enterprise Application Architecture, 2002.
- Hoare, C. A. R., “An Axiomatic Basis for Computer Programming,” 1969.
- Liskov, B. and Wing, J., “A Behavioral Notion of Subtyping,” 1994.
- Gamma, E. et al., Design Patterns, 1994.
- Hohpe, G. and Woolf, B., Enterprise Integration Patterns, 2003.
- Newman, S., Building Microservices, 2015.

- Kleppmann, M., Designing Data-Intensive Applications, 2017.
- Conway, M. E., “How Do Committees Invent?”, 1968.
- Team Topologies, Skelton, M. and Pais, M., 2019.

SECTION

v0.1 — 2026-07-27

- 完成十八篇地基論文第十篇。
- 區分模組、資料夾、套件、類別、程序、服務與部署單元。
- 建立模組九元模型與責任七元結構。
- 提出權威狀態單一責任原則。
- 以不變量、狀態生命週期與失敗責任決定模組凝聚力。
- 建立責任圖並區分呼叫圖、資料流與部署圖。
- 區分 Query、Command、Event、Evidence 與 Shared State。
- 提出共享寫入等於共同不變量責任。
- 建立十二元完整契約模型。
- 區分結構、語意、操作與治理四層契約。
- 分析同步、非同步、事件發布與人工工作流。
- 建立跨邊界失敗三層所有權。
- 提出委任不消除原始責任。
- 建立九步模組拆分法與十二項邊界測試。
- 加入訂單付款、帳號權限、網站部署、AI Agent、研究工作流與多團隊平台案例。
- 提出二十六類失敗模式與十四項可證偽研究方向。

- 完成與 PU-1-05《可視化作為外部結構記憶》的銜接。