可視化作為外部結構記憶：多角色、多尺度的程式理解

Visualization as External Structural Memory: Multi-Role and Multi-Scale Program Understanding

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

https://thisoneisneok.com/html/papers/pu-1-05.html

SECTION

Visualization as External Structural Memory: Multi-Role and Multi-Scale Program Understanding

論文編號： PU-1-05

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

前四篇已依序建立程式作為可執行問題模型、問題世界建模、資料—狀態—事件—行動四元動力，以及責任模組與跨邊界契約。然而，若這些結構只存在於原作者腦中、散落於來源碼、文件、會議紀錄與 Runtime，團隊仍無法形成共同系統理解。大型程式系統的核心瓶頸，往往不是缺少資料，而是缺少可被不同角色在不同尺度理解的外部結構記憶。

本文提出：

> 【Visualization
> ≠
> Decoration】

並將程式可視化定義為：

> 把問題世界、狀態、事件、行動、責任、契約、依賴、權限、時間、失敗與證據，投影為可被特定角色在特定任務與尺度下理解、核查與追蹤的外部結構記憶。

形式上，設完整系統本體為 Ω，則某一視圖為：

> 【V⊠
> =
> π⊠
> (
> Ω
> mid
> Role⊠,
> Task⊠,
> Scale⊠,
> Time⊠
>)】

其中：

- Ω：系統問題世界與執行結構；
- Role⊠：觀看者角色；
- Task⊠：當前理解或決策任務；
- Scale⊠：觀看尺度；
- Time⊠：視圖所對應時間或版本；
- V⊠：角色化、任務化、尺度化投影。

本文核心命題是：

> 【No Single View Can Represent the Whole System】

單一架構圖、ER 圖、流程圖、程式碼瀏覽器、服務拓撲或儀表板，都只能表示系統的一部分。成熟可視化不追求一張全知總圖，而應建立可互相連接、具有來源與版本的多投影視圖族：

> mathcal V
> =
> {
> V_W,
> V_E,
> V_S,
> V_A,
> V_R,
> V_C,
> V_D,
> V_T,
> V_F,
> V_H
> }

包括：

- 問題世界圖；
- 實體—關係圖；
- 狀態—事件—行動圖；
- 責任與模組圖；
- 契約圖；
- 資料與證據流圖；
- 依賴與部署圖；

- 時間與歷史圖；
- 失敗與恢復圖；
- 權限與治理圖。

本文提出「外部結構記憶」概念。傳統文件主要保存文字敘述；來源碼主要保存可執行表示；外部結構記憶則保存跨檔案、跨角色、跨模組與跨時間的關係。它使團隊不必每次都從大量局部資訊重新推導整體結構。

本文區分：

- 靜態視圖：某一版本下的世界、關係與責任結構；
- 動態視圖：狀態、事件、工作流與時間演化；
- Runtime 視圖：真實流量、呼叫、延遲、錯誤與資源；
- 歷史視圖：架構、規則、狀態與事故如何演化；
- 規範視圖：權限、責任、受影響主體與非保證。

本文也提出「視圖契約」：

> 【mathcal C_V
> =
> <
> Purpose,
> Audience,
> Source,
> Scope,
> Omission,
> Authority,
> Freshness,
> Trace,
> Update,
> Version
> >】

每個視圖必須說明：

- 為何存在；
- 給誰使用；
- 由哪些資料與模型生成；
- 表示哪些範圍；
- 刻意省略什麼；
- 是否為權威視圖；
- 新鮮度與延遲；
- 如何追蹤到來源碼、契約與 Runtime；
- 由誰更新；
- 對應哪個版本。

  若缺少視圖契約，圖表容易成為沒有責任的靜態裝飾。

  本文進一步建立「語意縮放」：觀看者可以從整體問題世界逐層展開到模組、契約、狀態機、事件、來源碼與 Runtime 證據，而不在縮放過程中失去語意身份。形式上：

> V^((0))
> xrightarrowzoom
> V^((1))
> xrightarrowzoom
> ...
> xrightarrowzoom
> V^((n))

且每層之間應保持可追蹤映射：

> Trace
> (
> x^((k)),
> x^((k+1))

本文主張，縮放不是把圖放大，而是轉換問題尺度。例如從「訂單系統」縮放到「付款責任」，再到「AuthorizePayment 契約」、事件、狀態變化、程式實作與執行證據。

本文亦處理多角色理解。使用者關心目的、狀態與可恢復操作；產品角色關心流程與受影響主體；領域專家關心規則與例外；工程師關心責任、契約與依賴；維運者關心 Runtime、失敗與恢復；安全與稽核角色關心權限、證據與規則版本；AI Agent 則需要機器可讀、可查詢、具有語意身份與來源的結構投影。

本文提出：

> 【Same System
> ≠
> Same View for Every Role】

但多角色視圖不能彼此斷裂。所有重要視圖都應連回共同語意本體或可追蹤中介表示，避免每個團隊各畫一套互不相容的世界。

本文進一步分析視圖漂移：當來源碼、規則、契約、部署或 Runtime 改變，但視圖未更新，視圖便成為錯誤外部記憶。成熟系統應盡可能由結構化來源生成視圖，並建立差異檢查、版本、更新責任與過期標記。

本文使用訂單付款、網站部署、AI Agent、醫療工作流、研究系統與多團隊平台等案例，展示多視圖如何降低認知負荷、責任模糊與知識集中。本文最後提出可證偽研究綱領，包括視圖漂移率、角色理解差異、語意縮放成功率、事故定位時間、交接時間、單點知識依賴、多投影一致性、AI Agent 導航效率與視圖生成自動化效果。

本文為第 1 冊最後一篇〈失敗也是程式〉建立理解與觀測地基：只有系統結構能被看見，失敗、驗證、恢復與長期維護才不會淪為事故發生後的臨時追查。

關鍵詞： 程式可視化、外部結構記憶、多角色視圖、語意縮放、多尺度理解、視圖契約、架構漂移、AI 原生可視化

SECTION

Abstract

This paper defines visualization as external structural memory rather than decoration.

Given a system ontology Ω, a view is a role-, task-, scale-, and time-specific projection:

> V⊠
> =
> π⊠

No single diagram can represent the whole system. Mature systems require a connected family of views covering the problem world, entities, state transitions, responsibilities, contracts, data, dependencies, time, failures, permissions, and runtime evidence.

The paper introduces view contracts, semantic zoom, role-specific projections, traceability, view drift, machine-readable visualization, and AI-native navigation.

Keywords: software visualization, external structural memory, semantic zoom, multi-role views, architecture drift, traceability

當專案很小，作者可以在腦中同時維持：

- 問題世界；
- 檔案位置；
- 狀態流；
- 模組責任；
- 外部依賴；
- 失敗路徑。

但專案成長後，這些結構會散布於：

- 原始碼；
- 資料庫；
- API；
- 事件；
- 部署設定；
- 文件；
- Issue；
- 事故紀錄；
- 團隊口述。

此時團隊缺少的不是更多文字，而是一種能把關係重新外部化的結構記憶。

如果整體結構只存在於少數資深工程師腦中，系統便形成認知單點故障。

SECTION

2.1 裝飾性圖表

裝飾性圖表通常：

- 美觀；
- 適合簡報；
- 缺少來源；
- 不標示版本；
- 無法追蹤；
- 很快過時。

SECTION

2.2 結構性視圖

結構性視圖必須能支援：

- 理解；
- 決策；
- 驗證；
- 除錯；
- 交接；
- 稽核；
- 變更影響分析。

SECTION

2.3 圖片不等於可視化系統

PNG 或白板截圖只是靜態輸出。

成熟可視化更接近：

- 可查詢圖；
- 可縮放模型；
- 可連結節點；
- 可切換角色；
- 可比較版本；
- 可追蹤來源；
- 可連接 Runtime。

本文定義：

> 外部結構記憶是由團隊共同維持、可被多角色查詢，並保存系統跨對象、跨模組、跨時間關係的外部化認知結構。

它不同於：

- 原始碼：主要保存可執行表示；
- 文件：主要保存敘述；
- 日誌：主要保存執行紀錄；
- 資料庫：主要保存世界或投影資料。

外部結構記憶主要保存：

- 誰與誰相關；
- 誰擁有什麼；

- 哪個事件改變什麼；
- 哪個契約連接哪些模組；
- 哪些失敗沿何路徑傳播；
- 某一視圖從何而來；
- 某一結構在哪個版本成立。

設系統本體為：

> Ω
> =
> <
> W,
> E,
> S,
> A,
> V,
> R,
> M,
> C,
> D,
> F,
> H,
> P,
> T
> >

其中包含問題世界、實體、狀態、行動、事件、規則、模組、契約、資料、失敗、歷史、權限與時間。

某一視圖定義為：

> 【V⊠
> =
> π⊠
> (
> Ω

SECTION

4.1 Role

誰正在觀看：

- 使用者；
- 產品設計者；
- 領域專家；
- 工程師；
- 維運者；
- 安全人員；
- 稽核者；
- AI Agent。

SECTION

4.2 Task

觀看者要完成什麼：

- 理解流程；
- 修改功能；
- 定位事故；
- 審核權限；
- 判斷影響；
- 驗證契約；
- 規劃遷移。

SECTION

4.3 Scale

從哪個尺度觀看：

- 整體問題世界；
- 領域；
- 模組；
- 工作流；
- 實體；
- 狀態；
- 函式；
- Runtime 實例。

SECTION

4.4 Time

視圖對應：

- 當前版本；
- 過去版本；
- 計畫版本；
- 事故時間；
- Runtime 時間窗；
- 規則生效區間。

SECTION

4.5 Projection

投影必然包含選擇與省略，因此視圖不是系統本身。

SECTION

5.1 資訊過量

若把所有實體、函式、資料表、服務、事件與呼叫放在同一張圖中，結果通常不可理解。

SECTION

5.2 關係種類不同

下列關係不能用同一箭頭無差別表示：

- 呼叫；
- 狀態擁有；
- 資料讀取；
- 事件訂閱；
- 權限授予；
- 失敗傳播；
- 物理部署；
- 歷史依賴。

SECTION

5.3 角色目的不同

維運者需要知道事故與 Runtime。

領域專家需要知道規則與例外。

使用者需要知道可做什麼與如何恢復。

SECTION

5.4 尺度衝突

高階視圖需要壓縮細節。

低階視圖需要保存具體結構。

SECTION

5.5 核心命題

> 【One System
> →
> Many Valid Views】

而不是：

> One Perfect Diagram
> =
> The System

本文提出：

> mathcal V
> =
> {
> V_W,
> V_E,
> V_S,
> V_R,
> V_C,

SECTION

6.1 問題世界圖 V_W

表示：

- 目的；
- 主體；
- 核心實體；
- 外部世界；
- 系統邊界；
- 非保證。

SECTION

6.2 實體—關係圖 V_E

表示：

- 身份；
- 屬性；
- 組成；
- 擁有；
- 委任；
- 規範關係。

SECTION

6.3 狀態—事件—行動圖 V_S

表示：

- 合法狀態；
- 行動；
- 事件；
- 前置條件；
- 轉移；
- 爭議與恢復。

SECTION

6.4 責任模組圖 V_R

表示：

- 狀態擁有者；
- 不變量；
- 決策權；
- 失敗所有者；
- 非責任。

SECTION

6.5 契約圖 V_C

表示：

- Query；
- Command；
- Event；

- Evidence；
- Shared State；
- 版本與時間承諾。

SECTION

6.6 資料與證據流圖 V_D

表示：

- 權威資料；
- 投影；
- 快取；
- 來源；
- 轉換；
- 證據；
- 敏感資料。

SECTION

6.7 依賴與部署圖 V_T

表示：

- 程序；
- 服務；

  -節點；
- 網路；
- Runtime；

- 資源；
- 外部服務。

SECTION

6.8 失敗與恢復圖 V_F

表示：

- 失敗來源；
- 傳播；
- 重試；
- 降級；
- 補償；
- 人工接管；
- 回復路徑。

SECTION

6.9 權限與治理圖 V_P

表示：

- 誰能讀；
- 誰能寫；
- 誰能批准；
- 誰能改規則；
- 誰能看歷史；
- 誰能申訴。

SECTION

6.10 歷史與演化圖 V_H

表示：

- 版本；
- 決策；
- 架構變更；
- 契約棄用；
- 事故；
- 遷移；
- 技術債。

本文定義：

> 【mathcal C_V
> =
> <
> Purpose,
> Audience,
> Source,
> Scope,
> Omission,
> Authority,
> Freshness,
> Trace,
> Update,
> Version
> >】

SECTION

7.1 Purpose

視圖支援哪一類理解或決策。

SECTION

7.2 Audience

哪些角色可正確解讀此視圖。

SECTION

7.3 Source

視圖由哪些權威模型、程式、事件、Runtime 或人工決策生成。

SECTION

7.4 Scope

表示哪些領域、模組、時間與環境。

SECTION

7.5 Omission

刻意不表示哪些內容。

SECTION

7.6 Authority

視圖是：

- 權威；

- 衍生；
- 草稿；
- 規劃；
- 說明；
- Runtime 觀測。

SECTION

7.7 Freshness

多久更新，允許多大延遲。

SECTION

7.8 Trace

如何連回：

- 問題世界；
- 契約；
- 原始碼；
- 部署；
- 執行證據。

SECTION

7.9 Update

誰或哪個流程負責更新。

SECTION

7.10 Version

視圖對應的模型與系統版本。

SECTION

8.1 圖可能過時

圖上顯示模組 A 擁有狀態，但實際程式可能已移轉至模組 B。

SECTION

8.2 Runtime 視圖也有限

監控拓撲顯示真實呼叫，但不自動知道領域責任與規範意義。

SECTION

8.3 規格視圖也有限

規格描述應然結構，Runtime 可能偏離。

SECTION

8.4 雙重核查

成熟系統應同時比較：

> V_(declared)
>
> and
>
> V_(observed)

並標記差異：

> ΔV
> =
> V_(observed)
> -
> V_(declared)

SECTION

8.5 視圖差異本身是重要資料

宣稱架構與實際執行不一致，可能代表：

- 漂移；
- 隱藏依賴；
- 未文件化變更；
- 異常；
- 侵權或越權。

SECTION

9.1 放大不等於縮放

把同一張圖放大，只是增加視覺尺寸。

語意縮放是改變問題層級。

SECTION

9.2 縮放鏈

例如：

電商世界

→ 訂單責任
→ 付款工作流
→ AuthorizePayment 契約
→ PaymentAuthorized 事件
→ 訂單狀態轉移
→ 實作函式
→ Runtime 證據
形式上：

$$
V^{(0)} \xrightarrow{\zeta} V^{(1)} \xrightarrow{\zeta} \cdots \xrightarrow{\zeta} V^{(n)}
$$

SECTION

9.3 身份保持

同一概念在不同尺度必須保留可追蹤身份：

$$
\mathrm{Trace}(x^{(k)}, x^{(k+1)})
$$

SECTION

9.4 摘要與展開

高階節點應顯示：

- 責任；

- 重要狀態；
- 對外契約；
- 主要失敗；
- 風險。

展開後再顯示內部細節。

SECTION

9.5 語意跳躍失敗

若從高階「付款模組」跳到低階檔案列表，卻無法知道哪段程式維持哪條不變量，就不是有效縮放。

SECTION

10.1 使用者視圖

關心：

- 現在發生什麼；
- 可以做什麼；
- 哪些狀態等待外部；
- 如何撤回、申訴或恢復。

SECTION

10.2 產品視圖

關心：

- 目的；
- 使用流程；

- 角色；
- 受影響者；
- 完成與失敗；
- 指標與非目標。

SECTION

10.3 領域專家視圖

關心：

- 實體；
- 規則；
- 例外；
- 不變量；
- 歷史承諾；
- 人工判斷。

SECTION

10.4 工程師視圖

關心：

- 模組；
- 契約；
- 狀態擁有；
- 依賴；
- 資料流；

- 版本；
- 測試。

SECTION

10.5 維運者視圖

關心：

- Runtime；
- 流量；
- 延遲；
- 錯誤；
- 資源；
- 部署；
- 回復。

SECTION

10.6 安全與稽核視圖

關心：

- 權限；
- 敏感資料；
- 規則版本；
- 操作證據；
- 越權；
- 例外；

- 責任鏈。

SECTION

10.7 AI Agent 視圖

需要：

- 機器可讀語意；
- 節點身份；
- 關係型別；
- 查詢接口；
- 來源；
- 版本；
- 權限；
- 允許行動與禁止行動。

SECTION

10.8 共同本體

角色視圖雖不同，仍應連回共同語意：

> V⊠
> xrightarrowtrace
> Ω
> xleftarrowtrace
> V⊠

SECTION

11.1 人類工作記憶有限

大型系統可能包含數千個檔案、服務與事件，無法同時被單一人腦維持。

SECTION

11.2 視圖的目的不是顯示最多

好的視圖應顯示完成任務所需的最少充分結構。

SECTION

11.3 結構壓縮

可定義某視圖的壓縮：

> Compression(V)
> =
> (|Ω|)/(|V|)

但壓縮率高不代表品質高。

若關鍵關係被省略，視圖會誤導。

SECTION

11.4 任務充分性

> Sufficiency(V,T)
> =
> viewV
> contains enough structure for taskT

SECTION

11.5 分層揭露

先顯示：

- 核心責任；
- 主要狀態；
- 關鍵契約；
- 高風險失敗；

再按需展開細節。

SECTION

12.1 靜態視圖

表示某一版本的：

- 實體；
- 模組；
- 契約；
- 權限；
- 部署計畫。

SECTION

12.2 動態視圖

表示：

- 行動；
- 事件；
- 狀態轉移；
- 工作流；

- 競爭；
- 補償。

SECTION

12.3 Runtime 視圖

表示真實運行中的：

- 呼叫；
- 延遲；
- 錯誤；
- 佇列；
- 資源；
- 流量；
- 版本。

SECTION

12.4 歷史視圖

表示：

- 架構何時改變；
- 規則何時生效；
- 契約如何遷移；
- 事故如何形成；
- 責任如何轉移。

SECTION

12.5 四者需要對照

規格正確但 Runtime 偏離，或 Runtime 正常但規範責任錯誤，都可能造成系統失敗。

SECTION

13.1 時間軸

把事件依：

- 發生時間；
- 接收時間；
- 處理時間；
- 生效時間；

分開呈現。

SECTION

13.2 長時工作流

需要顯示：

- 當前階段；
- 等待對象；
- 超時；
- 重試；
- 補償；
- 人工節點；

- 完成證據。

SECTION

13.3 版本時間

架構圖應能回答：

這個邊界在哪個版本成立？
事故發生時使用哪個契約？
規則改變前後差在哪裡？

SECTION

13.4 視圖回放

歷史回放可協助：

- 事故分析；
- 責任判定；
- 狀態重建；
- 變更理解。

SECTION

14.1 成功流程不足

只畫正常流程會隱藏真正維護成本。

SECTION

14.2 失敗節點

每個重大步驟應顯示：

- 可能失敗；

- 是否可重試；
- 是否冪等；
- 是否部分成功；
- 是否需要補償；
- 誰負責；
- 需要什麼證據。

SECTION

14.3 失敗傳播圖

> F□
> →
> F□
> →
> F□

顯示局部故障如何影響流程、使用者與外部世界。

SECTION

14.4 回復圖

顯示：

- 重啟；
- 重試；
- 降級；
- 切換；
- 回滾；

- 補償；
- 人工接管。

SECTION

14.5 失敗作為一等視圖

失敗不是架構圖旁的註解，而應與成功流程具有同等結構地位。

SECTION

15.1 權限不能只看角色名稱

需要看到：

- 主體；
- 情境；
- 資源；
- 行動；
- 時間；
- 委任；
- 批准；
- 撤銷。

SECTION

15.2 世界修改能力

視圖應標示誰能：

- 建立身份；

- 修改狀態；
- 發布事件；
- 改規則；
- 刪歷史；
- 修改視圖本身。

SECTION

15.3 超級權限

高權限節點應顯示：

- 使用條件；
- 稽核；
- 雙重批准；
- 緊急程序；
- 風險。

SECTION

15.4 規則治理

視圖也需標示：

- 規則所有者；
- 版本；
- 生效時間；
- 受影響者；
- 異議與申訴路徑。

SECTION

16.1 定義

視圖漂移是指：

> V⊠
> ≠
> π
> (
> Ω⊠
>)

也就是視圖已不再正確表示當前系統。

SECTION

16.2 漂移來源

- 原始碼改變；
- 契約改變；
- 狀態擁有者轉移；
- 服務拆分；
- 權限修改；
- Runtime 新增隱藏依賴；
- 文件未更新；
- 人工流程改變。

SECTION

16.3 漂移風險

過時視圖可能導致：

- 修改錯誤模組；
- 依賴不存在的契約；
- 遺漏真正故障路徑；
- 誤判權限；
- 錯誤事故回復；
- 交接失敗。

SECTION

16.4 過期標記

若無法確認視圖新鮮度，應明示：

stale
unverified
planning-only
runtime-diverged

SECTION

16.5 漂移檢查

可比較：

- 宣告契約與實際呼叫；
- 模組責任與資料寫入；
- 部署圖與 Runtime 拓撲；
- 權限規格與稽核紀錄；
- 狀態圖與真實事件序列。

SECTION

17.1 自動生成視圖

可由：

- 程式碼；
- schema；
- API 規格；
- 事件定義；
- Runtime tracing；
- 部署設定；
- 權限政策；

自動生成。

SECTION

17.2 自動生成的限制

工具能看見技術結構，但未必知道：

- 世界目的；
- 語意責任；
- 非目標；
- 受影響主體；
- 人工流程；
- 規範邊界。

SECTION

17.3 手工視圖

人類可表達：

- 設計理由；
- 領域規則；
- 例外；
- 預期架構；
- 價值與治理。

  但手工視圖較容易漂移。

SECTION

17.4 混合策略

理想方式是：

> 【V
> =
> V_(generated)
> +
> V_(curated)
> +
> Δ V_(checked)】

即由機器生成可驗證基礎，再由人類補充語意，最後持續比較差異。

SECTION

18.1 從目的到執行

應能追蹤：

目的
→ 問題世界
→ 責任模組
→ 契約
→ 狀態／事件
→ 原始碼
→ 部署
→ Runtime 證據

SECTION

18.2 從事故回到目的

也應能反向追蹤：

錯誤指標
→ Runtime 實例
→ 契約
→ 模組責任
→ 世界狀態
→ 使用者影響
→ 原始目的

SECTION

18.3 追蹤身份

每個重要節點需要穩定識別：

```
trace_id:
  world_concept: "payment.authorization"
  module: "payment"
  contract: "authorize-payment-v2"
  event: "PaymentAuthorized"
  code_symbol: "PaymentAuthorizer.authorize"
  runtime_service: "payments-prod"
```

SECTION

18.4 追蹤不等於字串搜尋

名稱可能改變，真正追蹤需要：

- 身份；
- 版本；
- 關係；
- 來源；
- 遷移歷史。

SECTION

19.1 AI 不能只讀長文件

長文件可提供內容，但 AI 若要可靠操作系統，需要可查詢的結構。

SECTION

19.2 機器可讀視圖

節點與邊應具有：

- 類型；
- 身份；
- 語意；
- 來源；
- 版本；
- 權限；
- 新鮮度；
- 可用操作。

SECTION

19.3 Agent 導航

Agent 可依任務查詢：

哪個模組擁有訂單付款狀態？
部署失敗後由誰補償？
這個 API 成功代表哪一層完成？
修改此規則會影響哪些角色？

SECTION

19.4 視圖作為行動約束

結構記憶不只協助回答，也可限制 Agent：

- 不得直接寫入非擁有狀態；
- 不得跳過契約；
- 不得執行未授權操作；
- 高風險節點需要人工批准。

SECTION

19.5 AI 生成視圖的風險

AI 可能：

- 錯誤推斷關係；
- 把命名相似視為同一責任；
- 混淆宣告與 Runtime；
- 生成看似完整但缺少來源的圖。

所以 AI 產生的關係需標示：

declared
observed
inferred
unverified

SECTION

20.1 視圖所有者

每個關鍵視圖應有維護責任。

SECTION

20.2 修改權

需要知道誰能：

- 新增節點；
- 修改關係；
- 改變權威標記；
- 刪除歷史；
- 批准新版本。

SECTION

20.3 視圖審查

重大架構、規則與權限變更應觸發視圖審查。

SECTION

20.4 視圖保留

歷史視圖可用於：

- 事故回放；
- 遷移；
- 法規稽核；

- 設計決策；
- 技術債追蹤。

SECTION

20.5 視圖退出

過時視圖應被：

- 棄用；
- 封存；
- 加警告；
- 連至替代版本；

而不是繼續與新視圖並列。

SECTION

21.1 問題世界圖

顯示：

- 客戶；
- 商家；
- 訂單；
- 付款；
- 履約；
- 爭議；
- 外部支付與物流。

SECTION

21.2 責任圖

顯示：

- 訂單模組擁有訂單生命週期；
- 付款模組擁有付款嘗試與授權；
- 物流模組擁有配送主張；
- 流程協調責任仍由訂單模組承擔。

SECTION

21.3 狀態圖

區分：

payment-pending
authorized
paid
settlement-pending
refunded
outcome-unknown

SECTION

21.4 Runtime 視圖

顯示付款回呼重複、延遲、失敗率與佇列。

SECTION

21.5 使用者視圖

只需顯示：

- 目前付款狀態；

- 是否需要操作；
- 是否可重試；
- 何時聯絡支援。

同一系統需要不同視圖，但狀態語意必須一致。

SECTION

22.1 計畫視圖

顯示：

Release
→ Approval
→ Deployment
→ Verification
→ Traffic Shift

SECTION

22.2 依賴視圖

顯示：

- 建置系統；
- Artifact 儲存；
- 部署平台；
- DNS；
- 監控；
- 外部服務。

SECTION

22.3 Runtime 視圖

顯示：

- 實際版本；
- 流量比例；
- 錯誤；
- 延遲；
- 資源；
- 健康檢查。

SECTION

22.4 漂移檢查

若宣告版本與實際執行版本不同，立即標記。

SECTION

22.5 失敗視圖

顯示：

- 哪一步失敗；
- 是否部分完成；
- 能否回滾；
- 外部 DNS 是否已改變；
- 誰負責驗收。

SECTION

23.1 目標視圖

顯示 Agent 的：

- 委任；
- 目標；
- 非目標；
- 停止條件；
- 人類保留選擇。

SECTION

23.2 能力與權限視圖

區分：

- 可使用工具；
- 已授權工具；
- 需要批准的工具；
- 禁止操作。

SECTION

23.3 計畫與執行視圖

顯示：

Goal
→ Plan
→ Action
→ Tool
→ Evidence
→ State Update

SECTION

23.4 記憶視圖

區分：

- 當前工作記憶；
- 長期記憶；
- 外部文件；
- 推論；
- 未驗證主張。

SECTION

23.5 事故視圖

若 Agent 重複寄信，可追蹤：

- 原始行動身份；
- 工具回覆；
- 重試原因；
- 冪等契約；
- 誰批准；
- 外部影響。

SECTION

24.1 角色視圖

病人、醫師、護理師、行政人員看到不同內容。

SECTION

24.2 規則視圖

顯示：

- 哪些狀態需要專業批准；
- 哪些資料只是模型建議；
- 哪些操作需病人同意；
- 哪些例外需要人工覆核。

SECTION

24.3 時間視圖

顯示檢驗發生時間、結果回報時間、醫師閱讀時間與處置生效時間。

SECTION

24.4 權限視圖

敏感資料與高風險操作必須具有明示存取與稽核鏈。

SECTION

24.5 不確定性視圖

應顯示：

- 資料不足；
- 模型適用域外；
- 專業意見不一致；
- 等待病人選擇。

SECTION

25.1 問題圖

顯示研究問題、定義、命題與前置依賴。

SECTION

25.2 證據圖

連接：

- 文獻；
- 計算；
- 實驗；
- 反例；
- 證書；
- 失敗紀錄。

SECTION

25.3 狀態視圖

區分：

proposed
open
partially-supported
contradicted
verified
published

SECTION

25.4 歷史視圖

保留命題如何改變、失敗路徑與版本分支。

SECTION

25.5 AI 研究 Agent

Agent 應能沿證據圖查詢，而不是只從最終論文文字猜測研究狀態。

SECTION

26.1 高階世界圖

顯示身份、內容、搜尋、推薦、計費、通知與稽核等責任域。

SECTION

26.2 團隊視圖

顯示團隊擁有模組、契約與值班責任。

SECTION

26.3 依賴圖

顯示同步、非同步、共享資料與外部服務。

SECTION

26.4 事故視圖

將技術錯誤映射到：

- 哪個責任；
- 哪些使用者；
- 哪個契約；
- 哪個團隊；

- 哪個恢復路徑。

SECTION

26.5 管理視圖

不只顯示服務數量，也顯示：

- 責任孤島；
- 契約爆炸；
- 狀態無主；
- 高權限集中；
- 知識單點風險。
- 圖即系統： 把單一圖表視為完整本體。
- 可視化即裝飾： 圖只服務簡報，不服務理解與驗證。
- 一張總圖： 所有節點與關係塞進同一畫面。
- 箭頭同義化： 呼叫、事件、權限、依賴與失敗使用同一箭頭。
- 無角色視圖： 所有人被迫看同一技術投影。
- 縮放只是放大： 沒有問題尺度轉換與身份追蹤。
- 無來源： 圖中節點無法連回規格、程式或 Runtime。
- 無省略聲明： 觀看者不知道視圖沒畫什麼。
- 規格即現況： 宣告架構被當成 Runtime 真實。
- Runtime 即責任： 真實呼叫拓撲被誤認為領域邊界。
- 視圖漂移： 系統改變但圖未更新。
- 過時不標記： 舊圖與新圖並列，無版本與新鮮度。

- 自動生成全知迷思： 工具拓撲被當成完整世界語意。
- 手工圖永久化： 白板決策沒有進入結構化來源。
- 投影互相矛盾： 狀態圖、契約圖與 UI 使用不同語意。
- 只畫成功： 失敗、補償與人工接管缺席。
- 權限不可見： 世界修改能力未進入視圖。
- 時間被壓平： 當前、歷史、規劃與事故版本混合。
- AI 推論無標記： 推測關係被呈現為已驗證事實。
- 知識單點： 圖仍需資深作者口頭解釋才能理解。
- 視圖無治理： 沒有所有者、審查、棄用與遷移。
- 視覺複雜度崇拜： 圖越複雜被誤認為越專業。

SECTION

28.1 視圖漂移率

比較視圖宣告與來源碼、契約、部署及 Runtime 的差異：

> R_(drift)
> =
> (|incorrect or stale view relations|)/(|sampled view relations|)

SECTION

28.2 角色理解差異

給不同角色同一張圖與角色化視圖，測量其對：

- 目的；
- 狀態；

- 責任；
- 失敗；
- 權限；

的理解準確率。

SECTION

28.3 語意縮放成功率

測量觀看者能否從高階概念正確追蹤至低階契約、程式與 Runtime，並反向回到世界目的。

SECTION

28.4 事故定位時間

比較只有日誌與來源碼，以及具有責任、失敗、Runtime 多視圖的團隊，定位事故所需時間。

SECTION

28.5 交接時間

測量新成員在多視圖外部結構記憶輔助下，達到可安全修改系統所需時間。

SECTION

28.6 知識單點依賴

統計必須詢問特定資深成員才能理解的關鍵結構比例。

SECTION

28.7 多投影一致性

比較：

- 問題世界圖；
- 狀態圖；
- 契約；
- 原始碼；
- UI；
- Runtime；

  之間的語意差異。

SECTION

28.8 視圖任務充分性

對特定任務 T，測量視圖是否包含足夠且不過量的資訊：

> Q_V
> =
> Sufficiency(V,T)
> -
> Overload(V,T)

SECTION

28.9 過期視圖誤導率

統計事故、錯誤修改與溝通失敗中，有多少受到過時視圖影響。

SECTION

28.10 自動生成效益

比較手工、全自動及混合視圖在：

- 新鮮度；
- 語意完整；
- 維護成本；
- 漂移；

  上的差異。

SECTION

28.11 AI Agent 導航效率

比較 Agent 僅使用來源碼搜尋，以及使用機器可讀責任、契約與狀態圖時，完成系統理解與修改任務的準確率。

SECTION

28.12 推論關係錯誤率

測量 AI 自動推導之責任、依賴與語意關係中，經人工驗證後的錯誤比例。

SECTION

28.13 失敗視圖覆蓋率

檢查重大工作流是否正式表示重試、補償、超時、部分成功與人工接管。

SECTION

28.14 權限可見率

測量系統中的高風險世界修改能力，有多少能在權限與治理視圖中被識別。

- 可視化不是裝飾，而是系統外部結構記憶。
- 任何視圖都是角色、任務、尺度與時間條件下的部分投影。

- 不存在能完整代表大型程式系統的單一全知視圖。
- 同一系統需要問題世界、實體、狀態、責任、契約、資料、部署、失敗、權限與歷史等多種視圖。
- 每個視圖都必須明示目的、受眾、來源、範圍、省略、權威、新鮮度、追蹤、更新與版本。
- 圖表本身不自動具有權威地位。
- 宣告視圖與 Runtime 觀測視圖必須分離並持續比較。
- 語意縮放不是視覺放大，而是問題尺度轉換。
- 縮放過程必須保持概念身份與追蹤關係。
- 不同角色需要不同視圖，但所有視圖應連回共同語意本體。
- 好視圖追求任務所需的最少充分結構，而非最大資訊量。
- 靜態、動態、Runtime 與歷史視圖不能互相取代。
- 失敗、補償與恢復應與成功流程具有同等可視地位。
- 權限圖必須呈現世界修改能力，而不只是角色名稱。
- 視圖漂移會把外部記憶轉化為錯誤記憶。
- 自動生成視圖擅長技術事實，但不能自動取得完整領域語意。
- 手工視圖擅長設計意圖，但需要版本與漂移治理。
- 成熟視圖應由機器生成基礎、人類補充語意並持續差異核查。
- 雙向可追蹤性應連接目的、世界、責任、契約、程式、部署與證據。
- AI Agent 需要機器可讀、具來源、版本與權限的結構視圖。
- AI 推導的視圖關係必須區分宣告、觀測、推論與未驗證。

22.

> 【可理解的系統，】

> 【不是把所有細節同時展示，】

> 【而是讓每個角色都能在適當尺度看見】

> 【自己需要理解的世界，並能追蹤回共同結構。】

SECTION

30.1 承接 PU-1-01

PU-1-01 將程式定義為可執行問題模型。

本篇指出，可執行問題模型不能只存在於來源碼，而應具有多重可理解投影。

SECTION

30.2 承接 PU-1-02

PU-1-02 建立實體、關係、規則、邊界、未知與歷史。

本篇把這些結構外部化為問題世界圖、關係圖、規則圖與時間圖。

SECTION

30.3 承接 PU-1-03

PU-1-03 建立資料、狀態、事件與行動四元動力。

本篇將動力投影為狀態圖、事件時間線、工作流與 Runtime 回放。

SECTION

30.4 承接 PU-1-04

PU-1-04 建立責任模組與契約。

本篇使狀態擁有、不變量、契約、失敗所有權與非責任可被多角色共同看見。

SECTION

30.5 銜接 PU-1-06

下一篇將完成第 1 冊，集中處理：

- 驗證；
- 可觀測性；
- 錯誤與失敗；
- 重試與補償；
- 回復；
- 事故；
- 長期維護；
- 系統演化。

  而本篇的外部結構記憶，將成為定位失敗、理解影響與保存維護知識的基礎。

  程式碼是精確且必要的投影。

  但來源碼通常依檔案、語言與實作順序組織，並不自然呈現：

- 問題世界全貌；
- 狀態擁有權；
- 跨模組契約；
- 失敗傳播；
- 權限；
- 歷史；
- 使用者與受影響者。

因此，只靠閱讀程式碼理解大型系統，等於要求每個觀看者從局部表示重新推導整體世界。

這種推導成本會隨系統規模、時間與團隊變動持續上升，最終使少數長期維護者成為唯一架構記憶。

本文提出另一種結構：

> 【Program Understanding
> =
> Executable Representations
> +
> Connected External Views
> +
> Runtime Evidence
> +
> Historical Trace】

外部結構記憶不是要取代程式碼，也不是為所有人建立同一張巨大圖。

它的目的，是讓不同角色能夠回答不同但相連的問題：

這個系統希望改變什麼？
世界裡有哪些對象？
誰擁有哪個狀態？
哪個事件使它改變？
哪個契約跨越邊界？
哪些資料只是投影？
哪些權限能修改世界？
失敗會傳到哪裡？
目前 Runtime 是否符合宣告架構？
這個結構在哪個版本成立？

真正成熟的可視化也不應停留在圖片。

它應該是：

- 可查詢；
- 可縮放；
- 可比較；
- 可追蹤；

- 可版本化；
- 可由 Runtime 校驗；
- 可供人類與 AI 共同使用。

本文將其收束為：

> 【External Structural Memory
> =
> Multi-View Projection
> +
> Semantic Identity
> +
> Bidirectional Traceability
> +
> View Governance】

最終命題是：

> 【系統不應只被機器執行，】

> 【也必須能被不同主體共同看見。】

只有當結構能被共同看見、核查、反駁與更新，程式才不再是少數天才腦中的隱性世界，而能成為可教學、可協作、可維護與可治理的公共工程結構。

```
view_contract:
 view_id: "payment-responsibility-view"
 purpose:
  - "show state ownership and cross-module contracts"
 audience:
  - "engineer"
  - "incident-commander"
  - "AI-agent"
 source:
  declared:
   - "module-registry"
   - "contract-registry"
  observed:
```

```
    - "runtime-traces"
  scope:
    domain: "payment"
    environment: "production"
    version: "2026.07"
  omissions:
    - "internal function call graph"
    - "non-critical metrics"
  authority:
    responsibility: "declared-authoritative"
    runtime_edges: "observed"
  freshness:
    declared: "on-merge"
    runtime: "5m"
  trace:
    enabled: true
    targets:
      - "contract"
      - "code-symbol"
      - "deployment"
      - "incident"
  owner: "payments-team"
view_node:
  node_id: "payment.authorization"
  node_type: "world-state"
  label: "Payment Authorization"
  ownership:
    module: "payment"
  relations:
    - type: "changed-by"
      target: "AuthorizePayment"
    - type: "produces"
      target: "PaymentAuthorized"
    - type: "projected-as"
      target: "order-payment-status"
  provenance:
    source: "payment-domain-model-v2"
    status: "declared"
    verified_at: "2026-07-27"
  permissions:
    writable_by:
      - "payment-module"
semantic_zoom:
  root: "commerce-world"
  levels:
```

```
    - level: 0
      node: "commerce-world"
      view: "problem-world"
    - level: 1
      node: "payment-responsibility"
      view: "responsibility"
    - level: 2
      node: "authorize-payment-v2"
      view: "contract"
    - level: 3
      node: "PaymentAuthorized"
      view: "event-state"
    - level: 4
      node: "PaymentAuthorizer.authorize"
      view: "code"
    - level: 5
      node: "payments-prod-trace"
      view: "runtime"
view_drift:
  view_id: "deployment-topology"
  detected_at: "2026-07-27T15:00:00+08:00"
  declared:
    edge: "gateway -> deployment-service"
  observed:
    edge: "gateway -> legacy-deployer"
  classification: "runtime-diverged"
  severity: "high"
  impact:
    - "incident runbook may route to wrong owner"
  owner: "platform-team"
  resolution:
    status: "open"
```

- 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，《責任、模組與契約：從檔案分類到系統邊界》，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。
- Neo.K with Aletheia，《可編譯世界：從程式執行到世界狀態演化》，2026。

SECTION

一般理論背景

- Tufte, E. R., The Visual Display of Quantitative Information, 1983.
- Shneiderman, B., “The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations,” 1996.
- Card, S. K., Mackinlay, J. D., and Shneiderman, B., Readings in Information Visualization, 1999.
- Ware, C., Information Visualization: Perception for Design, 2000.
- Larkin, J. H. and Simon, H. A., “Why a Diagram Is Sometimes Worth Ten Thousand Words,” 1987.
- Hutchins, E., Cognition in the Wild, 1995.
- Norman, D. A., Things That Make Us Smart, 1993.
- Clements, P. et al., Documenting Software Architectures, 2002.
- Rozanski, N. and Woods, E., Software Systems Architecture, 2005.
- Harel, D., “Statecharts: A Visual Formalism for Complex Systems,” 1987.

- Bass, L., Clements, P., and Kazman, R., Software Architecture in Practice, 2012.
- Storey, M.-A., “Theories, Methods and Tools in Program Comprehension,” 2005.

SECTION

v0.1 — 2026-07-27

- 完成十八篇地基論文第十一篇。
- 將程式可視化定義為外部結構記憶。
- 建立角色、任務、尺度與時間條件下的視圖投影模型。
- 提出不存在單一全知系統視圖。
- 建立問題世界、關係、狀態、責任、契約、資料、部署、失敗、權限與歷史視圖族。
- 建立十元視圖契約。
- 提出語意縮放與跨尺度身份保持。
- 區分使用者、產品、領域、工程、維運、安全、稽核與 AI Agent 視圖。
- 分析認知負荷、最少充分結構與分層揭露。
- 區分靜態、動態、Runtime 與歷史視圖。
- 建立失敗視圖、權限治理視圖與時間回放。
- 提出視圖漂移、過期標記與宣告—觀測差異。
- 建立自動生成、人類補充與差異核查混合策略。
- 建立雙向可追蹤與 AI 原生機器可讀視圖。
- 加入訂單付款、網站部署、AI Agent、醫療、研究系統與多團隊平台案例。
- 提出二十二類失敗模式與十四項可證偽研究方向。
- 完成與 PU-1-06《失敗也是程式》的銜接。