問題世界建模：實體、關係、規則與系統邊界

Problem-World Modeling: Entities, Relations, Rules, and System Boundaries

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

https://thisoneisneok.com/html/papers/pu-1-02.html

SECTION

Problem-World Modeling: Entities, Relations, Rules, and System Boundaries

論文編號： PU-1-02

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

前篇將程式定義為「可執行問題模型」，並指出來源文字只是程式在特定語言中的投影。本篇進一步處理程式設計中最常被跳過、卻又決定整個系統結構的前置工作：問題世界建模。

本文將問題世界定義為：

> 系統為了理解、判斷與改變某一現實或數位領域，而主動選取、分類、關聯、約束與排除後形成的可執行認知世界。

形式上：

> 【mathcal W_P
> =
> <
> E,
> A,
> R,
> S,
> V,
> C,
> B,
> X,
> U,
> H
> >】

其中：

- E：實體；
- A：屬性；
- R：關係；
- S：狀態；
- V：事件與變化；
- C：規則與約束；
- B：系統邊界；
- X：外部世界與相依；

- U：未知、爭議與未決內容；
- H：歷史、承諾與版本。

本文主張，問題世界不是現實本身，也不是資料庫 schema、類別圖、需求清單或使用者故事。它是一個經過目的選擇與責任界定的中介世界：

> 【mathcal P
> supsetneq
> mathcal W_P
> supsetneq
> mathcal M_C】

其中：

- mathcal P：更廣泛的物理與社會世界；
- mathcal W_P：系統選定的問題世界；
- mathcal M_C：實際進入計算模型的結構。

因此，建模第一步不是問「要建立哪些資料表」，而是問：

- 世界裡有哪些東西？
- 哪些東西只是屬性，哪些具有持續身份？
- 哪些關係決定責任、權限與歷史？
- 哪些狀態合法，哪些狀態不可出現？
- 哪些事件由系統處理，哪些只被觀測？
- 哪些規則屬於領域，哪些只是目前實作限制？
- 系統真正負責到哪裡？
- 哪些內容尚未知道、不能決定或不應自動化？

本文提出「實體最小判準」：若某對象具有可追蹤身份、跨時間狀態、關係責任或獨立生命週期，便應被優先視為實體，而非單純欄位。反之，僅為描述性值、無獨立身份且不參與關係的內容，可作為屬性。

本文也區分六類關係：

- 組成關係：部分與整體；
- 擁有關係：控制與歸屬；
- 角色關係：主體在特定情境中的位置；
- 依賴關係：一方需要另一方才能存在或完成；
- 時序關係：事件與狀態的先後；
- 規範關係：權限、責任、義務與禁止。

本文指出，真正複雜的系統通常不是被實體數量壓垮，而是被未明示的關係壓垮。兩個相同實體集合，若關係、權限與責任不同，就形成不同問題世界。

本文進一步建立「規則分層」：

- 世界規則：領域本身或制度所要求的規則；
- 系統規則：為維持一致性、安全與可執行性而建立的規則；
- 實作規則：受目前技術與架構限制而存在的規則；
- 暫時規則：尚待驗證、遷移或人工處理的過渡規則。

若四者不被區分，技術限制就會被誤認為世界必然，暫時補丁也可能固化為永久制度。

本文將系統邊界定義為：

> 【B
> =
> <
> Responsibility,
> Authority,
> Observation,
> Guarantee,
> Dependency,
> NonGuarantee
> >】

邊界不是架構圖上的外框，而是對責任、控制、可觀測、保證、相依與非保證的明示。系統只能對自己能控制或能驗證的狀態負責；對邊界外效果，必須提供證據接口、補償與人類接管，而不能以單一成功旗標僭稱完成。

本文提出「未知保留」與「爭議建模」：成熟問題世界不應強迫所有內容立即進入確定類型，而應正式保留：

- unknown；
- unresolved；
- contested；
- not-applicable；
- external-decision；
- human-review。

本文使用待辦事項、訂單系統、醫療分流、網站部署、AI Agent 與研究計畫等案例，說明如何從開放現實中建立可執行問題世界。本文亦提出可證偽研究綱領，包括實體誤判率、關係遺漏率、規則層級混淆、邊界錯配、未知壓平、世界模型可解釋性及寫碼前建模對維護成本的影響。

本文為下一篇〈資料—狀態—事件—行動〉建立靜態世界骨架：只有先知道世界裡有哪些實體、關係、規則與邊界，才能進一步定義它們如何隨時間變化。

關鍵詞： 問題世界、世界建模、實體、關係、規則、系統邊界、未知保留、程式設計方法論

SECTION

Abstract

The previous paper defined a program as an executable problem model. This paper develops a systematic method for problem-world modeling.

A problem world is defined as an executable cognitive world formed by selecting, classifying, relating, constraining, and excluding aspects of a broader physical or digital domain:

mathcal W_P
=
<
E,

The paper distinguishes entities from attributes, develops six classes of relations, separates world rules, system rules, implementation rules, and temporary rules, and defines system boundaries in terms of responsibility, authority, observation, guarantees, dependencies, and non-guarantees.

It introduces unknown preservation and contested modeling and argues that problem-world modeling must precede data schemas, classes, folders, and source code.

Keywords: problem world, domain modeling, entities, relations, rules, system boundaries, unknown preservation

任何程式都不可能處理全部現實。

它必須選擇：

- 哪些東西值得被命名；
- 哪些關係值得被保存；
- 哪些狀態需要區分；
- 哪些事件需要回應；
- 哪些規則必須執行；
- 哪些外部效果不在自己控制範圍內。

這些選擇共同形成問題世界。

若設計者不主動建模，問題世界仍會出現，只是以隱含方式散落在：

- 欄位名稱；
- 函式參數；
- if 條件；
- UI 文案；
- 權限判斷；
- 資料表；
- 例外處理；

- 開發者記憶。

  因此：

> 【No Explicit World Model
> not⇒
> No World Model】

它只代表世界模型沒有被看見、驗證與共同治理。

本文定義：

> 【mathcal W_P
> =
> <
> E,
> A,
> R,
> S,
> V,
> C,
> B,
> X,
> U,
> H
> >】

SECTION

2.1 實體 E

具有身份、狀態、關係或生命週期的對象。

SECTION

2.2 屬性 A

描述實體的值或特徵。

SECTION

2.3 關係 R

實體間的結構、依賴、角色、權限與時序連接。

SECTION

2.4 狀態 S

問題世界在特定時間的合法配置。

SECTION

2.5 事件 V

使狀態需要重新評估的發生。

SECTION

2.6 規則 C

限制實體、關係、狀態與事件的合法性。

SECTION

2.7 邊界 B

系統的責任、控制、觀測、保證與非保證。

SECTION

2.8 外部世界 X

系統依賴但不完全控制的主體、服務、制度與物理環境。

SECTION

2.9 未知與爭議 U

尚未知道、尚未決定、不適合自動化或存在異議的內容。

SECTION

2.10 歷史 H

實體與規則如何形成、變更、承諾與持續。

問題世界是選定後的中介世界。

> 【mathcal P
> supsetneq
> mathcal W_P
> supsetneq
> mathcal M_C】

SECTION

3.1 現實大於問題世界

現實中存在大量與當前目的無關的內容。

SECTION

3.2 問題世界大於實際模型

即使某些內容已被認為相關，也可能尚未進入程式、資料或介面。

SECTION

3.3 建模不是複製

建模包含：

- 選擇；
- 命名；
- 分類；
- 抽象；
- 忽略；
- 邊界；
- 責任分配。

因此，問題世界建模本身就是一種設計與治理活動。

SECTION

4.1 實體最小判準

若某對象至少符合下列一項，應優先考慮為實體：

- 具有穩定身份；
- 跨時間改變狀態；
- 參與重要關係；
- 承擔權限或責任；
- 有獨立生命週期；
- 需要被歷史追蹤。

SECTION

4.2 屬性判準

若某內容：

- 無獨立身份；
- 只描述另一實體；
- 不承擔關係；
- 不具有獨立生命週期；

  則較適合作為屬性。

SECTION

4.3 實體化錯誤

把所有欄位都做成實體，會造成：

- 過度複雜；
- 無意義身份；
- 關係爆炸；
- 維護成本上升。

SECTION

4.4 屬性化錯誤

把本應具有身份與歷史的對象壓成欄位，則會：

- 失去生命週期；
- 無法追蹤責任；
- 無法表達關係；
- 無法處理版本；
- 無法遷移。

SECTION

4.5 例子

地址可以只是文字屬性。

但若系統需要：

- 多地址；
- 驗證；
- 使用歷史；
- 配送關係；
- 法律地位；

  地址就可能成為實體。

  複雜系統真正困難的部分，往往不是實體數量，而是關係。

SECTION

5.1 組成關係

描述部分與整體：

> PartOf(x,y)

例如：

- 訂單包含訂單項目；
- 專案包含任務；
- 文件包含章節。

  需區分：

- 可獨立存在的部分；

- 隨整體刪除的部分；
- 可被共享的部分。

SECTION

5.2 擁有關係

描述控制、歸屬或處分權：

> Owns(a,x)

擁有可能包括：

- 建立權；
- 修改權；
- 轉移權；
- 刪除權；
- 使用權。

SECTION

5.3 角色關係

同一主體在不同情境可扮演不同角色：

> Role(a,c)=r

例如同一人可以是：

- 買家；
- 賣家；
- 審核者；

- 管理者；
- 被影響者。

  角色不應被過早固化為人格身份。

SECTION

5.4 依賴關係

> DependsOn(x,y)

若 y 失效，x 的存在、完成或合法性會受影響。

需區分：

- 強依賴；
- 弱依賴；
- 可替代依賴；
- 外部依賴。

SECTION

5.5 時序關係

事件與狀態具有：

- 先於；
- 晚於；
- 同時；
- 包含；
- 重疊；

- 期限。

時序關係不等於單一時間戳。

SECTION

5.6 規範關係

描述：

- 權限；
- 義務；
- 責任；
- 禁止；
- 委任；
- 申訴；
- 承諾。

規範關係會直接決定系統中的合法行動空間。

SECTION

6.1 相同實體，不同世界

假設兩個系統都具有：

User
Document
若系統 A 中使用者擁有文件，系統 B 中使用者只能暫時存取文件，兩者就是不同世界。

SECTION

6.2 關係決定責任

文件內容錯誤時，誰負責？

- 作者；
- 所有者；
- 發布者；
- 審核者；
- 平台；

答案取決於關係，而不是欄位。

SECTION

6.3 關係具有時間

擁有、委任與角色通常有：

- 生效時間；
- 終止時間；
- 暫停；
- 版本；
- 來源證據。

SECTION

6.4 關係本身可能成為實體

若關係具有：

- 自己的身份；
- 狀態；
- 權限；

- 歷史；
- 生命週期；

它應被建模為關係實體。

例如：

- 合約；
- 訂閱；
- 婚姻；
- 任務委任；
- 會員資格。

SECTION

7.1 使用者不是唯一主體

問題世界中的主體可以包括：

- 直接使用者；
- 管理者；
- 受影響者；
- 第三方服務；
- 審核者；
- 法規機構；
- 未來維護者；
- Agent。

SECTION

7.2 角色不等於權限集合全部

角色可以提供預設權限，但實際行動還可能受：

- 情境；
- 對象；
- 時間；
- 委任；
- 風險；
- 人類批准；

  約束。

SECTION

7.3 沉默主體

沒有登入系統的人，也可能受系統影響。

例如：

- 被推薦系統降低曝光的創作者；
- 被公共模型分類的居民；
- 被 AI 產生內容提及的人。

SECTION

7.4 主體可見性

問題世界應記錄：

actors:

direct_users: []
operators: []
affected_non_users: []
external_authorities: []
future_maintainers: []

SECTION

8.1 世界規則

源自領域、制度或現實要求。

例如：

- 已付款訂單才能出貨；
- 未成年人需要特定同意；
- 一項醫療決定需由合格專業者確認。

SECTION

8.2 系統規則

為維持系統一致、安全與責任建立。

例如：

- 同一事件不得重複處理；
- 高風險操作需要雙重批准；
- 關鍵狀態修改必須留存證據。

SECTION

8.3 實作規則

源自目前技術限制。

例如：

- 檔案大小上限；
- 批次每次最多處理一百筆；
- 只支援特定格式。

SECTION

8.4 暫時規則

為遷移、試驗或人工過渡建立。

例如：

- 舊資料暫由人工審核；
- 新功能只開放測試群體；
- 某欄位等待重新分類。

SECTION

8.5 層級混淆

若實作規則被說成世界規則，使用者會誤以為技術限制不可改變。

若暫時規則沒有期限，便可能固化為制度。

完整規則至少包含：

> Rule
> =
> <
> Trigger,
> Condition,
> Actor,
> Action,
> Effect,
> Evidence,
> Exception

SECTION

9.1 觸發

何時需要評估規則。

SECTION

9.2 條件

何種世界狀態下成立。

SECTION

9.3 主體

誰要求或執行行動。

SECTION

9.4 行動

允許、禁止或要求做什麼。

SECTION

9.5 效果

狀態與外部世界如何改變。

SECTION

9.6 證據

如何證明規則已被正確執行。

SECTION

9.7 例外

哪些情境需要人工或特殊程序。

SECTION

10.1 不變量

不變量描述無論經過何種合法行動都必須成立的條件：

> Inv(S□)=true

SECTION

10.2 轉換規則

> S_(t+1)
> =
> F
> (
> S□,
> e□,
> a□
>)

SECTION

10.3 規則衝突

兩條規則可能同時要求不同結果。

例如：

- 保護隱私；
- 滿足稽核；
- 提供資料可攜；
- 遵守刪除要求。

SECTION

10.4 衝突不能靠 if 順序解決

若規範衝突只由程式碼執行順序決定，就把制度選擇隱藏成技術偶然。

應明示：

- 優先原則；
- 適用條件；
- 人類決策；
- 例外程序。

本文定義：

> 【B
> =
> <
> Responsibility,
> Authority,
> Observation,
> Guarantee,
> Dependency,
> NonGuarantee
> >】

SECTION

11.1 責任

系統承諾處理哪些狀態與失敗。

SECTION

11.2 控制權

系統能直接改變哪些對象。

SECTION

11.3 觀測範圍

系統能可靠知道哪些結果。

SECTION

11.4 保證

在明示條件下，系統能保證什麼。

SECTION

11.5 相依

需要哪些外部服務、主體與物理條件。

SECTION

11.6 非保證

系統明確不能承諾什麼。

SECTION

12.1 能控制但不能完整觀測

系統可能發出設備命令，卻沒有感測器確認物理效果。

SECTION

12.2 能觀測但不能控制

監測系統可以看見氣候、交通或市場變化，卻不能直接改變它們。

SECTION

12.3 有責任但依賴外部

訂單平台可能對使用者承擔服務責任，但配送由外部物流完成。

SECTION

12.4 邊界矩陣

對象	可控制	可觀測	承擔責任
本地資料庫	高	高	高
第三方付款	低	中	部分
物流配送	低	低至中	協調責任
使用者真實感受	無	低	不可直接保證

SECTION

13.1 外部不等於不重要

位於邊界外的對象，仍可能決定系統成功。

例如：

- 銀行；
- 郵件服務；
- 使用者裝置；
- 法律程序；
- 實體配送；
- 人類批准。

SECTION

13.2 相依描述

每個重要相依應記錄：

```
dependency:
 name: "payment-provider"
 provides:
  - "payment authorization"
 system_controls: false
 observable:
  - "provider response"
 not_observable:
  - "final bank settlement"
 failure_modes:
  - "timeout"
  - "duplicate callback"
  - "delayed settlement"
```

SECTION

13.3 外部成功不可假定

第三方 API 回傳成功，通常只能證明其接受請求或完成某階段。

SECTION

13.4 相依替代性

需區分：

- 唯一相依；
- 可替代相依；
- 多供應商；
- 人工備援；
- 無替代方案。

SECTION

14.1 未知不是錯誤值

> Unknown
> ≠
> Null
> ≠
> False
> ≠
> 0

它們具有不同語意。

SECTION

14.2 爭議狀態

不同主體可能對同一事實有不同主張。

例如：

- 訂單是否已交付；
- 文件是否已批准；
- 任務是否真正完成；

- AI 是否正確引用來源。

  爭議不應被強迫壓成單一布林值。

SECTION

14.3 未決狀態

某些問題需要：

- 更多資料；
- 人類選擇；
- 外部判定；
- 未來事件；

  才能完成。

SECTION

14.4 正式保留

問題世界應允許：

unknown
unresolved
contested
not-applicable
external-decision
human-review

SECTION

14.5 未知的生命週期

未知也應記錄：

- 來源；
- 發現時間；

- 影響；
- 責任人；
- 下一步；
- 是否阻止執行。

SECTION

15.1 假設必然存在

任何模型都依賴假設。

例如：

- 身份驗證服務可信；
- 時鐘差異可接受；
- 使用者能理解提示；
- 外部服務最終會回覆；
- 某狀態不會同時發生。

SECTION

15.2 明示假設

> mathcal A_H
> =
> {
> a⊠,a⊠,…,a⊠
> }

每個假設應記錄：

- 內容；

- 來源；
- 依據；
- 適用域；
- 驗證方式；
- 失效後果。

SECTION

15.3 假設不是保證

假設可用於簡化模型，但必須知道它可能失效。

SECTION

15.4 隱含假設風險

事故往往不是程式違反已知規則，而是世界違反設計者從未寫下的假設。

SECTION

16.1 當前狀態不夠

兩個當前狀態完全相同的實體，可能因歷史不同而具有不同責任。

SECTION

16.2 歷史來源

- 建立；
- 委任；
- 核准；
- 修改；

- 撤回；
- 爭議；
- 補償；
- 遷移。

SECTION

16.3 承諾

承諾會限制未來合法行動。

例如已接受付款後，系統不能任意把訂單當成從未存在。

SECTION

16.4 版本

規則與模型更新後，需要知道：

- 哪個版本處理哪個事件；
- 舊資料是否遷移；
- 舊承諾是否仍有效；
- 新規則是否可追溯適用。

本文提出十一步流程。

SECTION

第一步：寫出原始目的

不是功能，而是希望改變什麼。

SECTION

第二步：列出直接主體與受影響主體

包括不使用系統但承受後果者。

SECTION

第三步：找出具有身份與生命週期的實體

避免從資料表開始。

SECTION

第四步：列出屬性與證據

區分描述值與權威證明。

SECTION

第五步：建立重要關係

特別是擁有、責任、委任與依賴。

SECTION

第六步：列出合法、非法、未知與爭議狀態

不要只列成功狀態。

SECTION

第七步：列出事件與行動

區分世界發生與系統命令。

SECTION

第八步：分層建立規則

區分世界、系統、實作與暫時規則。

SECTION

第九步：劃定責任、控制與觀測邊界

明示保證與非保證。

SECTION

第十步：建立假設與未知帳本

將隱含風險外部化。

SECTION

第十一步：才開始建立資料、模組、介面與來源碼

問題世界建模不必產生單一巨大文件。

可使用多種投影：

- 世界詞彙表；
- 實體—關係圖；
- 狀態圖；
- 規則表；
- 邊界矩陣；
- 主體—權限表；
- 外部相依清單；

- 假設帳本；
- 未知與爭議清單；
- 歷史與版本圖。

  每一種投影回答不同問題。

SECTION

19.1 原始目的

降低遺忘，協助個人與團隊協調工作。

SECTION

19.2 實體

- 使用者；
- 任務；
- 專案；
- 委任；
- 提醒；
- 完成證據。

SECTION

19.3 關係

- 使用者擁有任務；
- 任務屬於專案；
- 任務被委任給執行者；

- 審核者接受完成證據。

SECTION

19.4 狀態

proposed
active
blocked
completed
cancelled
disputed

SECTION

19.5 邊界

系統可保證：

- 保存任務狀態；
- 發出提醒請求；
- 記錄完成主張。

系統不能直接保證：

- 使用者真正看見提醒；
- 現實工作確實完成；
- 任務本身值得完成。

SECTION

20.1 實體

- 客戶；
- 商家；

- 商品；
- 庫存單位；
- 訂單；
- 付款；
- 配送；
- 爭議。

SECTION

20.2 重要關係

訂單不是單純屬於客戶，它同時連接：

- 商家履約責任；
- 付款提供者；
- 物流委任；
- 商品與庫存；
- 退款與補償。

SECTION

20.3 規則層級

世界規則：

- 沒有付款承諾不得出貨。

系統規則：

- 付款通知必須冪等處理。

實作規則：

- 批次更新每次最多一百筆。

暫時規則：

- 舊訂單爭議由人工處理。

SECTION

20.4 外部世界

系統不能完全控制：

- 銀行結算；
- 實體物流；
- 客戶是否收到；
- 法律爭議結果。

SECTION

21.1 原始目的

在有限資源下，協助辨識需要優先處理的病人，同時保留專業判斷與病人權利。

SECTION

21.2 直接主體

- 病人；
- 醫師；
- 護理師；
- 行政人員。

SECTION

21.3 受影響主體

- 家屬；
- 等候中的其他病人；
- 醫療機構；
- 緊急資源提供者。

SECTION

21.4 實體與屬性

病人是實體。

症狀若需要跨時間追蹤、來源與嚴重度歷史，也可成為獨立觀測實體，而不是單一文字欄位。

SECTION

21.5 規則

系統可以計算風險，但不得將模型分數直接等同於完整醫療決定。

SECTION

21.6 未知保留

需要正式表達：

- 資料不足；
- 症狀矛盾；
- 模型適用域外；
- 病人拒絕；

- 等待專業覆核。

SECTION

22.1 問題世界

網站部署不只是把檔案上傳。

它包含：

- 原始碼版本；
- 建置產物；
- 設定；
- 憑證；
- 網域；
- 環境；
- 發布；
- 健康檢查；
- 回滾；
- 使用者流量。

SECTION

22.2 實體

- release；
- environment；
- deployment；
- artifact；

- approval；
- incident。

SECTION

22.3 關係

部署使用某一 release，作用於某一 environment，並由特定批准授權。

SECTION

22.4 邊界

部署系統可保證：

- 產物被傳送；
- 服務程序已啟動；
- 健康檢查通過。

不能直接保證：

- 所有使用者都能正常操作；
- 外部 DNS 已完全傳播；
- 新功能符合商業目的。

SECTION

23.1 問題世界

Agent 世界至少包含：

- 委任者；
- Agent 身分；

- 目標；
- 任務；
- 記憶；
- 計畫；
- 工具；
- 權限；
- 預算；
- 證據；
- 檢查點；
- 人類接管。

SECTION

23.2 角色關係

模型不是 Agent。

工具也不是 Agent。

Agent 是在 Runtime 中承擔任務、持有狀態與接受治理的持續數位實體。

SECTION

23.3 規則分層

世界規則：

- 不得永久替主體完成不可撤回選擇。

系統規則：

- 不可逆操作需要人工批准。

實作規則：

- 每輪最多呼叫特定數量工具。

  暫時規則：

- 新工具先在沙盒測試。

SECTION

23.4 邊界

Agent 能保證提出計畫、執行獲准工具與保存證據。

它不能保證外部世界完整符合使用者目的。

SECTION

24.1 問題世界

研究不只是文件集合。

它包含：

- 問題；
- 假設；
- 定義；
- 命題；
- 證據；
- 反例；
- 實驗；
- 失敗；
- 版本；

- 研究者；
- 後續節點。

SECTION

24.2 關係

論文引用證據。

命題依賴定義。

實驗檢驗假設。

失敗紀錄限制後續路徑。

SECTION

24.3 邊界

研究系統可以保存與組織推理，不能因文件生成完成就保證真理。

SECTION

24.4 未知

unproven
partially-supported
contradicted
open-question
requires-experiment
必須是正式研究狀態。

SECTION

25.1 問題世界先於資料模型

資料模型應來自問題世界，而不是反過來。

SECTION

25.2 一對多投影

一個實體可被投影到多個資料表、文件、索引與事件流。

SECTION

25.3 多對一投影

多個問題世界概念也可能因效能或儲存被合併，但不能因此失去語意區分。

SECTION

25.4 資料遷移不是世界遷移全部

若 schema 改變，還需處理：

- 身份；
- 關係；
- 規則；
- 歷史；
- 承諾；
- 權限；
- 未知狀態。

SECTION

26.1 介面是世界投影

UI 告訴使用者哪些實體、狀態與行動存在。

SECTION

26.2 介面缺席

若某狀態在模型中存在，但介面沒有顯示，使用者仍可能無法理解或操作它。

SECTION

26.3 多角色介面

同一問題世界應對不同角色提供不同投影：

- 使用者；
- 管理者；
- 審核者；
- 維護者；
- 稽核者。

SECTION

26.4 介面不得假造邊界

介面顯示「已完成」，若實際只是等待外部結果，就造成世界敘事錯配。

SECTION

27.1 模組應承接責任

模組不是把相似檔案放在一起，而是承擔某一世界責任。

SECTION

27.2 狀態擁有權

每項核心狀態應有明確維護者。

SECTION

27.3 關係跨模組

跨模組關係需要契約、事件或明確接口。

SECTION

27.4 邊界錯置

若模組邊界切斷一個必須保持一致的世界不變量，系統會以大量同步與補丁重新縫合。

- 需求句子即世界： 一句需求被當成完整領域。
- 資料表先行： 從儲存格式反推世界。
- 實體過度化： 每個名詞都變成有身份對象。
- 屬性過度化： 有生命週期與責任的對象被壓成欄位。
- 關係遺漏： 只列對象，不建所有權、責任與依賴。
- 角色固化： 把情境角色永久寫成人格身份。
- 使用者中心狹化： 只看直接使用者，忽略受影響者。
- 世界規則與技術限制混淆： 暫時實作被當成必然。
- 規則藏於 if： 重要制度選擇沒有外部化。
- 規則衝突靠執行順序： 技術偶然取代明示優先原則。
- 邊界只畫外框： 沒有責任、控制與非保證。
- 外部成功假定： 第三方回應被當成最終世界結果。
- 未知歸零： 未知、爭議與不適用被壓成預設值。
- 假設隱形： 重要前提只存在於作者腦中。

- 當前狀態取代歷史： 忽略承諾與責任來源。
- 單一圖全知： 一張 ER 圖被當成完整世界模型。
- 介面僭位： 畫面能操作的被誤認為世界全部。
- 模組先於世界： 先分資料夾與服務，再猜領域責任。
- 自動化壓平爭議： 所有狀態都被迫立即確定。
- 世界模型永久化： 忽略領域、制度與目的會演化。

SECTION

29.1 實體誤判率

從後續需求變更、事故與資料遷移中統計：

- 本應是實體卻被建成屬性；
- 本應是屬性卻被建成實體；

的比例與成本。

SECTION

29.2 關係遺漏率

比較初始模型與後續事故中新增的重要關係：

> R_(relation)
> =
> (|later-discovered critical relations|)/(|all critical relations identified|)

SECTION

29.3 規則層級混淆

測量世界規則、系統規則、實作規則與暫時規則被混用的事件數。

SECTION

29.4 邊界錯配率

比較系統宣稱的保證與其實際控制、觀測及驗證能力。

SECTION

29.5 未知壓平率

統計 unknown、contested、not-applicable 等狀態被轉為零、否定或預設值的比例。

SECTION

29.6 沉默主體召回率

檢查問題世界能否辨識未直接使用系統但受到結果影響的主體。

SECTION

29.7 假設失效事故率

追蹤事故中有多少源於未明示或未重新驗證的世界假設。

SECTION

29.8 關係實體化效益

比較將重要長時關係建模為欄位與獨立實體，對歷史、權限與維護的影響。

SECTION

29.9 寫碼前建模實驗

比較兩組團隊：

- 直接從功能與資料表開始；
- 先完成問題世界模型；

  在變更、除錯、交接與邊界說明上的差異。

SECTION

29.10 世界模型可解釋性

測量新成員能否在有限時間內回答：

- 世界有哪些實體；
- 哪些狀態非法；
- 誰有何責任；
- 系統保證到哪裡。

SECTION

29.11 模型演化成本

研究具有版本、未知與假設帳本的模型，是否能降低 schema、流程與模組演化成本。

SECTION

29.12 多投影一致性

比較世界詞彙表、ER 圖、狀態圖、規則表、程式碼與 UI 的語意差異。

- 問題世界不是現實本身，而是為特定目的選取與組織的中介世界。
- 沒有明示世界模型，不代表系統沒有世界模型。
- 問題世界應先於資料表、類別、模組與來源碼。

- 具有身份、狀態、關係責任或生命週期的對象，應優先被視為實體。
- 無獨立身份且只描述其他實體的內容，較適合作為屬性。
- 相同實體集合在不同關係下會形成不同問題世界。
- 關係可能具有自己的身份、狀態、權限與歷史。
- 直接使用者不是唯一需要被建模的主體。
- 世界規則、系統規則、實作規則與暫時規則必須分離。
- 重要規則不能只隱藏在 if、執行順序或人工習慣中。
- 邊界是責任、控制、觀測、保證、相依與非保證的組合。
- 可控制、可觀測與應負責三者並不相同。
- 外部對象雖在系統邊界外，仍可能決定系統成功。
- unknown、null、false 與 zero 具有不同語意。
- 爭議狀態不應被強迫壓成單一確定值。
- 假設必須具備來源、適用域、驗證方式與失效後果。
- 當前狀態不能取代歷史、承諾與規則版本。
- 問題世界需要多投影共同呈現，而非依賴單一圖表。
- 世界模型應能演化、分支、版本化並保留未知。

20.

> 【系統性程式設計的第一步，】
>
> 【不是決定怎麼寫，】
>
> 【而是決定程式究竟認為世界是什麼。】

SECTION

31.1 承接 PU-1-01

前篇將程式定義為可執行問題模型。

本篇展開其中的「問題世界」，建立：

- 實體；
- 屬性；
- 關係；
- 規則；
- 邊界；
- 外部相依；
- 未知；
- 歷史。

SECTION

31.2 銜接 PU-1-03

本篇主要建立世界的靜態骨架。

下一篇將把它推進為動態四元結構：

> 【mathbb S_P
> =
> <
> D,S,E,A
> >】

並集中回答：

- 資料與狀態有何不同；
- 事件與命令有何不同；
- 行動如何改變狀態；
- 不變量如何跨時間維持；
- 失敗與重試如何進入狀態動力。

多數系統錯誤在來源碼中出現，卻不一定在來源碼中誕生。

它們可能更早產生於：

- 把重要對象當成欄位；
- 遺漏責任關係；
- 把角色固化成身份；
- 把技術限制誤認為世界規則；
- 把外部服務回應當成最終結果；
- 把未知壓成預設值；
- 把邊界外效果納入虛假保證；
- 把歷史承諾壓成當前狀態。

所以，真正的系統性程式設計不能先問：

要建立哪幾個 class？
要開哪幾張資料表？
要使用哪個框架？

而應先問：

世界裡有哪些具有身份與生命週期的東西？
它們之間有哪些責任、擁有、依賴與時序關係？
哪些狀態合法？
哪些規則來自領域，哪些只是目前技術限制？
哪些外部效果系統無法控制？
哪些內容尚未知道、仍有爭議或必須由人類決定？

本文將問題世界模型收束為：

> 【Problem World
> =
> Entities
> +
> Relations
> +
> Legal States
> +
> Rules
> +
> Boundaries
> +
> Unknowns
> +
> History】

這個模型不是為了在寫碼前增加形式負擔，而是把原本必然存在於作者腦中、散落於程式碼中的假設，轉化為團隊可以共同檢查、反駁與修改的外部結構。

因此：

> 【問題世界模型，】

> 【才是程式真正的第一份原始碼。】

它不直接由 CPU 執行，卻決定後續所有可執行結構究竟代表哪一個世界。

```
problem_world:
 world_id: "order-world-v1"
 purpose:
  - "coordinate purchase, payment, fulfillment, and dispute"
 actors:
  direct:
   - "customer"
   - "merchant"
  affected:
```

```
    - "recipient"
    - "support-agent"
  external:
    - "payment-provider"
    - "logistics-provider"
 entities:
  order:
    identity: true
    lifecycle: true
  payment:
    identity: true
    lifecycle: true
  delivery:
    identity: true
    lifecycle: true
 relations:
  - "customer places order"
  - "merchant fulfills order"
  - "payment authorizes order"
  - "delivery fulfills shipment"
 states:
  order:
    - "created"
    - "payment-pending"
    - "paid"
    - "fulfilling"
    - "completed"
    - "disputed"
    - "cancelled"
 unknowns:
  - "physical delivery confirmation"
  - "recipient actual acceptance"
 boundary:
  guarantees:
    - "local state consistency"
    - "event traceability"
  non_guarantees:
    - "final bank settlement"
    - "physical receipt by recipient"
rules:
 world:
  - id: "paid-before-fulfillment"
    reason: "merchant fulfillment obligation"
 system:
  - id: "payment-event-idempotency"
```

```
    reason: "prevent duplicate transition"
  implementation:
   - id: "batch-limit-100"
     reason: "current infrastructure capacity"
     revisable: true
  temporary:
   - id: "legacy-dispute-manual-review"
     expires_after: "migration-complete"
```

項目 可控制 可觀測 可保證 非保證

本地訂單狀態 高 高 一致性與歷史 外部履約

付款 API 低 中 請求與回應紀錄 最終銀行結算

物流配送 低 低至中 委任與狀態紀錄 實體收件

使用者滿意 無 低 收集回饋 真實滿意

```
world_knowledge:
  unknowns:
   - id: "delivery-received"
     status: "external-verification"
     blocks_completion: true
  contested:
   - id: "item-condition"
     claims:
       customer: "damaged"
       merchant: "intact"
  assumptions:
   - id: "provider-callback-eventual"
     statement: "payment provider eventually retries callback"
     evidence: "provider contract v3"
     failure_effect: "manual reconciliation required"
```

- 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，《三重宇宙論：意圖、計算與物理存在的基本區分》，2026。
- Neo.K with Aletheia，《表示落差：意圖、計算模型與物理現實之間的不可完備映射》，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，《生成與限制：計算機宇宙如何創造並封閉可能性》，2026。
- Neo.K with Aletheia，《我們到底想要什麼：計算目的、價值邊界與世界選擇》，2026。
- Neo.K，《世界編織論》，2026。

SECTION

一般理論背景

- Jackson, M., Problem Frames, 2001.
- Evans, E., Domain-Driven Design, 2003.
- Parnas, D. L., “On the Criteria To Be Used in Decomposing Systems into Modules,” 1972.
- Harel, D., “Statecharts: A Visual Formalism for Complex Systems,” 1987.
- Lamport, L., Specifying Systems, 2002.
- Meyer, B., Object-Oriented Software Construction, 1997.
- Fowler, M., Analysis Patterns, 1997.
- Booch, G., Rumbaugh, J., and Jacobson, I., The Unified Modeling Language User Guide, 1999.
- Checkland, P., Systems Thinking, Systems Practice, 1981.

- Dijkstra, E. W., “On the Role of Scientific Thought,” 1974.
- Brooks, F. P., “No Silver Bullet,” 1986.

SECTION

v0.1 — 2026-07-27

- 完成十八篇地基論文第八篇。
- 建立問題世界十元模型。
- 區分問題世界、物理世界與計算模型。
- 提出實體與屬性最小判準。
- 建立組成、擁有、角色、依賴、時序與規範六類關係。
- 區分直接使用者、受影響者、外部權威與未來維護者。
- 建立世界、系統、實作與暫時四層規則。
- 形式化系統邊界六元結構。
- 區分控制、觀測、責任、保證與非保證。
- 建立未知、爭議、未決與假設帳本。
- 提出十一步問題世界建模流程。
- 加入待辦、訂單、醫療、網站部署、AI Agent 與研究計畫案例。
- 提出二十類失敗模式與十二項可證偽研究方向。
- 完成與 PU-1-03《資料—狀態—事件—行動》的銜接。