失敗也是程式：驗證、可觀測、恢復與長期維護

Failure Is Also Program: Verification, Observability, Recovery, and Long-Term Maintenance

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

https://thisoneisneok.com/html/papers/pu-1-06.html

SECTION

Failure Is Also Program: Verification, Observability, Recovery, and Long-Term Maintenance

論文編號： PU-1-06

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

第 1 冊前五篇依序將程式定義為可執行問題模型，建立問題世界、資料—狀態—事件—行動四元動力、責任模組與契約，以及可視化外部結構記憶。本篇作為第 1 冊封頂篇，集中處理最容易被錯誤地視為「例外」或「開發後期工作」的部分：失敗、驗證、可觀測、恢復與長期維護。

本文提出：

> 【Failure Handling
> ≠
> Peripheral Exception Code】

以及更核心的命題：

> 【Failure Is Also Program】

如果程式是一個可執行問題模型，那麼失敗就不是偏離程式的外部事件，而是問題世界中可預期、可分類、可觀測、可恢復或不可恢復的正式狀態與路徑。只描述成功流程的程式，並沒有描述完整世界。

本文提出系統失敗結構：

> 【mathcal F_P
> =
> <
> D,
> C,
> S,
> I,
> P,
> O,
> R,
> E,
> L,
> M
> >】

其中：

- D：Detection，失敗如何被發現；
- C：Classification，失敗如何分類；
- S：Scope，影響範圍；
- I：Impact，對主體、狀態與外部世界的影響；

- P：Propagation，失敗如何傳播；
- O：Observability，系統如何呈現真實狀態；
- R：Recovery，如何重試、回復、降級、補償或重建；
- E：Evidence，完成與失敗的證據；
- L：Learning，事故如何轉化為結構改進；
- M：Maintenance，系統如何跨版本與時間持續可理解。

本文區分：

- 錯誤 Error：系統內部的錯誤條件、違反假設或非法狀態；
- 失敗 Failure：系統無法履行某項契約、責任或世界效果；
- 事故 Incident：失敗已經影響使用者、受影響主體、制度或物理世界；
- 災難 Catastrophe：失敗超過既有恢復能力並造成大範圍或不可逆影響。

本文主張：

> 【Error
> not⇒
> Failure
> not⇒
> Incident
> not⇒
> Catastrophe】

錯誤若被局部吸收，不必成為外部失敗；失敗若被隔離與恢復，不必成為事故；事故若具有邊界、回復與補救，不必升級為災難。系統韌性不等於沒有錯誤，而是能阻止錯誤沿責任與依賴鏈無限制擴散。

本文建立「驗證階梯」：

- 結構驗證：格式、型別與 schema；
- 語意驗證：資料與操作是否具有合法問題世界意義；

- 狀態驗證：前置條件與不變量是否成立；
- 權限驗證：主體是否具有合法行動權；
- 效果驗證：權威狀態是否真正改變；
- 外部驗證：數位或物理世界效果是否有證據；
- 目的驗證：受影響主體是否接受結果，原始目的是否仍成立。

因此：

> 【TestPass
> not⇒
> WorldCorrect】

測試通過只能證明已測條件下的某些性質，不能自動證明問題世界完整、外部效果成立或目的合理。

本文也將可觀測性定義為：

> 系統能否僅憑外部可取得的證據，重建其內部狀態、重要事件、責任、因果、影響與恢復位置的能力。

形式上：

> 【mathcal O_P
> =
> <
> State,
> Event,
> Trace,
> Metric,
> Log,
> Evidence,
> Context,
> Ownership
> >】

可觀測性不是「有日誌」或「有監控儀表板」。若無法回答：

- 現在世界處於什麼狀態；
- 哪個事件造成變化；
- 誰執行；
- 哪條規則與契約生效；
- 影響哪些主體；
- 是否部分成功；
- 誰負責回復；

  則系統仍不可觀測。

  本文提出恢復五分法：

- Retry：在相同行動身份下重試，不重複世界效果；
- Rollback：回到先前數位狀態；
- Compensation：以新行動抵銷已發生效果；
- Degradation：降低能力以保存核心責任；
- Reconstruction：由事件、快照、備份與外部證據重建世界。

  本文特別強調：

> 【Rollback_D
> not⇒
> Rollback_W】

數位狀態可以回滾，但通知、付款、物理操作、公開資訊與人類決策可能已經發生。成熟恢復設計必須區分可撤銷、可補償、只能說明、以及完全不可逆的效果。

本文進一步提出「維護不是修補，而是持續重新理解」。長期維護包含：

- 規則演化；

- schema 與契約版本；
- 相依更新；
- Runtime 漂移；
- 權限變化；
- 視圖更新；
- 知識交接；
- 技術債與限制性債務；
- 事故學習；
- 退役與資料遷移。

本文將維護成熟度表示為：

> 【Maintainability
> =
> Comprehensibility
> +
> Traceability
> +
> Reversibility
> +
> Replaceability
> +
> Historical Memory】

若系統只能由原作者修改、無法知道變更影響、無法局部回復、不能替換相依、沒有歷史決策記憶，即使目前可運作，也不具有長期可維護性。

本文使用訂單付款、網站部署、AI Agent、醫療系統、研究平台與多團隊服務等案例，分析失敗如何成為正式狀態。本文最後提出可證偽研究綱領，包括失敗檢出延遲、錯誤到事故轉化率、可觀測性覆蓋、恢復成功率、補償完整性、事故定位時間、責任辨識、變更失敗率、知識單點、視圖漂移、退役殘留與事故學習閉環。

本文作為第 1 冊結論，將系統性程式設計收束為：

> 【Problem World
> →
> Dynamic State
> →
> Responsibility
> →
> Contract
> →
> Shared Understanding
> →
> Failure and Recovery】

下一冊《程式語言的本質》將在此方法論地基上追問：如果程式是可執行問題模型，那麼程式語言真正應表示的，究竟是文字指令、結構、狀態轉移、效果、世界差分，還是意圖本身？

關鍵詞： 失敗工程、驗證、可觀測性、恢復、補償、事故、維護、韌性、長期軟體系統

SECTION

Abstract

This paper concludes Volume 1 by arguing that failure handling is not peripheral exception code. Failure is part of the program.

A system failure structure is modeled as:

> mathcal F_P
> =
> <
> D,
> C,
> S,
> I,
> P,
> O,
> R,

The paper distinguishes errors, failures, incidents, and catastrophes; develops a seven-level verification ladder; defines observability as the ability to reconstruct state, events, responsibility, causality, impact, and recovery position from external evidence; and distinguishes retry, rollback, compensation, degradation, and reconstruction.

Maintainability is defined as the combination of comprehensibility, traceability, reversibility, replaceability, and historical memory.

Keywords: failure engineering, verification, observability, recovery, maintenance, resilience

許多設計文件呈現：

```
輸入
→ 驗證
→ 執行
→ 成功
```

失敗則被濃縮為：

```
catch error
return 500
```

但真實系統可能出現：

- 輸入有效但世界狀態已改變；
- 行動已部分執行；
- 外部服務成功但回覆遺失；
- 系統內部成功但物理效果未發生；
- 重試造成重複付款；
- 回滾後通知已被使用者讀取；
- 補償失敗；
- 沒有人知道誰負責結案。

所以失敗不是成功流程之外的雜訊。

失敗本身具有：

- 狀態；

- 時間；
- 主體；
- 責任；
- 影響；
- 證據；
- 恢復路徑。

  本文定義：

> 【mathcal F_P
> =
> <
> D,
> C,
> S,
> I,
> P,
> O,
> R,
> E,
> L,
> M
> >】

SECTION

2.1 Detection

何時、由誰、根據何種證據發現異常。

SECTION

2.2 Classification

這是：

- 驗證錯誤；
- 權限拒絕；
- 暫時失敗；
- 永久失敗；
- 結果未知；
- 部分成功；
- 契約違反；
- 世界模型缺失。

SECTION

2.3 Scope

影響：

- 單一請求；
- 單一實體；
- 模組；
- 工作流；
- 全系統；
- 外部主體；
- 物理世界。

SECTION

2.4 Impact

造成：

- 資料不一致；
- 權限錯誤；
- 財產損失；
- 隱私洩漏；
- 延遲；
- 信任下降；
- 不可逆效果。

SECTION

2.5 Propagation

失敗如何跨：

- 呼叫；
- 事件；
- 共享資料；
- 權限；
- 組織責任；

擴散。

SECTION

2.6 Observability

能否知道真實發生了什麼。

SECTION

2.7 Recovery

能否重試、回復、補償、降級或重建。

SECTION

2.8 Evidence

能否證明失敗與恢復結果。

SECTION

2.9 Learning

事故是否形成規則、測試、契約與視圖改進。

SECTION

2.10 Maintenance

系統是否能跨版本持續理解與修復。

SECTION

3.1 錯誤 Error

錯誤是內部條件偏離預期，例如：

- 型別不符；
- 不變量失敗；
- 相依回覆無效；

- 程式缺陷；
- 假設失效。

錯誤可以被局部捕捉與吸收。

SECTION

3.2 失敗 Failure

失敗是系統未能履行某項契約或責任。

例如：

- 無法完成付款授權；
- 無法保存資料；
- 無法在期限內發布；
- 無法提供承諾的查詢新鮮度。

SECTION

3.3 事故 Incident

當失敗已影響：

- 使用者；
- 受影響者；
- 制度；
- 外部服務；
- 物理世界；

便成為事故。

SECTION

3.4 災難 Catastrophe

災難是影響超出既有隔離、回復與補救能力，並可能造成廣泛或不可逆後果。

SECTION

3.5 升級鏈

> 【Error
> not⇒
> Failure
> not⇒
> Incident
> not⇒
> Catastrophe】

系統韌性的目的，是在每一層阻止不必要升級。

SECTION

4.1 不是所有失敗都應轉成例外

可預期世界狀態，例如：

- 付款被拒；
- 使用者取消；
- 資料等待人工審核；
- 外部結果未知；
- 庫存不足；

不應全部被當成技術例外。

SECTION

4.2 失敗狀態

系統應正式表示：

rejected
timed-out
partially-complete
outcome-unknown
compensation-pending
degraded
human-intervention-required
irrecoverable

SECTION

4.3 失敗身份

重要失敗需要：

- failure_id；
- affected_entity；
- originating_action；
- related_events；
- current_owner；
- impact；
- recovery_state。

SECTION

4.4 二元結果不足

只使用 success／failure 會壓平：

- 是否已產生部分效果；

- 是否可安全重試；
- 是否需要補償；
- 是否等待外部證據；
- 是否已影響他人。

本文提出七層驗證。

SECTION

5.1 結構驗證

檢查：

- 型別；
- schema；
- 必填欄位；
- 編碼；
- 範圍。

結構正確不代表語意正確。

SECTION

5.2 語意驗證

檢查資料與操作是否具有合法問題世界意義。

例如金額欄位格式正確，但使用錯誤幣別或錯誤訂單，仍屬語意錯誤。

SECTION

5.3 狀態驗證

檢查：

- 前置條件；
- 當前版本；
- 不變量；
- 是否已終止；
- 是否處於爭議。

SECTION

5.4 權限驗證

確認主體是否能在此情境、時間與作用域下執行。

SECTION

5.5 效果驗證

確認權威狀態真正改變，且未破壞其他不變量。

SECTION

5.6 外部驗證

確認外部數位或物理效果是否有足夠證據。

SECTION

5.7 目的驗證

確認原始目的仍成立，且結果被必要主體接受。

SECTION

5.8 驗證鏈

> V_(structure)
> →
> V_(semantic)
> →
> V_(state)
> →
> V_(permission)
> →
> V_(effect)
> →
> V_(external)
> →
> V_(purpose)

SECTION

6.1 單元測試

驗證局部函式與模組行為。

SECTION

6.2 整合測試

驗證契約、資料、時間與相依互動。

SECTION

6.3 系統測試

驗證整體工作流與權威狀態。

SECTION

6.4 外部效果測試

驗證真實部署、第三方與物理世界效果。

SECTION

6.5 人類接受

驗證使用者、領域專家與受影響者是否接受。

SECTION

6.6 核心命題

> 【TestPass
> not⇒
> WorldCorrect】

測試通過只代表已明示測試在其假設下通過。

SECTION

6.7 測試盲區

未測內容可能包括：

- 未知主體；
- 規則衝突；
- 長時延遲；
- 真實流量；
- 權限組合；
- 歷史資料；
- 外部不可逆效果。

SECTION

7.1 案例測試

測試具體輸入與預期輸出。

SECTION

7.2 性質驗證

驗證跨大量案例必須成立的不變量：

> $\forall\, x \in X,$
>
> $P(x) = \text{true}$

SECTION

7.3 模型檢查

在有限狀態空間中探索：

- 不可達狀態；
- 死鎖；
- 違反不變量；
- 遺漏轉移。

SECTION

7.4 Runtime 證據

測試環境正確，不代表生產環境未漂移。

因此還需：

- 真實追蹤；
- 指標；
- 事件；
- 版本；
- 外部確認。

SECTION

7.5 證據鏈

> Requirement
> →
> Rule
> →
> Test
> →
> Execution
> →
> Evidence
> →
> Acceptance

本文定義：

> 可觀測性是系統能否僅憑可取得的外部證據，重建其內部狀態、重要事件、責任、因果、影響與恢復位置的能力。

形式上：

> 【mathcal O_P
> =
> <
> State,
> Event,

SECTION

8.1 State

現在問題世界與工作流處於何種狀態。

SECTION

8.2 Event

哪些重要事情已經發生。

SECTION

8.3 Trace

一次意圖如何跨模組、契約與相依傳播。

SECTION

8.4 Metric

系統整體趨勢、容量與異常。

SECTION

8.5 Log

局部處理細節與除錯資料。

SECTION

8.6 Evidence

能證明世界效果與責任的資料。

SECTION

8.7 Context

版本、環境、主體、規則與時間。

SECTION

8.8 Ownership

誰應查看、處理、恢復與結案。

SECTION

9.1 日誌

適合記錄局部處理細節，但可能：

- 重複；
- 無語意；
- 缺少身份；
- 難以關聯；
- 被大量噪音淹沒。

SECTION

9.2 指標

適合觀察趨勢：

- 錯誤率；
- 延遲；
- 吞吐；

- 資源；
- 佇列。

但指標通常無法還原單一世界事件。

SECTION

9.3 分散式追蹤

連接一次行動跨模組的處理路徑。

但追蹤拓撲不自動知道領域責任。

SECTION

9.4 領域事件

保存具有問題世界意義的已發生事實。

SECTION

9.5 證據

證據回答「為何相信完成或失敗」。

成熟可觀測性需把四者連接，而不是只大量收集資料。

一個可觀測系統至少應回答：

- 哪個主體提出了哪個行動？
- 行動依據哪個狀態版本？
- 哪個模組作出決策？
- 使用哪條規則與契約版本？
- 產生哪些事件？
- 哪些狀態已改變？

- 哪些外部效果已被證實？
- 哪些結果仍未知？
- 哪些主體受到影響？
- 目前失敗由誰負責？
- 是否可重試、回復或補償？
- 下一個安全行動是什麼？

  若只能回答 CPU、記憶體與 HTTP 狀態碼，系統只是技術可監控，而非世界可觀測。

SECTION

11.1 傳播路徑

失敗可以透過：

- 同步呼叫；
- 非同步事件；
- 共享資料；
- 共用相依；
- 權限；
- 人工流程；

  擴散。

SECTION

11.2 故障域

系統應定義：

> FaultDomain(M)

也就是某模組失敗時，理論上最多影響哪些責任與資源。

SECTION

11.3 隔離手段

包括：

- 資源配額；
- 佇列；
- 逾時；
- 熔斷；
- 沙盒；
- 權限最小化；
- 資料分區；
- 版本隔離。

SECTION

11.4 錯誤預算不是傷害預算

服務可以接受一定錯誤率，但高風險行動不能只以平均可用性合理化。

SECTION

11.5 失敗放大

小錯誤若造成：

- 無限重試；

- 廣播事件；
- 共用相依耗盡；
- 錯誤快取；
- 權限誤用；

  可能被系統架構放大。

  本文區分：

> 【Recovery
> =
> Retry
> ∪
> Rollback
> ∪
> Compensation
> ∪
> Degradation
> ∪
> Reconstruction】

SECTION

12.1 Retry

在相同行動身份下重新嘗試。

條件：

- 行動仍有效；
- 前置狀態未失效；
- 操作具冪等；
- 結果不是未知；

- 重試不擴大傷害。

SECTION

12.2 Rollback

回到先前數位狀態。

適合：

- 尚未產生外部不可逆效果；
- 狀態快照可靠；
- 後續相依可同步回復。

SECTION

12.3 Compensation

以新行動抵銷已發生效果，例如：

- 退款；
- 釋放庫存；
- 撤回發布；
- 恢復權限；
- 修正公告。

SECTION

12.4 Degradation

保留核心責任，暫時降低能力：

- 唯讀模式；

- 關閉推薦；
- 暫停自動化；
- 改由人工審核；
- 使用較舊但穩定版本。

SECTION

12.5 Reconstruction

由：

- 事件歷史；
- 快照；
- 備份；
- 外部證據；
- 人工核對；

重建狀態。

SECTION

13.1 數位回滾

> S_(t+n)
> →
> S⊠

SECTION

13.2 世界不可逆

在此期間可能已經：

- 寄出訊息；
- 扣除款項；
- 公開內容；
- 啟動設備；
- 改變人類決策；
- 建立法律或制度效果。

因此：

> 【Rollback_D
> not⇒
> Rollback_W】

SECTION

13.3 回滾前檢查

需知道：

- 哪些外部事件已發生；
- 哪些主體已看見；
- 哪些契約已履行；
- 哪些資料已被第三方複製；
- 哪些後續行動依賴新狀態。

SECTION

13.4 回滾不是刪除歷史

回滾本身也應成為事件，保留：

- 原因；
- 執行者；
- 範圍；
- 影響；
- 後續補救。

  本文定義補償：

> C_a
> =
> <
> OriginalEffect,
> CounterAction,
> Residual,
> Evidence,
> Owner,
> Deadline
> >

SECTION

14.1 原始效果

必須知道實際發生了什麼，而非只看預期效果。

SECTION

14.2 反向行動

補償行動不必完全對稱。

退款不會消除客戶曾被扣款的經驗與時間成本。

SECTION

14.3 補償殘差

> Residual_C
> =
> Effect_(original)
> +
> Effect_(compensation)

殘差可能包含：

- 時間損失；
- 信任損失；
- 匯率或費用；
- 資訊外洩；
- 人類情緒與決策影響。

SECTION

14.4 補償證據

不能只發出補償命令，還需確認補償已生效。

SECTION

14.5 補償所有者

需明示局部與使用者結案責任。

SECTION

15.1 降級不是任意關功能

系統必須先知道核心責任。

SECTION

15.2 能力層級

例如：

full-service
reduced-automation
manual-review
read-only
evidence-preservation
shutdown

SECTION

15.3 安全降級

降級後仍須保持：

- 身份；
- 權限；
- 不變量；
- 證據；
- 退出；
- 人類接管。

SECTION

15.4 錯誤降級

若為保持可用而關閉安全、稽核或權限檢查，便不是韌性，而是將風險轉嫁給受影響者。

SECTION

16.1 備份不等於可恢復

備份可能：

- 已損壞；
- 缺少金鑰；
- 不含相依；
- 不知版本；
- 無法在新環境啟動；
- 缺少恢復文件。

SECTION

16.2 重建所需

> Reconstruct
> =
> Data
> +
> Schema
> +
> Rules
> +
> Runtime
> +
> Secrets
> +
> History
> +

SECTION

16.3 恢復點與恢復時間

可區分：

- RPO：可接受遺失多少歷史；
- RTO：可接受多久恢復核心責任。

  但仍需評估世界效果，而不只是服務恢復。

SECTION

16.4 恢復演練

沒有演練的恢復計畫只是願望。

SECTION

16.5 人工重建

在資料不一致或證據衝突時，需要正式的人類核對與裁決流程。

SECTION

17.1 部分成功不是邊緣狀態

跨模組、跨制度與長時程工作流中，部分成功很常見。

SECTION

17.2 狀態表示

```
status: "partially-complete"
completed:
 - "payment-authorized"
failed:
 - "inventory-reserved"
```

unknown:
- "notification-received"

SECTION

17.3 下一步

部分成功後可：

- 繼續；
- 重試；
- 補償；
- 等待；
- 人工決策；
- 結束並揭露殘差。

SECTION

17.4 使用者敘事

UI 不應只顯示「發生錯誤」，而應告知：

- 已完成什麼；
- 尚未完成什麼；
- 是否會重試；
- 使用者是否需要行動；
- 何時得到更新。

本文提出：

signal
→ detection
→ triage
→ containment

→ diagnosis
→ recovery
→ compensation
→ verification
→ closure
→ learning

SECTION

18.1 Signal

異常訊號可能來自監控、使用者、稽核或外部服務。

SECTION

18.2 Detection

確認是否為真實異常。

SECTION

18.3 Triage

判斷範圍、影響與優先級。

SECTION

18.4 Containment

先阻止失敗擴散。

SECTION

18.5 Diagnosis

重建因果、規則、狀態與責任。

SECTION

18.6 Recovery

恢復核心責任。

SECTION

18.7 Compensation

處理已發生世界影響。

SECTION

18.8 Verification

證明恢復與補償有效。

SECTION

18.9 Closure

向受影響者與內部責任者完成結案。

SECTION

18.10 Learning

將事故轉化為結構改進。

SECTION

19.1 無責備不等於無責任

事故分析應避免把複雜系統問題簡化為個人疏忽，但仍需保留：

- 決策權；

- 契約；
- 權限；
- 審核；
- 失敗所有權；
- 補救責任。

SECTION

19.2 根因不是單一原因

事故常由：

- 世界模型缺失；
- 隱含假設；
- 邊界錯置；
- 契約模糊；
- 觀測不足；
- 壓力與時間；
- 組織誘因；

共同形成。

SECTION

19.3 學習產物

事故後應更新：

- 問題世界；
- 規則；

- 不變量；
- 契約；
- 測試；
- 可觀測；
- 回復程序；
- 視圖；
- 教學與值班知識。

SECTION

19.4 關閉條件

事故不應只在服務恢復後關閉。

還需確認：

- 世界影響已被理解；
- 受影響者已獲通知或補救；
- 殘留風險有責任者；
- 結構改進已追蹤。

本文提出：

> 【Maintainability
> =
> Comprehensibility
> +
> Traceability
> +
> Reversibility
> +
> Replaceability

SECTION

20.1 可理解性

新維護者能否理解：

- 問題世界；
- 狀態；
- 責任；
- 契約；
- 失敗；
- 非保證。

SECTION

20.2 可追蹤性

能否從目的追蹤到程式與證據，並從事故反向追蹤至世界影響。

SECTION

20.3 可逆性

修改失敗後能否安全停止、回復、補償或分支。

SECTION

20.4 可替換性

相依、模組與 Runtime 能否在保留語意與歷史下被替換。

SECTION

20.5 歷史記憶

系統是否保存：

- 為何如此設計；
- 哪些方案失敗；
- 哪些契約曾存在；
- 哪些承諾仍有效；
- 哪些限制只是歷史遺留。

SECTION

21.1 修補式維護

只處理：

- bug；
- 相依升級；
- 效能；
- 小功能。

SECTION

21.2 結構性維護

還需要重新檢查：

- 問題世界是否改變；
- 實體與關係是否仍正確；
- 規則是否仍正當；
- 模組邊界是否漂移；
- 契約是否仍可理解；

- 視圖是否仍準確；
- 失敗處理是否仍有效。

SECTION

21.3 規範性維護

當制度、權利與受影響主體改變，系統不能只更新程式碼。

SECTION

21.4 物理維護

硬體、網路、能源、憑證與外部服務都會老化或終止。

SECTION

22.1 技術債

為短期交付累積的實作成本。

SECTION

22.2 語意債

程式碼仍能運作，但：

- 名稱與真實意義不符；
- 狀態語意模糊；
- 權威來源不明；
- 規則散落；
- 歷史無法重建。

SECTION

22.3 責任債

系統具有重要工作，但沒有清楚：

- 狀態所有者；
- 失敗所有者；
- 使用者結案所有者；
- 規則修改者。

SECTION

22.4 觀測債

系統新增能力與相依，卻沒有同步增加足夠證據與追蹤。

SECTION

22.5 恢復債

備份、補償與回復方法存在於文件或想像中，但未被演練。

SECTION

22.6 債務帳本

```
maintenance_debt:
 semantic:
  - "paid status has three incompatible meanings"
 responsibility:
  - "no owner for external settlement reconciliation"
 observability:
  - "no trace from customer action to provider callback"
 recovery:
  - "backup restore never tested"
```

SECTION

23.1 修改不只是檔案差分

一項規則變更可能影響：

- 狀態機；
- 契約；
- 權限；
- 歷史資料；
- UI；
- 外部消費者；
- 恢復程序；
- 受影響主體。

SECTION

23.2 影響圖

> Change(c)
> →
> {
> States,
> Rules,
> Contracts,
> Modules,
> Views,
> Tests,
> Data,
> Users
> }

SECTION

23.3 變更前證據

需知道：

- 當前行為；
- 依賴；
- 風險；
- 回復點；
- 退出條件。

SECTION

23.4 變更後驗證

不能只看部署成功，還要檢查：

- 新舊語意；
- Runtime；
- 外部效果；
- 關鍵使用流程；
- 事故指標；
- 回復可用性。

SECTION

24.1 程式版本不足

完整版本還應包含：

- schema；
- 契約；
- 規則；
- 事件；
- Runtime；
- 權限；
- 視圖；
- 資料遷移。

SECTION

24.2 雙版本世界

遷移期間，新舊版本可能同時存在。

需明示：

- 哪些資料由誰寫入；
- 哪些事件可互通；
- 哪些規則適用；
- 如何停止舊版本。

SECTION

24.3 向後與向前相容

結構相容之外，還需檢查：

- 語意；
- 時間；

- 失敗；
- 權限；
- 恢復。

SECTION

24.4 遷移失敗

遷移本身必須有：

- 狀態；
- 檢查點；
- 重試；
- 補償；
- 人工核對；
- 完成證據。

SECTION

25.1 停止服務不等於完成退役

退役需要處理：

- 身份；
- 資料；
- 歷史；
- 契約；
- 權限；
- 外部依賴；

- 使用者出口；
- 法律保留；
- 備份；
- 網域與憑證。

SECTION

25.2 殘留效果

舊 API、舊資料、舊權限與舊文件可能持續存在。

SECTION

25.3 退役流程

announce
→ freeze-new-dependencies
→ migrate
→ verify
→ revoke
→ archive
→ delete-or-retain
→ monitor-residual
→ close

SECTION

25.4 退役證據

需要證明：

- 消費者已遷移；
- 權限已撤銷；
- 敏感資料已依政策處理；
- 舊流量已停止；

- 殘留責任有所有者。

SECTION

26.1 事故

付款提供者已成功扣款，但本地訂單更新超時。

SECTION

26.2 錯誤與失敗

- 網路超時是錯誤訊號；
- 無法確定付款結果是失敗；
- 客戶再次付款造成重複扣款則成為事故。

SECTION

26.3 正式狀態

payment-outcome-unknown
reconciliation-required
duplicate-charge-suspected
compensation-pending

SECTION

26.4 恢復

不能盲目重試付款。

應先以冪等身份向提供者查詢，再：

- 記錄付款；
- 退款；
- 人工核對；

- 通知客戶。

SECTION

26.5 學習

更新命令契約、未知結果狀態、追蹤與客服結案流程。

SECTION

27.1 事故

新版本健康檢查通過，但真實登入流程失敗。

SECTION

27.2 原因

健康檢查只驗證程序存活，未驗證關鍵使用者流程與外部身份相依。

SECTION

27.3 降級

可：

- 停止流量擴張；
- 回切舊版本；
- 保留唯讀；
- 關閉新功能；
- 啟動人工支援。

SECTION

27.4 回滾殘差

部分使用者已建立新版本資料，回滾需處理 schema 與資料相容。

SECTION

27.5 學習

將登入旅程、資料相容與 Runtime 契約加入驗證階梯。

SECTION

28.1 事故

Agent 工具呼叫超時，誤以為未執行，重試後寄出兩封信。

SECTION

28.2 問題

- 工具回覆與世界事件混淆；
- 無行動冪等身份；
- 結果未知未被正式表示；
- Agent 自行決定重試。

SECTION

28.3 改進

工具契約需回傳：

- action_id；
- accepted；
- completed；
- evidence；

- outcome_unknown；
- safe_to_retry。

SECTION

28.4 高風險操作

寄信、發布、付款、刪除等需要操作紀錄與重試治理。

SECTION

28.5 Agent 維護

模型更新後也需重新驗證行動選擇、工具語意與停止條件。

SECTION

29.1 事故風險

模型分數正常產生，但輸入資料過時或病人已發生新變化。

SECTION

29.2 驗證分層

結構正確不代表：

- 資料新鮮；
- 模型適用；
- 專業判斷完成；
- 病人同意；
- 處置已執行。

SECTION

29.3 可觀測性

需追蹤：

- 資料來源；
- 模型版本；
- 醫師決策；
- 病人同意；
- 真實處置；
- 結果觀測。

SECTION

29.4 降級

模型不可用時，應回到明示人工流程，而不是隱性停擺。

SECTION

30.1 失敗類型

- 文獻來源錯誤；
- 實驗不可重現；
- 命題被反例推翻；
- 文件完成但證據不足；
- 版本依賴遺失。

SECTION

30.2 正式研究狀態

unverified
partially-supported
contradicted
replication-failed
evidence-missing
open

SECTION

30.3 恢復

研究失敗通常不能回滾，而應：

- 保留失敗；
- 修正命題；
- 分支；
- 重新實驗；
- 降低斷言；
- 更新依賴圖。

SECTION

30.4 長期維護

研究系統需保存資料、環境、程式、版本、證據與失敗紀錄。

SECTION

31.1 症狀

多個服務同時延遲，沒有團隊知道整體流程所有者。

SECTION

31.2 技術可觀測不足以結案

即使能看到追蹤，也可能不知道：

- 哪個世界責任失敗；
- 哪些使用者受影響；
- 誰應補償；
- 誰能宣布恢復。

SECTION

31.3 改進

建立：

- 責任圖；
- 流程所有者；
- 使用者影響視圖；
- 失敗三層所有權；
- 跨團隊恢復契約。
- 成功流程中心論： 只設計正常路徑，把失敗交給通用例外。
- 所有失敗皆例外： 將拒絕、等待、未知與部分成功當成技術錯誤。
- 例外即處理： 捕捉錯誤後只記錄或轉換狀態碼，沒有恢復責任。
- 錯誤等於事故： 局部可吸收錯誤被過度升級。
- 事故等於錯誤： 已影響使用者的事件仍只以內部 bug 處理。
- 驗證停在格式： schema 正確便宣稱世界語意正確。

- 測試通過即世界正確： 忽略未知條件、外部效果與目的接受。
- 只有案例測試： 沒有不變量、性質、狀態空間與歷史資料驗證。
- 有日誌即可觀測： 無法從大量文字重建狀態、因果與責任。
- 指標無世界語意： 只知道錯誤率升高，不知道影響何種主體與責任。
- 追蹤無契約： 看得見呼叫鏈，卻不知道每一步成功代表什麼。
- 告警無所有者： 訊號產生後無明確處理、升級與結案責任。
- 重試一切： 權限拒絕、永久錯誤與結果未知被盲目重試。
- 回滾萬能論： 忽略外部效果與人類認知已經發生。
- 補償等於抵銷全部： 不記錄時間、費用、信任與資訊殘差。
- 降級即關閉安全： 為保持可用而取消驗證、權限或稽核。
- 備份等於可恢復： 從未演練，缺少版本、金鑰、Runtime 與程序。
- 服務恢復即事故結案： 未處理使用者影響、補償與殘留風險。
- 根因單一化： 把事故歸因於最後一個操作或個人。
- 無責備等於無責任： 為避免個人責難而抹除決策與補救責任。
- 事後文件化： 事故知識只留在報告，未更新規則、測試、契約與視圖。
- 維護即修 bug： 不重新檢查世界、責任、契約與權限。
- 程式碼版本唯一化： 忽略資料、規則、事件、Runtime 與視圖版本。
- 語意債隱形： 名稱與真實世界意義已分離，仍以能運作為由拖延。
- 責任債隱形： 重要流程沒有失敗所有者與使用者結案所有者。
- 恢復債隱形： 回復程序未演練，卻在文件中被視為可用。
- 遷移無狀態： 資料轉換被當成一次腳本，沒有檢查點與補償。
- 退役即關機： 舊權限、API、資料、文件與外部依賴持續殘留。

- 知識英雄化： 只有原作者知道如何診斷與恢復。
- AI 自動恢復越權： Agent 在未知結果或不可逆狀態下自行重試、回滾或補償。

SECTION

33.1 失敗檢出延遲

> T_D
> =
> T_(detected)
> -
> T_(occurred)

測量不同可觀測結構對檢出時間的影響。

SECTION

33.2 錯誤—失敗轉化率

統計內部錯誤中，有多少未被隔離而形成契約失敗。

SECTION

33.3 失敗—事故轉化率

統計系統失敗中，有多少實際影響使用者、制度或物理世界。

SECTION

33.4 可觀測性覆蓋

對重要責任檢查是否可重建：

- 行動；
- 狀態；

- 事件；
- 契約；
- 因果；
- 影響；
- 所有者。

SECTION

33.5 告警可行動率

統計告警中有多少能直接指向：

- 影響範圍；
- 責任者；
- 下一安全行動；
- 恢復程序。

SECTION

33.6 結果未知辨識率

測量超時與外部失聯後，系統是否保留 outcome-unknown，而非誤判為成功或失敗。

SECTION

33.7 重試安全率

> R_(safe-retry)
> =
> 1-
> (|duplicate or amplified effects|)/(|retried actions|)

SECTION

33.8 恢復成功率

分別測量 retry、rollback、compensation、degradation 與 reconstruction 的成功與殘差。

SECTION

33.9 補償完整性

檢查補償是否涵蓋：

- 數位狀態；
- 財務；
- 通知；
- 權限；
- 使用者溝通；
- 長期殘差。

SECTION

33.10 恢復演練差異

比較只具有文件、以及定期演練的團隊，在真實事故中的恢復時間與錯誤率。

SECTION

33.11 事故定位時間

比較只有技術監控，以及具有問題世界、責任、契約、失敗視圖的系統。

SECTION

33.12 責任辨識率

測量事故期間能否快速辨識：

- 局部失敗所有者；
- 流程所有者；
- 使用者結案所有者。

SECTION

33.13 事故學習閉環率

統計事故後建議中，有多少真正進入：

- 模型；
- 程式；
- 測試；
- 契約；
- 視圖；
- 演練。

SECTION

33.14 變更失敗率

測量不同維護成熟度專案的部署失敗、回滾、事故與使用者影響比例。

SECTION

33.15 知識單點風險

統計關鍵診斷與恢復程序中，必須依賴特定個人的比例。

SECTION

33.16 視圖漂移與事故

研究過時架構、責任與恢復視圖是否顯著增加事故定位與錯誤處置時間。

SECTION

33.17 退役殘留率

統計退役後仍存在的：

- 流量；
- 帳號；
- 權限；
- 資料；
- 相依；
- 網域；
- 文件。

SECTION

33.18 AI 恢復治理

比較具備結果未知、冪等、權限與人工批准模型的 Agent，與僅依錯誤訊息自行重試的 Agent 事故率。

- 失敗處理不是程式之外的例外碼，而是程式問題世界的一部分。
- 只描述成功路徑的程式沒有描述完整世界。
- 錯誤、失敗、事故與災難具有不同層級。
- 系統韌性不等於沒有錯誤，而是阻止錯誤不必要地升級與擴散。

- 可預期拒絕、等待、未知與部分成功應被建模為正式狀態。
- 二元成功／失敗不足以表示跨系統世界變化。
- 驗證必須從結構延伸至語意、狀態、權限、效果、外部與目的。

8.

> TestPass
> not⇒
> WorldCorrect

- 測試證明的是已測條件下的性質，而不是完整世界真理。
- 可觀測性必須能重建狀態、事件、因果、責任、影響與恢復位置。
- 日誌、指標、追蹤、領域事件與證據不能互相取代。
- 每一個告警都應具有語意、影響、所有者與下一安全行動。
- 故障域與隔離邊界必須對應責任，而非只對應程序。
- 重試、回滾、補償、降級與重建是不同恢復形式。
- 結果未知時，不應盲目重試可能產生不可逆效果的行動。

16.

> Rollback_D
> not⇒
> Rollback_W

- 補償是新的世界行動，並可能留下不可消除殘差。
- 降級必須保存核心責任、權限、不變量與證據。
- 備份只有在能被驗證重建時，才具有恢復價值。
- 事故關閉必須包含服務恢復、世界補救、證據與學習閉環。

- 維護是持續重新理解問題世界、責任、契約與 Runtime，而非只修補程式碼。
- 可維護性由可理解、可追蹤、可逆、可替換與歷史記憶共同構成。
- 退役是正式程式流程，而不是停止服務或刪除部署。

24.

> 【成熟程式不承諾永不失敗，】

> 【而是承諾失敗能被看見、限制、理解、】

> 【恢復、補救並轉化為下一版本的結構記憶。】

第 1 冊六篇形成以下方法論：

SECTION

35.1 程式本體

> Program
> =
> Executable Problem Model

SECTION

35.2 問題世界

> Problem World
> =
> Entities
> +
> Relations
> +
> Rules
> +

SECTION

35.3 動力結構

> Action
> →
> Event
> →
> State
> →
> Data Projection

SECTION

35.4 系統架構

> Architecture
> =
> Responsibility Modules
> +
> State Ownership
> +
> Contracts
> +
> Failure Ownership

SECTION

35.5 共同理解

> External Structural Memory
> =
> Multi-View Projection
> +

SECTION

35.6 長期生命

> Long-Lived System
> =
> Verification
> +
> Observability
> +
> Recovery
> +
> Learning
> +
> Maintenance

第 1 冊因此形成完整鏈：

> 【問題
> →
> 世界
> →
> 狀態
> →
> 責任
> →
> 理解
> →
> 韌性】

SECTION

36.1 承接 PU-1-01

可執行問題模型若沒有失敗與完成判準，就不是完整程式。

SECTION

36.2 承接 PU-1-02

問題世界必須正式包含未知、爭議、外部相依與假設失效。

SECTION

36.3 承接 PU-1-03

資料、狀態、事件與行動的分離，使結果未知、部分成功、補償與重建得以被正確表示。

SECTION

36.4 承接 PU-1-04

責任模組與契約使失敗具有局部所有者、流程所有者與使用者結案所有者。

SECTION

36.5 承接 PU-1-05

外部結構記憶使事故能從 Runtime 追蹤到狀態、責任、契約、主體與世界目的。

SECTION

36.6 銜接第 2 冊

第 2 冊《程式語言的本質》將研究：

- 為何文字只是程式的一種投影；
- 結構是否應先於語法；
- 符號如何成為可組合算子；

- 程式語言如何表達狀態、效果、權限、失敗與恢復；
- 意圖如何被編譯為多重可執行投影；
- 未來程式語言是否應直接描述世界差分。

任何真正長期運作的系統都會遇到：

- 錯誤；
- 相依失效；
- 資料漂移；
- 規則變更；
- 人類誤解；
- 未知事件；
- 資源不足；
- 攻擊；
- 組織調整；
- 技術退役。

所以，要求系統永不失敗並不現實。

但這不代表只能接受混亂。

系統仍可以被設計成：

- 失敗可被快速發現；
- 影響範圍可被限制；
- 當前狀態可被重建；
- 責任可被辨認；
- 外部效果可被驗證；

- 結果未知不被偽裝成成功；
- 可逆效果能回復；
- 不可逆效果能補償與說明；
- 核心責任能降級保存；
- 事故能轉化為下一版本的結構改善。

本文將可靠性重新定義為：

> 【Reliability
> ≠
> Absence of Failure】

而是：

> 【Reliability
> =
> Bounded Failure
> +
> Observable State
> +
> Responsible Recovery
> +
> Verified Restoration】

本文也將維護重新定義為：

> 【Maintenance
> ≠
> Patch Accumulation】

而是：

> 【Maintenance
> =
> Continuous Re-Understanding

一個能運作但不能解釋、不能恢復、不能交接、不能替換、不能退役的系統，只是暫時沒有倒下。

真正成熟的程式，不只知道成功時該做什麼，也知道：

失敗發生時，世界現在在哪裡？
哪些效果已經發生？
誰受到影響？
誰負責下一步？
哪些操作仍然安全？
哪些結果仍未知？
如何證明已恢復？
如何避免同一結構再次製造相同事故？

本文的最終命題是：

> 【失敗也是程式，】

> 【因為只有把失敗寫進世界模型，】

> 【程式才可能跨越單次執行，】

> 【成為能被人類與機器長期共同維持的存在結構。】

```
failure_state:
 failure_id: "payment-outcome-unknown-001"
 classification: "outcome-unknown"
 originating_action:
  action_id: "authorize-payment-778"
  idempotency_key: "order932-payment1"
 affected:
  entities:
   - "order-932"
   - "payment-attempt-778"
  actors:
   - "customer-17"
 known:
  - "request was accepted by local payment module"
  - "provider response timed out"
 unknown:
  - "whether provider authorized payment"
 prohibited_actions:
  - "blindly create new payment attempt"
 safe_next_actions:
```

```
    - "query provider using original idempotency identity"
    - "escalate to reconciliation after deadline"
  ownership:
    local: "payment-module"
    workflow: "order-module"
    user_resolution: "support-process"
observable_operation:
  trace_id: "trace-20260727-889"
  action_id: "deploy-release-42"
  actor: "release-manager-3"
  context:
    environment: "production"
    release: "v2.8.0"
    contract: "deploy-release-v3"
    rule_version: "deployment-policy-5"
  state:
    before: "approved"
    current: "verification-failed"
  events:
    - "DeploymentStarted"
    - "ProcessStarted"
    - "HealthCheckPassed"
    - "UserJourneyFailed"
  impact:
    users_affected: "estimated-12%"
    scope: "authentication-flow"
  ownership:
    local: "deployment-module"
    workflow: "release-process"
    incident_commander: "platform-oncall"
  recovery:
    safe_action: "shift-traffic-to-v2.7.4"
    evidence_required:
      - "authentication-journey-pass"
recovery_plan:
  responsibility: "order-processing"
  failure_modes:
    - id: "payment-provider-unavailable"
      recovery:
        primary: "degradation"
        mode: "accept-order-with-payment-pending"
      retry:
        allowed: true
        maximum: 8
        idempotency_required: true
```

```
    escalation_after: "30m"
  - id: "duplicate-charge"
    recovery:
      primary: "compensation"
      action: "refund-duplicate-charge"
    human_review_required: true
reconstruction:
  sources:
    - "event-history"
    - "daily-snapshot"
    - "provider-receipts"
  last_exercise: "2026-07-01"
  verified: true
incident_learning:
 incident_id: "INC-2026-071"
 structural_causes:
  - "outcome-unknown was not represented"
  - "tool retry contract lacked idempotency evidence"
 changes:
  problem_world:
   - "add outcome-unknown state"
  contract:
   - "require action_id and safe_to_retry"
  code:
   - "deduplicate by action identity"
  observability:
   - "trace tool request to external evidence"
  recovery:
   - "add reconciliation workflow"
  view:
   - "update agent tool-state diagram"
 verification:
  tests_added: true
  recovery_exercised: true
  owner_reviewed: true
```

- 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，《可視化作為外部結構記憶：多角色、多尺度的程式理解》，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，《Agent Runtime：能力規劃、工具調用與可恢復執行》，2026。

SECTION

一般理論背景

- Dijkstra, E. W., “The Humble Programmer,” 1972.
- Laprie, J.-C., “Dependable Computing and Fault Tolerance,” 1985.
- Avizienis, A. et al., “Basic Concepts and Taxonomy of Dependable and Secure Computing,” 2004.
- Gray, J. and Reuter, A., Transaction Processing, 1992.
- Lamport, L., Specifying Systems, 2002.
- Beyer, B. et al., Site Reliability Engineering, 2016.
- Nygard, M. T., Release It!, 2007.
- Kleppmann, M., Designing Data-Intensive Applications, 2017.

- Allspaw, J. and Robbins, J., Web Operations, 2010.
- Dekker, S., The Field Guide to Understanding Human Error, 2006.
- Woods, D. D., “Four Concepts for Resilience and the Implications for the Future of Resilience Engineering,” 2015.
- Kim, G. et al., The DevOps Handbook, 2016.
- Forsgren, N., Humble, J., and Kim, G., Accelerate, 2018.

SECTION

v0.1 — 2026-07-27

- 完成十八篇地基論文第十二篇。
- 完成第 1 冊《系統性程式設計》六篇封頂。
- 將失敗定義為程式問題世界的一部分。
- 建立失敗十元結構。
- 區分錯誤、失敗、事故與災難。
- 建立結構、語意、狀態、權限、效果、外部與目的七層驗證。
- 形式化可觀測性八元模型。
- 區分日誌、指標、追蹤、領域事件與證據。
- 建立失敗傳播、故障域與隔離方法。
- 建立重試、回滾、補償、降級與重建五類恢復。
- 提出數位回滾不等於世界回滾。
- 建立補償殘差與部分成功治理。
- 建立事故十階段生命週期與學習閉環。

- 將可維護性定義為可理解、可追蹤、可逆、可替換與歷史記憶。
- 區分技術債、語意債、責任債、觀測債與恢復債。
- 建立變更影響、版本遷移與正式退役流程。
- 加入付款、部署、AI Agent、醫療、研究平台與多團隊事故案例。
- 提出三十類失敗模式與十八項可證偽研究方向。
- 完成與第 2 冊《程式語言的本質》的銜接。