風格的時間相位：設計師思想、語言演化與社群治理如何分離

Temporal Phases of Design Style: Separating Designer Thought, Language Evolution, and Community Governance

v1.0 / 2026-07-30 / Neo.K / 公開版／第一部封頂論文

https://thisoneisneok.com/html/papers/pldst-005.html

英文名稱： Temporal Phases of Design Style: Separating Designer Thought, Language Evolution, and Community Governance

系列： Programming Language Designer Style Taxonomy（PLDST）

文件編號： PLDST-005

版本： v1.0

日期： 2026-07-30

作者： Neo.K

文件狀態： 公開版／第一部封頂論文

SECTION

摘要

程式語言設計師常被賦予一個看似穩定的終身標籤：

- Wirth 代表極簡；
- Stroustrup 代表相容性與零額外成本；
- Guido 代表可讀性；

- Matz 代表程式設計者愉悅；
- Rust 代表安全；
- Go 代表簡潔。

這些標籤可以作為初步索引，卻容易將數十年的思想、技術與制度演化壓縮為一個靜態人格。設計者可能改變觀點；語言可能在創始者離開後繼續演化；共同體可能把創始原則制度化，也可能以相同口號支持與原意不同的決策；向後相容可能把早期偶然選擇固定成永久表面；工具與生態還可能建立一套規格之外的「事實語言」。

Niklaus Wirth 從 Pascal 走向 Modula-2 時，曾為修正 Pascal 的缺陷而犧牲向上相容；到了 Oberon，他又依新硬體、系統實作經驗與對軟體膨脹的反思進一步縮減語言。Python 從 Guido van Rossum 的 BDFL 裁決轉為 Steering Council 治理。Rust 使用 Edition 機制，讓部分不向後相容的語法與解析變更採明確 opt-in，同時允許不同 Edition 的 crate 互操作。Go 透過 Go 1 Compatibility Promise 將穩定性變成長期制度承諾，並在 Go 1.21 進一步建立與語言版本相關的相容行為。C++ 的創始設計進入 WG21 後，演化受到多國標準化、既有程式與提案制度共同塑造。ECMAScript 新功能則經 TC39 的 Champion、階段成熟、實作與共識流程進入年度規格。[R1][R2][R3][R4][R5][R6]

本文提出 設計風格時間相位模型（Temporal Phase Model of Design Style, TPM-PL），將下列三條時間線分離：

> 【Σ_(designer)(t)
> ≠
> L_(language)(t)
> ≠
> G_(governance)(t)】

其中：

- Σ_(designer)(t)：設計師思想與直接決策風格；
- L_(language)(t)：語言規格、實作、工具與生態的實際狀態；
- G_(governance)(t)：誰能提出、批准、否決與維護變更。

本文進一步定義七個時間相位，以及八種風格流變：

- 保留；

- 放大；
- 弱化；
- 反轉；
- 化石化；
- 分岔；
- 制度化；
- 回顧性重構。

  核心命題為：

> 【早期設計原則的延續
> not⇒
> 創始者仍在做決定】

以及：

> 【語言今天呈現某種風格
> not⇒
> 創始者在所有時期都持有相同觀點】

PLDST 對一位設計師的正確輸出，不應是一個跨越終身的固定分數，而應是一組具有時間、決策範圍、治理背景、證據與信心的風格相位：

> Σ_d
> =
> {
> Σ_(d,t□),
> Σ_(d,t□),
> …,
> Σ_(d,t□)
> }

關鍵詞： 程式語言設計、時間相位、語言演化、設計師思想、社群治理、向後相容、Edition、標準化、PLDST

SECTION

一、靜態標籤的便利與危險

靜態標籤很方便：

某人＝極簡
某語言＝安全
某共同體＝保守
某委員會＝功能擴張

但它會隱藏四個問題：

- 設計師在不同年代可能面對不同問題；
- 同一原則在不同規模下可能導致相反決策；
- 語言進入制度後，決策者已經改變；
- 今日特徵可能只是歷史相容性，而不是當代偏好。

SECTION

二、個人思想不等於語言狀態

設計者可以：

- 後悔早期選擇；
- 提倡新的方向；
- 不再具有正式權力；
- 離開專案；
- 轉向另一種語言；
- 以回顧方式重新解釋過去。

但語言中的既有程式與相容承諾不會因此消失。

因此：

> ΔΣ_(designer)
> not⇒
> Δ L_(language)

設計者改變思想，不代表語言能同步改變。

SECTION

三、語言狀態不等於治理偏好

某個歷史特徵被保留，可能是因為：

- 委員會仍然喜歡它；
- 不能破壞既有程式；
- 缺乏遷移工具；
- 多實作無法同步；
- 生態成本過高；
- 沒有人願意投入修改；
- 正在等待下一個 Edition 或 major version。

  因此：

> FeaturePresent(t)
> not⇒
> FeaturePreferred(t)

SECTION

四、治理制度不等於社群結果

正式制度可能批准某功能，但生態：

- 不採用；
- 只在特定領域採用；
- 建立 Linter 禁止；
- 以 Framework 包裝；
- 形成不同慣例。

  反之，一個未進入核心的模式也可能因程式庫與工具而成為事實標準。

SECTION

五、設計師思想軌跡

定義：

> Σ_d(t)
> =
> (
> Problems,
> Values,
> Tradeoffs,
> RejectedChoices,
> Responsibility,
> Complexity,
> GovernancePreference
>)⊠

它只涵蓋能合理歸因於設計者本人的材料：

- 原始文章；
- 演講；
- 訪談；

- 直接設計；
- 親自拒絕；
- 具有裁決權時的決定。

SECTION

六、語言技術軌跡

定義：

> L(t)
> =
> (
> Syntax,
> Semantics,
> TypeSystem,
> Runtime,
> Libraries,
> Tools,
> Implementations,
> Compatibility
>)⊠

它回答：

> 某個時間點，使用者實際面對的是什麼語言？

SECTION

七、治理權力軌跡

定義：

```
G(t)
=
(
Proposal,
Review,
Approval,
Veto,
Release,
Maintenance,
Representation
)⊠
```

它回答：

- 誰能提案？
- 誰能要求修改？
- 誰能接受？
- 誰能阻止？
- 誰控制 Release？
- 誰維護規格與實作？

SECTION

八、生態使用軌跡

雖然核心模型以三線為主，完整研究還應保存：

```
E(t)
=
(
Adoption,
Idioms,
Libraries,
```

因為語言的有效風格常由生態而非規格單獨決定。

SECTION

九、觀察到的風格

某時間點外界感受到的語言風格可表示為：

> O_L(t)
> =
> α⊠Σ_d(t)
> +
> β⊠L(t)
> +
> γ⊠G(t)
> +
> δ⊠E(t)
> +
> ε⊠X(t)

其中 X(t) 是硬體、平台、公司與時代限制。

各權重會隨時間改變。創始期通常 α⊠ 高；制度成熟後，γ⊠ 與 δ⊠ 可能上升。

SECTION

十、相位一：問題孕育期

特徵：

- 尚無正式語言；
- 設計者批判既有工具；
- 問題 framing 尚在形成；
- 設計替代方案多；

- 後來核心原則可能尚未明說。

主要資料：

- 早期備忘錄；
- 前代專案；
- 實驗；
- 組織問題；
- 個人回憶。

風險：

- 以後來成功敘事回寫早期動機。

SECTION

十一、相位二：創始凝結期

特徵：

- 核心語法與語義形成；
- 第一個實作；
- 小型團隊；
- 個人裁決權高；
- 變更成本低；
- 可接受大幅重寫。

此時最適合研究個人設計風格，但仍需列出共同設計者與實作者。

SECTION

十二、相位三：公開擴張期

特徵：

- 外部使用者加入；
- 新領域採用；
- 程式庫與工具增加；
- Bug 與邊界案例暴露；
- 原始原則接受壓力測試。

此時可能發生：

- 原則被放大；
- 原則被修正；
- 功能快速加入；
- 核心團隊擴張；
- 實作者權重上升。

SECTION

十三、相位四：制度化期

特徵：

- PEP／RFC／Proposal；
- Core Team；
- Council；
- Committee；
- 正式 Release；
- Stability process；

- Feature gate。

  個人偏好開始被轉譯成可重複制度。

SECTION

十四、相位五：相容性鎖定期

當程式、生態與組織大量依賴既有行為後：

> Freedom_(design)(t)downarrow

> Cost_(change)(t)uparrow

此時保留舊特徵不一定表示贊同，而可能是：

- 相容承諾；
- 遷移成本；
- 多實作；
- 教材；
- 信任。

SECTION

十五、相位六：後創始者期

創始者不再具有最終裁決權，可能：

- 退出；
- 退休；
- 轉為顧問；
- 只保留象徵性影響；

- 與現行制度意見不同。

  研究此相位時應主要分析制度與共同體，而不是持續替創始者記帳。

SECTION

十六、相位七：反身重構期

成熟共同體開始主動管理自己的歷史：

- Edition；
- Migration；
- Deprecation；
- Compatibility mode；
- Retrospective；
- Governance reform；
- Style reform；
- Tool-assisted rewrite。

  語言不再只是演化，而是建立「如何演化」的二階機制。

SECTION

十七、保留 Conservation

創始原則在後續制度中持續被支持。

例如：

- 小型核心；
- 可讀性；
- 向後相容；

- 零額外成本。

保留需要證據：

- 後期正式文件；
- 重複決策；
- 反例處理；
- 制度化規則。

SECTION

十八、放大 Amplification

共同體把原始傾向推得更遠。

例如：

- 原始設計只偏好安全，後來加入更完整的 Lint、Proof 或 Supply-chain policy；
- 原始語言重視可讀，後來 Formatter 與 Style guide 將其制度化。

放大後的結果不能完全回寫成創始者本人主張。

SECTION

十九、弱化 Attenuation

原始原則仍存在，但因生態與相容性而降低優先級。

例如：

- 小型語言核心在大型程式庫與 Framework 下不再等同簡單使用經驗；
- 自由表達在大型組織 Style guide 中受到限制。

SECTION

二十、反轉 Inversion

後來制度採取與創始期相反的方向。

可能原因：

- 規模改變；
- 安全事故；
- 新硬體；
- 新使用者；
- 公司策略；
- 創始者離開。

  反轉不應被勉強解釋成「其實仍符合原始精神」，除非有明確證據。

SECTION

二十一、化石化 Fossilization

一個早期選擇因相容性被永久保存，即使現代設計者不再偏好。

化石特徵：

存在
但不推薦
不能移除
工具警告
新程式避免
舊程式依賴

SECTION

二十二、分岔 Bifurcation

同一語言或共同體形成兩條路線：

- 核心與生態；
- Stable 與 Experimental；

- Strict 與 Dynamic；
- Edition A 與 Edition B；
- 標準與方言；
- 原專案與 Fork。

此時不應輸出單一風格。

SECTION

二十三、制度化 Institutionalization

原本依賴個人判斷的原則，被轉化為：

- Proposal template；
- Compatibility policy；
- Feature gate；
- Review team；
- Test requirement；
- Release cadence；
- Council charter。

制度化是原則的程序化，不代表程序能完全保存原始價值。

SECTION

二十四、回顧性重構 Retrospective Reconstruction

後來的成功、失敗與身份認同會重新敘述早期歷史。

常見形式：

- 將偶然限制寫成有意原則；

- 將多方共同設計寫成個人願景；
- 將後期價值回寫到第一版；
- 將被淘汰路線從歷史中刪除。

PLDST 必須分開：

contemporary evidence
later recollection
institutional narrative
current interpretation

SECTION

二十五、不能只按版本號分期

版本號未必代表權力或思想轉換。

真正相位邊界可能是：

- 核心成員改變；
- 創始者離開；
- 第一個正式標準；
- 相容承諾；
- RFC 制度；
- 多實作；
- 重大事故；
- 公司收購；
- Edition；
- Governance reform。

SECTION

二十六、相位轉換指標

定義事件集合：

> B=
> {
> b_(authority),
> b_(compatibility),
> b_(implementation),
> b_(community),
> b_(specification),
> b_(ecosystem)
> }

若多項指標同時改變，可判定相位轉換。

SECTION

二十七、權力轉換

當下列主體改變時：

- 最終裁決者；
- Release owner；
- Spec editor；
- Core team；
- Committee；
- Council；

  治理相位可能已變。

SECTION

二十八、相容性轉換

當專案首次承諾：

- 1.x compatibility；
- Semantic Versioning；
- Stable ABI；
- Edition interoperability；
- Deprecation period；

  設計自由的時間結構已改變。

SECTION

二十九、實作轉換

第一個實作與多個獨立實作之間，語言規格的地位會改變。

單一實作時：

> Implementation≈ Specification

多實作時必須更清楚區分：

> Specification
> ≠
> Implementation

SECTION

三十、Wirth：相位不是單純維持極簡

Wirth 的語言歷程至少包含：

ALGOL W
→ Pascal
→ Modula／Modula-2
→ Oberon

→ Oberon-07

ETH 的專案回顧指出，他從 Pascal 轉向 Modula-2 時，願意犧牲向上相容以避免 Pascal 的缺陷；其後又因大型系統、模組、併行與硬體經驗持續修正設計。[R1]

在 Modula-2 到 Oberon 的回顧中，Wirth 也具體說明硬體變化如何改變程式碼密度、指令選擇與編譯器設計考量。[R1]

因此他的風格不是靜態的「永遠刪功能」，而更接近：

持續尋找最小且足以支撐當前系統的機制
願意以新語言修正舊語言
讓硬體與完整系統實作反饋語言設計
這是一種相位化極簡，而非固定功能數量。

SECTION

三十一、Python：思想延續與治理權轉移

Python 的可讀性、實用性與開放整合原則在多個時期仍可觀察，但治理已發生根本轉換。

PEP 13 記錄：

- Guido van Rossum 啟動 Python；
- 自創始至 2018 年 7 月擔任 BDFL；
- 現代治理以五人 Steering Council 為核心；
- Council 傾向透過程序與委任而非頻繁直接裁決。[R2]

  因此：

> StyleContinuity
> ∧
> AuthorityDiscontinuity

可以同時成立。

今日 Python 仍可能延續早期風格，但新決策的正式歸因需放在 PEP、Delegate、Core Team 與 Steering Council。

SECTION

三十二、Rust Edition：將斷裂限制在明確相位

Rust Edition Guide 說明：

- 不向後相容變更會放入下一 Edition；
- Edition 採 opt-in；
- 既有 crate 不會自動改變；
- 不同 Edition crate 可以互相連結；
- 新 Compiler 支援其發布前存在的 Edition。[R3]

  這建立：

> LanguageVersion
> ≠
> Edition

Edition 不是傳統 major-version 全面斷裂，而是局部管理解析、關鍵字與部分語言變更的時間邊界。Rust Book 將 Edition 差異概括為編譯器初始解析方式的差異；本文採此官方相容模型，但不將它擴張為「所有 Rust 語義演化都只涉及 Parser」的普遍斷言。

其制度風格為：

允許修正
但限制跨代破壞
提供遷移工具
維持生態互操作
Rust 2024 的相容 Lint 與 cargo fix --edition 進一步把相位轉換工具化。[R7]

SECTION

三十三、Go：相容承諾形成演化重力

Go 1 Compatibility Promise 的基本期待是：為 Go 1 撰寫的程式應在後續 Go 1 版本中持續編譯與正確運行，但官方文件也列出安全、未指定行為、工具與 Bug 修正等邊界。[R4]

這使 Go 語言設計進入相容性鎖定期。

後續語言變更需通過提案與語言審查；Go Team 對 Go 2 的討論也明確傾向拒絕尚未充分發展、收益不足以支付永久成本的功能。[R8]

Go 1.21 又透過版本與 GODEBUG 相容行為，強化新工具鏈對舊版本語義的支援。[R4]

因此 Go 的「簡潔」在後期不只是創始者審美，而是：

- 相容承諾；
- 高語言變更門檻；
- 提案成本；
- 實作與經驗要求；

  共同維持的制度結果。

SECTION

三十四、C++：創始原則與標準化疊加

Stroustrup 的原始設計持續影響：

- C 相容；
- 硬體映射；
- 一般化抽象；
- 零額外成本。

  但 C++ 的現代演化由 WG21 多國專家、Working Groups、Papers、實作者與既有程式共同塑造。WG21 成立於 1990–1991 年，現代語言已經歷多次標準版本。[R5]

  因此 C++ 應至少分為：

C with Classes 創始期
早期 C++ 擴張期
ARM／標準化前期
C++98 制度定型
C++11 現代化
後續週期性標準演化

「C++ 風格」在後期是個人原則、委員會制度與相容性歷史的疊加，而非單一意志。

SECTION

三十五、ECMAScript：從快速創始到年度制度演化

JavaScript 的早期形成與今日 ECMAScript 規格制度具有完全不同的時間條件。

TC39 Process 要求 Stage 1 以上提案由委員會擁有，並透過逐階成熟處理問題、規格文字、實作、測試與共識；完成 Stage 4 的提案會進入後續年度規格。[R6]

因此今日 ECMAScript 功能應描述為：

某 Champion 推動
TC39 共同體審查
引擎實作者提供回饋
Test262 與規格編輯驗證
共識後進入規格
它不是創始者風格的直接延伸。

SECTION

三十六、何謂設計者真正改變觀點

不能只因某語言新增功能就說設計者改變思想。

至少需要：

- 本人後期明確陳述；
- 由本人直接控制的後期決策；
- 多項決策呈現一致方向；
- 排除相容性與制度強迫；
- 與早期材料比較。

SECTION

三十七、觀點修正與問題改變

設計者可能沒有改變價值，而是問題尺度改變。

例如：

- 個人程式轉為大型團隊；
- 單機轉為分散式；
- 研究原型轉為生產平台；
- 單一實作轉為標準；
- 小型生態轉為全球生態。

  相同原則在新尺度下可能產生不同機制。

SECTION

三十八、工具改變可行選項

新技術可能讓過去昂貴的設計變得可行：

- 增量編譯；
- IDE；
- SMT；
- JIT；
- GC；
- LSP；
- 自動遷移；
- AI 輔助。

  因此後期決策不同，不必然表示價值反轉。

SECTION

三十九、沉默不等於同意

創始者沒有公開反對某項後期功能，不代表：

- 贊成；
- 參與；
- 應被歸因；
- 功能符合原始風格。

PLDST 只依可確認決策。

SECTION

四十、相位記錄

phase_id
designer_or_body
language
start_date
end_date
trigger_events
decision_authority
active_constraints
core_values
representative_decisions
rejected_decisions
compatibility_regime
implementation_regime
ecosystem_scale
evidence
confidence

SECTION

四十一、決策時間標記

每筆決策 q 保存：

$$
\begin{aligned}
&q=\\
&(\\
&t_{proposal},\\
&t_{decision},\\
&t_{implementation},\\
&t_{stabilization},\\
&t_{adoption},\\
&t_{deprecation}\\
&)
\end{aligned}
$$

同一功能的提出、接受、實作與廣泛採用可能相隔多年。

SECTION

四十二、風格相位向量

$$
\begin{aligned}
&\Sigma_{a,p}\\
&=\\
&(\\
&V,\\
&C,\\
&R,\\
&E,\\
&G,\\
&X\\
&)
\end{aligned}
$$

其中：

- V：價值優先序；
- C：複雜度配置；
- R：責任配置；

- E：演化偏好；
- G：治理偏好；
- X：外部限制。

SECTION

四十三、相位差

比較兩相位：

> ΔΣ_(p⊠→ p⊠)
> =
> Σ_(p⊠)-Σ_(p⊠)

輸出不只顯示數值，還要分類：

保留
放大
弱化
反轉
化石化
分岔
制度化
不可判定

SECTION

四十四、不得使用終身平均

錯誤格式：

Guido 的治理集中度：8/10
因為 BDFL 與 Steering Council 時期完全不同。

正確格式：

創始／BDFL 期：最終裁決高度集中
後 BDFL 期：個人創始影響仍在，但正式權力轉移至 Council

SECTION

四十五、人物與語言同時分期

每個人物個案至少建立兩張表：

設計師思想相位

早期問題
核心形成
公開回顧
後期修正
退出後觀點

語言演化相位

原型
首次發布
穩定版本
治理制度
相容鎖定
後創始者
兩張表不得強制一一對齊。

SECTION

四十六、制度變更必須單獨成段

若出現：

- BDFL 退出；
- Council；
- RFC；
- Edition；
- ISO；
- Stage process；

- Compatibility promise；

  必須說明它如何改變決策權重。

SECTION

四十七、歷史特徵分類

每個現存特徵標記為：

active preference
compatibility fossil
institutional compromise
ecosystem convention
implementation constraint
deprecated legacy

SECTION

四十八、同時期比較

比較設計者應優先選擇相近時期與規模：

- 1970 年代研究語言；
- 1990 年代商業物件語言；
- 2010 年代系統語言；
- 成熟標準化語言。

  避免以現代工具要求早期設計。

SECTION

四十九、跨時期比較

跨時期比較應問：

- 哪些問題已被工具消除？
- 哪些相容成本不同？

- 硬體有何變化？
- 使用者規模有何變化？
- 治理制度有何變化？

SECTION

五十、同一設計者跨作品比較

這是辨識深層風格的重要方法。

例如 Wirth：

若特徵改變但取捨規則重複
→ 深層風格可能保留
若取捨規則也改變：

→ 思想相位可能轉換

SECTION

五十一、輸入

designer
language
time_range
decision
version
governance_event
source_documents

SECTION

五十二、處理管線

重新網路搜尋
→ 時間線建立
→ 人物／語言／治理三線分離
→ 相位邊界偵測
→ 決策歸因
→ 風格流變分類
→ 相容性化石檢查
→ 回顧性重構檢查

→ 反例搜尋
→ 第二輪事實校對
→ 報告

SECTION

五十三、相位邊界演算法雛形

```
for each event:
  if authority changed:
    add governance boundary
  if compatibility regime changed:
    add compatibility boundary
  if implementation count changed materially:
    add implementation boundary
  if founder exited:
    add post-founder boundary
  if edition/major migration introduced:
    add reflexive-evolution boundary
merge nearby boundaries
require source evidence
assign confidence
```

SECTION

五十四、輸出 JSON 雛形

```
{
 "subject": "Python governance style",
 "phases": [
  {
   "name": "BDFL phase",
   "authority": ["Guido van Rossum"],
   "style": {
    "decision_centralization": "high",
    "delegation": "present"
   }
  },
  {
   "name": "Steering Council phase",
   "authority": ["elected steering council"],
   "style": {
    "decision_centralization": "collective",
    "delegation": "high",
    "formal_process": "high"
```

```
    }
   }
  ],
  "continuities": ["PEP-based public design"],
  "discontinuities": ["final authority"],
  "founder_attribution_limit": "post-2018 decisions require separate attribution"
}
```


SECTION

五十五、SKILL 禁止事項

不得：

- 用目前語言狀態代表創始期；
- 用創始期思想代表現代制度；
- 用版本號自動判斷思想轉變；
- 把相容保留寫成現代偏好；
- 把創始者沉默寫成支持；
- 把制度口號直接當實際權力；
- 把回顧文章當唯一歷史來源；
- 用一個終身分數覆蓋多相位；
- 忽略硬體、公司與實作變化。

SECTION

五十六、PLDST-001：決策風格

第一篇建立：

> Σ_(d,t)
> =
> (
> Context,
> Problem,

本篇正式將 t 從標記提升為核心分析維度。

SECTION

五十七、PLDST-002：複雜度配置

複雜度配置會隨相位改變：

- 創始期願意破壞相容；
- 成熟期把成本移入治理；
- Edition 把成本分期；
- 工具把遷移成本攤銷。

因此：

> C(d,t)
> ≠
> C(d)

SECTION

五十八、PLDST-003：責任配置

責任也會隨制度轉移：

創始者裁決
→ Core team
→ Council／Committee
→ Tool／Migration／Compatibility policy
所以：

> Resp(a,k,t)

必須帶時間索引。

SECTION

五十九、PLDST-004：多主體歸因

上一文建立：

> Credit
> ≠
> Causality
> ≠
> Authority
> ≠
> Accountability
> ≠
> Maintenance

本篇補充：這五種歸因都會隨時間改變。

SECTION

六十、第一部統一模型

PLDST 第一部的完整分析單位為：

> 【mathfrakD
> =
> (
> Actor,
> Decision,
> Context,
> Time,
> Style,
> Complexity,
> Responsibility,
> Attribution,
> Governance,

這將成為第二部風格原型與第三部人物個案的共同方法。

SECTION

六十一、相位邊界不是自然唯一

歷史是連續的，研究者畫出的相位是分析工具。

不同研究問題可能需要不同分期。

SECTION

六十二、文件日期不等於思想形成日期

文章發布時間只證明觀點最晚在該時可被確認，不保證觀點在那天才形成。

SECTION

六十三、制度規則不等於實際行為

正式文件可能與實際權力、公司資源與非正式影響不同。

需要結合：

- 決議；
- 實作；
- 會議；
- Release；
- 參與者證詞。

SECTION

六十四、相容性化石難以判定

一個舊特徵也可能仍受部分使用者喜愛。除非有：

- 正式棄用；
- 明確反對；
- 替代機制；
- 新程式不建議；

  否則不應武斷稱為化石。

SECTION

六十五、創始者思想可能多線並存

設計者可以同時：

- 重視簡潔；
- 接受某些複雜機制；
- 反對某種相容；
- 保護另一種相容。

  相位分析不能把人變成單一方向向量。

SECTION

六十六、Wirth 的相容性與硬體回饋

已核對 ETH 官方專案回顧與〈The History of Modula-2 and Oberon〉：

- Wirth 從 Pascal 轉向 Modula-2 時，確實選擇犧牲 upward compatibility 以避免 Pascal 缺陷；
- Oberon 的設計與完整系統實作相關；
- 硬體指令與記憶體特性變化確實改變了編譯器與程式碼密度考量。

  本文沒有把他的思想簡化成永遠拒絕所有新功能。

SECTION

六十七、Python 治理

已核對 PEP 13：

- Guido 至 2018 年 7 月為 BDFL；
- 當前治理以五人 Steering Council 為核心；
- Council 具有廣泛權力，但傾向建立流程與委任。

  因此本文將風格延續與權力斷裂分開。

SECTION

六十八、Rust Edition

已核對 Rust Edition Guide、Rust Book 與 Edition RFC：

- Edition 是 opt-in；
- 不向後相容變更可放入下一 Edition；
- 不同 Edition crate 可互操作；
- Compiler 支援其發布前既有 Edition；
- Rust 2024 具有遷移 Lint 與 cargo fix --edition 支援。

  本文沒有把 Edition 說成完全獨立語言或傳統 major version。

SECTION

六十九、Go 相容性

已核對 Go 1 Compatibility Promise 與 Go 官方相容性文章：

- Go 1 對多數既有程式提供長期相容期待；

- 官方明確列出有限例外；
- Go 1.21 加強了工具鏈、語言版本與 GODEBUG 的相容模型。

本文沒有將承諾誇張為任何程式永不破壞的絕對保證。

SECTION

七十、C++ WG21

已核對 WG21 官方頁面：

- WG21 於 1990–1991 年成立；
- 是 C++ 的國際標準化 Working Group；
- 現代演化涉及 Papers、分組、專家與標準程序。

本文保留 Stroustrup 創始原則的持續影響，但未將全部後期標準決策歸於個人。

SECTION

七十一、TC39

已核對 TC39 Process 與 ECMAScript 規格頁：

- Stage 1 以上提案由 Committee 擁有；
- 提案採分階段成熟；
- Stage 4 完成後進入後續年度規格；
- 現行規格頁會整合年度快照與已完成提案。

本文沒有把現代 ECMAScript 功能回寫成 JavaScript 創始者的直接決策。

程式語言設計風格不是一張永久貼在人物身上的標籤。它是在特定時期、問題、權力、實作與相容條件下，透過一連串決策呈現的模式。

本文將三條時間線正式分離：

> 【Σ_(designer)(t)
> ≠
> L_(language)(t)
> ≠
> G_(governance)(t)】

並加入生態與外部條件：

> O_L(t)
> =
> α⊠Σ_d(t)
> +
> β⊠L(t)
> +
> γ⊠G(t)
> +
> δ⊠E(t)
> +
> ε⊠X(t)

語言風格可能經歷：

> 【保留
> +
> 放大
> +
> 弱化
> +
> 反轉
> +
> 化石化
> +
> 分岔
> +
> 制度化
> +

因此，PLDST 不再輸出：

某設計師一生都是某一派。

而輸出：

> 在某一時間相位、某組可直接歸因的決策與某種治理結構下，該設計者呈現何種優先序；後續語言與制度如何保留、修正、放大或反轉這些選擇。

第一部至此建立了完整方法論：

- 從語言特徵轉向設計決策；
- 分析複雜度被如何配置；
- 分析錯誤與控制責任由誰承擔；
- 分離創始者、共同體與制度；
- 以時間相位防止靜態人格化。

其統一分析物件為：

> 【mathfrakD
> =
> (
> Actor,
> Decision,
> Context,
> Time,
> Style,
> Complexity,
> Responsibility,
> Attribution,
> Governance,
> Evidence
>)】

後續第二部將不再只列出「極簡、實用、安全、表達力」等抽象原型，而會用這套方法分析每一種風格在什麼情境形成、把複雜度與責任放到哪裡、如何隨時間演化，以及在何種條件下從優勢轉為負債。

相位　主要特徵

問題孕育期　問題 framing、前代批判、實驗

創始凝結期　核心設計、第一實作、個人權重高

公開擴張期　使用者、工具、程式庫與壓力測試

制度化期　PEP／RFC／Council／Committee

相容性鎖定期　變更成本、既有程式與承諾

後創始者期　正式權力轉移、制度與共同體主導

反身重構期　Edition、遷移、治理改革、歷史修正

Conservation
Amplification
Attenuation
Inversion
Fossilization
Bifurcation
Institutionalization
Retrospective Reconstruction

[R1] Niklaus Wirth, “The History of Modula-2 and Oberon,” revised manuscript, 2006; ETH Zürich project history and Oberon language reports.

— Pascal、Modula-2、Oberon 的設計演化、相容選擇、系統實作與硬體回饋。

[R2] Python Enhancement Proposals, “PEP 13 – Python Language Governance,” and related PEP 8000-series governance records.

— BDFL 時期、2018 年治理轉換與 Steering Council。

[R3] Rust Project, The Rust Edition Guide, The Rust Programming Language, RFC 2052, RFC 3085, and RFC 3501.

— Edition 的 opt-in、跨 Edition 互操作、遷移與週期制度。

[R4] Go Project, “Go 1 and the Future of Go Programs,” “Backward Compatibility, Go 1.21, and Go 2,” and Go language proposal documentation.

— Go 1 相容承諾、例外、語言版本與 GODEBUG 相容機制。

[R5] ISO/IEC JTC1/SC22/WG21 official committee pages and Bjarne Stroustrup, The Design and Evolution of C++.

— C++ 創始設計、標準化與多階段演化。

[R6] Ecma TC39, “The TC39 Process,” and the ECMAScript specification repository.

— Champion、提案階段、委員會所有、實作與年度規格。

[R7] Rust Edition Guide, Rust 2024 migration chapters and compatibility lint documentation.

— cargo fix --edition 與 Edition migration。

[R8] Go Project, “Go 2, here we come!,” “Toward Go 2,” “Proposals for Go 1.15,” and Contribution Guide.

— 語言提案門檻、功能成本、經驗與受控演化。

[T-F] Founder phase
[T-C] Core formation
[T-E] Expansion
[T-I] Institutionalization
[T-L] Compatibility lock-in
[T-P] Post-founder
[T-R] Reflexive reconstruction
[S-C] Conservation
[S-A] Amplification
[S-W] Attenuation
[S-I] Inversion
[S-F] Fossilization
[S-B] Bifurcation
[S-N] Institutionalization
[S-R] Retrospective reconstruction