個人設計者、仁慈獨裁者與 RFC 制度：語言治理風格比較

Individual Designers, Benevolent Dictators, and RFC Systems: A Comparative Study of Programming Language Governance Styles

v1.0 / 2026-07-30 / Neo.K / 公開版／第四部跨設計師比較封頂篇

https://thisoneisneok.com/html/papers/pldst-026.html

英文名稱： Individual Designers, Benevolent Dictators, and RFC Systems: A Comparative Study of Programming Language Governance Styles

系列： Programming Language Designer Style Taxonomy（PLDST）

文件編號： PLDST-026

版本： v1.0

日期： 2026-07-30

作者： Neo.K

文件狀態： 公開版／第四部跨設計師比較封頂篇

SECTION

摘要

程式語言通常被描述為語法、型別系統、執行模型、標準函式庫與工具鏈的集合。然而，任何能持續演化的語言還包含另一個不可省略的構造：

> 誰有權提出改變、判定改變、實作改變、發布改變，並宣告改變後的系統仍是同一門語言？
>
> 個人設計者、仁慈獨裁者、核心團隊、RFC 制度、Steering Council、企業主導專案與標準委員會，不只是不同的行政安排。它們會直接塑造：

- 語言功能加入速度；
- 概念一致性；
- 向後相容；
- 少數意見的存留；
- 提案文件品質；
- 實作與規格的距離；
- 生態權力分布；
- 接班與危機處理；
- 長期複雜度。

  本文把語言治理形式化為：

> cal-G(L)
> =
> (
> A,
> P,
> D,
> I,
> R,
> C,
> S,
> T,
> E
>)

其中：

- A：Agenda setting，議程設定權；
- P：Proposal access，提案准入；

- D：Decision authority，最終裁決權；
- I：Implementation control，實作控制；
- R：Release authority，發布權；
- C：Compatibility obligation，相容義務；
- S：Succession，接班機制；
- T：Transparency，可追蹤性；
- E：Exit／Fork，退出與分叉能力。

本文主要比較五種原型：

> 【G□ : 個人設計者工作室;
> G□ : BDFL／品味中心制;
> G□ : RFC＋領域團隊聯邦制;
> G□ : Steering Group／企業—社群混合制;
> G□ : 標準委員會／國家代表共識制】

核心案例包括：

- Niklaus Wirth 與 Oberon：設計、實作、文件與系統共同收斂於小型設計核心；
- Guido van Rossum 與 Python：由 BDFL 最終裁決，經 PEP 制度化後轉向五人 Steering Council；
- Rust：重大變更以 RFC 公開形成紀錄，由領域團隊承擔技術責任，Leadership Council 處理跨團隊協調；
- Swift：公開 Evolution 提案與 Language Steering Group 裁決並存，同時保留 Apple 工程資源與產品路線的結構性影響；
- C++：以 WG21、國家代表、Study Group、Evolution Working Group、書面提案與共識程序形成標準；
- Go：以小型設計團隊、公開 Proposal process 與高度保守的語言變更准入形成混合治理。

本文的主要結論是：

> 語言治理不是把設計權從個人轉移給「社群」便告完成，而是把品味、證據、責任、否決、實作與接班重新配置。

RFC 也不等於直接民主。公開提案只解決「意見如何進入紀錄」，並不自動回答：

- 誰負責整體一致性；
- 誰有權結束無限討論；
- 誰為錯誤決策承擔維護成本；
- 誰能拒絕受歡迎但有害的功能；
- 誰控制參考實作與發布管線。

本文提出語言治理的核心公式：

> 【LegitimateEvolution
> =
> OpenReasoning
> +
> BoundedAuthority
> +
> ImplementationResponsibility
> +
> Succession
> +
> HistoricalRecord】

PLDST 因而不只分類設計者「偏好什麼功能」，還必須分類他們「如何讓偏好成為制度」，以及語言在離開創始者後如何保存、轉譯或失去原始風格。

關鍵詞： 語言治理、個人設計者、BDFL、PEP、RFC、Steering Council、Rust、Swift Evolution、WG21、C++ 標準化、Go Proposal、接班、PLDST

SECTION

一、語言從來不只是規格

一門現代程式語言至少包含：

> L
> =
> Syntax
> +
> Semantics
> +
> Implementation
> +
> Library
> +
> Tooling
> +
> Governance

若移除 Governance，便無法回答：

- 下一版由誰定義；
- 規格歧義由誰解釋；
- Bug fix 是否構成語義改變；
- 哪個實作具有事實權威；
- 哪些舊程式必須繼續工作；
- 社群分歧時誰能裁決。

SECTION

二、治理會進入語義

假設規格中存在歧義：

> Meaning(x)∈{m□,m□}

最終語義可能由以下任一者決定：

- 創始者解釋；
- Reference implementation；
- 標準委員會缺陷報告；
- RFC；
- Steering Council 決議；
- 多個實作的既成慣例；
- 社群分叉。

  所以：

> SemanticAuthority
> ⊆
> GovernanceAuthority

SECTION

三、語言設計具有不可逆性

功能一旦發布，會形成：

- Source compatibility；
- Binary compatibility；
- 教材；
- Framework；
- 套件；
- 使用者預期；
- 人才市場；

- 合規與標準依賴。

  因此：

> Cost(AddFeature)
> <
> Cost(RemoveFeature)

治理的主要工作往往不是批准創意，而是管理不可逆性。

SECTION

四、治理也是複雜度閥門

定義功能流入率：

> λ_F
> =
> (Accepted Features)/(Time)

若：

> λ_F
> >
> λ_R

其中 λ_R 為整理、移除、統一與工具吸收複雜度的能力，則語言複雜度會累積。

治理制度實際上控制：

> λ_F-λ_R

SECTION

五、設計品質不等於治理品質

卓越設計者可能：

- 無法接班；
- 缺乏文件；
- 無法擴展協作者；
- 過度依賴直覺；
- 讓反對理由消失；
- 造成 Bus factor。

優秀治理也可能：

- 產生折衷式功能；
- 稀釋整體品味；
- 過度程序化；
- 讓提案成本高到排除新參與者；
- 無人對整體負責。

SECTION

六、治理的最低問題集

任何語言治理都必須回答：

誰可以提出？
誰決定是否進入議程？
誰主持討論？
誰定義證據標準？
誰決定？
誰實作？
誰測試？
誰發布？
誰承擔相容成本？
誰解釋爭議？
誰能撤回決策？

SECTION

七、權力不是單一變量

語言權力至少分為：

> cal-P
> =
> (
> P_(agenda),
> P_(proposal),
> P_(review),
> P_(decision),
> P_(implementation),
> P_(release),
> P_(interpretation),
> P_(appointment)
>)

「任何人可以提交 RFC」只表示 P_(proposal) 較開放。

它不表示其他權力已平均分配。

SECTION

八、責任向量

定義：

> cal-R
> =
> (
> R_(coherence),
> R_(compatibility),
> R_(security),
> R_(implementation),

合法治理需要權力與責任大致對齊：

> P⊠
> ≈
> R⊠

若權力高而責任低，容易產生任意裁決。

若責任高而權力低，容易產生維護者耗竭。

SECTION

九、三層合法性

本文區分：

程序合法性

決策是否依公開且穩定的程序完成？

專業合法性

裁決者是否理解語義、實作、相容與生態代價？

結果合法性

決策是否實際提高語言品質與可持續性？

可表示為：

> Legitimacy
> =
> L_(procedure)
> +
> L_(expertise)
> +
> L_(outcome)

三者不能互相完全替代。

SECTION

十、治理吞吐量

> Throughput
> =
> fracDecisionQuality× AcceptedDecisions
> Time+CoordinationCost

個人設計者通常具有：

- 低協調成本；
- 高一致性；
- 有限吞吐量；
- 高單點風險。

大型委員會通常具有：

- 高證據量；
- 高代表性；
- 高協調成本；
- 可能較慢但較能承擔標準化責任。

SECTION

十一、治理延遲

> Latency
> =
> T_(proposal)
> +

快速裁決不等於快速落地。

慢速討論也不一定意味低品質。

SECTION

十二、治理記憶

制度是否保存：

- 原始提案；
- 替代方案；
- 反對意見；
- 決策理由；
- 實作追蹤；
- 後續修正；
- 被拒提案。

  定義：

> M_G
> =
> ProposalHistory
> +
> DecisionRationale
> +
> ImplementationTrace
> +
> RevisionRecord

PEP、RFC、Evolution Proposal 與 WG21 Paper 的重要價值之一，就是治理記憶。

SECTION

十三、治理風格不是法律形式

兩個專案都使用 RFC，可能有完全不同權力結構。

兩個專案都有 Steering Council，也可能在：

- 選舉；
- 任期；
- 公司影響；
- 委派；
- 公開程度；
- 技術裁決；

上完全不同。

因此 PLDST 必須研究實際權力流，而不能只看制度名稱。

SECTION

十四、原型定義

個人設計者工作室指：

- 核心概念主要由一人形成；
- 語法、語義、Compiler 或文件由同一人或極小團隊掌握；
- 決策可快速回到一致設計原則；
- 制度化提案程序較弱；
- 語言與設計者作品高度重疊。

SECTION

十五、Wirth／Oberon 作為典型

Oberon 由 Niklaus Wirth 與 Jürg Gutknecht 在 ETH 的小型環境中發展，語言、Compiler、OS、文件與硬體實驗彼此緊密連接。[R1][R2][R3]

其治理不是大型社群投票，而是：

明確問題
→ 小型設計核心
→ 完整實作
→ 書面報告
→ 教學與系統驗證

SECTION

十六、個人設計的主要優勢：整體一致性

個人設計者可同時看見：

- 語法；
- 型別；
- Compiler；
- Module；
- OS；
- 教學；
- 硬體限制。

因此：

> Coherence
> uparrow
>
> CoordinationCost
> downarrow

SECTION

十七、刪除能力

大型治理容易增加功能，因為每個提案都有受益者。

個人設計者較能做：

> FeatureRemoval
>
> FeatureRefusal
>
> ConceptualCompression

Wirth 的設計歷史反覆展示：

- 從複雜系統中移除；
- 重寫而非永久補丁；
- 以新語言重設邊界；
- 讓 Compiler 保持可理解。

SECTION

十八、作品責任

個人設計者通常無法把失敗歸因於程序。

其責任鏈短：

> DesignDecision
> →
> Designer

這提高：

- 風格清晰度；
- 決策速度；
- 歸因可見性。

SECTION

十九、個人設計的主要風險：不可擴張

當語言擴大到：

- 數百萬使用者；
- 多個平台；
- 安全回應；
- Package ecosystem；
- 多種實作；
- 企業長期支援；

一個人難以維持所有權威。

SECTION

二十、知識不可轉移

個人直覺可能存在於：

- 未寫出的拒絕理由；
- 對「不像此語言」的感受；
- 實作細節；
- 教學經驗；
- 隱含風格邊界。

若沒有治理記憶：

> FounderExit
> ⇒
> DesignKnowledgeLoss

SECTION

二十一、接班難題

個人作品可採：

- 凍結；
- 指定接班；
- 成立核心團隊；
- 開放多實作；
- 交給標準組織；
- 形成分叉。

  沒有一項是自動正確。

SECTION

二十二、個人設計者不等於任意專制

若設計者：

- 公開規格；
- 提供可運行實作；
- 說明理由；
- 接受技術反證；

- 保持使用者退出自由；

  其權威可具有強專業合法性。

SECTION

二十三、個人設計工作室公式

> G_(studio)
> =
> HighCoherence
> +
> FastDecision
> +
> WholeSystemVision
> -
> Succession
> -
> Scale
> -
> PluralInput

SECTION

二十四、BDFL 與個人工作室不同

BDFL 通常出現在：

- 語言已形成大型社群；
- 提案與實作由多人完成；
- 文件與討論公開；
- 最終仍有一名可信裁決者。

所以：

> BDFL
> ≠
> SoloDesigner

而是：

> DistributedContribution
> +
> CentralFinalJudgment

SECTION

二十五、Python 的 BDFL 時期

Python 長期由 Guido van Rossum 保留最終語言裁決，但：

- PEP 由多位作者提出；
- Core developer 實作；
- Mailing list／論壇討論；
- PEP Editor 管理格式；
- PEP Delegate 可承擔部分決策；
- 社群提供使用證據。

  Guido 的角色是最終收斂點，而不是完成所有工作。

SECTION

二十六、BDFL 的設計價值

BDFL 可維持：

- 語言品味；

- 拒絕能力；
- 跨提案一致性；
- 討論終止；
- 長期方向；
- 少數但高價值判斷。

  可表示為：

> CommunitySearchSpace
> xrightarrowBDFL
> CoherentDecision

SECTION

二十七、PEP 制度的出現

PEP 1 定義 PEP 為：

- 設計文件；
- 技術規格；
- 理由；
- 社群意見收集機制；
- 設計決策歷史紀錄。[R4]

  其重要轉變是：

> OralAuthority
> →
> DocumentedAuthority

SECTION

二十八、PEP 並未消滅 BDFL

在 BDFL 時期：

- PEP 開放論證；
- 作者建立共識；
- 社群留下異議；
- BDFL 或 Delegate 仍可接受／拒絕。

這是：

> OpenDeliberation
> +
> CentralDecision

SECTION

二十九、仁慈的制度含義

「仁慈」不能只靠人格假設。

可制度化為：

- 決策理由公開；
- 接受反證；
- 權力可委派；
- 不任意懲罰異議；
- 維持退出與分叉；
- 對生態成本負責；

- 願意退位。

SECTION

三十、BDFL 的風格負擔

一名裁決者必須持續吸收：

- 語法；
- 型別；
- Library；
- Packaging；
- Tooling；
- 社群；
- 相容；
- 企業需求；
- 教育需求。

當議題增長：

> DecisionLoad
> >
> HumanBandwidth

BDFL 便成為瓶頸。

SECTION

三十一、不可替代性悖論

BDFL 越成功維持一致性，社群越依賴其直覺。

SuccessOfCentralTaste
⇒
SuccessionDifficulty

SECTION

三十二、退出的治理意義

Guido 在 2018 年退出 BDFL 角色後，Python 社群以 PEP 8000 系列提出治理方案，以投票程序選擇 Steering Council 模型，再由 PEP 13 正式化。[R5][R6][R7]

這是一個重要案例：

FounderExit
not⇒
GovernanceVacuum

前提是語言已有：

- Core team；
- PEP 記憶；
- 選民邊界；
- 版本控制；
- 社群信任；
- 正式過渡程序。

SECTION

三十三、BDFL 原型公式

G_(BDFL)
=
OpenContribution
+

SECTION

三十四、PEP 13 的結構

Python 現行治理以五人 Steering Council 為核心。[R7]

其任務包括：

- 維護語言與 CPython 品質；
- 維持貢獻可持續性；
- 建立 PEP 決策程序；
- 優先尋求共識；
- 在其他方法失敗時作最終上訴。

SECTION

三十五、廣泛權力、低頻使用

PEP 13 給 Council 廣泛權力，但要求：

- 盡量少直接使用；
- 優先委派；
- 優先建立標準流程；
- 優先共識；
- 盡可能公開審議與投票。[R7]

這是一種：

> StrongReservePower
> +
> WeakDailyIntervention

SECTION

三十六、制度化的最終性

BDFL 提供人格化最終性。

Steering Council 提供：

- 多人；
- 任期；
- 選舉；
- 衝突利益規則；
- 不信任機制；
- 記錄。

因此：

> PersonalFinality
> →
> ConstitutionalFinality

SECTION

三十七、Council 不是設計委員會的全部

實際 PEP 決策仍可能由：

- PEP Delegate；
- 專業團隊；
- Core developer；
- Packaging／Typing 等子領域流程；

承擔。

Council 更像：

- 權力邊界制定者；
- 委派者；
- 最終上訴法院；
- 危機處理者。

SECTION

三十八、Python 模型的主要優勢

- 保留 PEP 設計記憶；
- 降低單點風險；
- 允許專業委派；
- 具有選舉合法性；
- 可處理跨領域衝突；
- 保持相對清楚的最終權威。

SECTION

三十九、Python 模型的主要風險

- Council 候選人需承擔大量非技術治理；
- 選舉可能偏向知名度與既有核心圈；
- 子領域制度可能碎片化；
- 廣泛權力依賴自我克制；
- 社群仍可能期待「新的 Guido」；

- 共識成本隨生態增長。

SECTION

四十、RFC 的最低功能

Rust RFC 2 將 RFC 定義為重大變更進入語言與標準函式庫的受控路徑。[R8]

重大變更通常包括：

- 非 Bugfix 的語法／語義改變；
- 移除功能；
- Compiler／Library 界面改變；
- Standard library 新增。

SECTION

四十一、RFC 是設計契約，不是願望清單

合格 RFC 通常需要：

- Motivation；
- Detailed design；
- Drawbacks；
- Alternatives；
- Unresolved questions；
- Implementation path；
- Compatibility thinking。

所以：

> Idea
> ≠
> RFC

RFC 是把想法轉成可審查責任。

SECTION

四十二、任何人可提案，不表示任何提案必須被接受

Rust 公開邀請參與，但最終由負責領域的團隊建立共識並作出決定。

因此：

> OpenProposalAccess
> ≠
> EqualDecisionAuthority

SECTION

四十三、領域團隊

Rust 把權力分配給：

- Language team；
- Compiler team；
- Library team；
- Dev tools；
- Infrastructure；
- Moderation；
- 其他專業團隊。

每個團隊對其 Purview 負責。[R9][R10]

SECTION

四十四、聯邦式治理

Rust 模型可表示為：

> Project
> =
> Σ□ Team□(Purview□)
> +
> Council(CrossTeam)

大部分決策在領域內完成。

Leadership Council 主要處理：

- 跨團隊協調；
- 權責空隙；
- 長期專案健康；
- 團隊代表連接；
- 委派與問責。[R10]

SECTION

四十五、共識不是全民一致

Rust 的共識更接近：

- 充分辨認重要反對；
- 調整設計；
- 專業團隊願意承擔；

- 沒有未處理的根本阻礙；
- 形成可實作方向。

它不是要求每一名參與者投贊成票。

SECTION

四十六、Final Comment Period

RFC 制度常以明確期間宣布：

- 團隊準備接受；
- 團隊準備拒絕；
- 最後收集重大異議。

其功能是避免：

- 無限討論；
- 突然裁決；
- 隱形反對；
- 提案長期停滯。

SECTION

四十七、RFC 與實作分離

RFC 被接受後仍需：

- Tracking issue；
- Implementation；
- Test；
- Documentation；

- Stabilization；
- Feature gate；
- Edition／Release integration。

  所以：

> RFCMerged
> not⇒
> FeatureStable

SECTION

四十八、實作證據回流

Rust 允許先：

- Nightly；
- Feature gate；
- 實驗實作；
- Crater／生態測試；
- Stabilization report；

  再決定穩定。

  這使治理從文字論證進入實作證據。

SECTION

四十九、RFC 模型的主要優勢

- 決策歷史清楚；
- 重大變更有統一入口；

- 公開討論；
- 專業領域承擔；
- 反對意見可被保存；
- 實作與穩定分階段；
- 可由團隊取代單一英雄。

SECTION

五十、RFC 模型的主要風險

- 文件與討論成本高；
- 熟悉制度者更有優勢；
- 長討論造成耗竭；
- 團隊帶寬限制；
- 提案可能停滯；
- 跨團隊邊界模糊；
- 共識語言可能掩蓋實際權力；
- 無人願意成為 Champion 時，合理功能也不會前進。

SECTION

五十一、治理即維護勞動

RFC 制度最容易忽略：

> 閱讀、整理、回答、追蹤、主持與拒絕本身都是高成本勞動。

定義：

> C_(governance)
> =
> C_(author)
> +
> C_(reviewer)
> +
> C_(team)
> +
> C_(implementation)
> +
> C_(moderation)

若沒有資源：

> OpenProcess
> →
> VolunteerExhaustion

SECTION

五十二、公開 Evolution

Swift Evolution 允許任何具有良好想法的人參與：

- Pitch；
- Forum discussion；
- Proposal refinement；
- Formal review；
- Acceptance／rejection；
- Implementation tracking。[R11][R12]

SECTION

五十三、Language Steering Group

Swift Language Steering Group 透過 Evolution process 引導語言與 Standard Library，並具有相關演化權威。[R13]

因此：

> PublicDeliberation
> +
> SteeringGroupDecision

SECTION

五十四、企業資源的結構性角色

Swift 是開源專案，但 Apple 長期提供：

- 主要 Compiler 工程；
- Xcode／平台整合；
- Release resources；
- ABI／SDK 約束；
- 大量核心貢獻者；
- 產品採用場景。

因此不能只由論壇形式判斷權力。

SECTION

五十五、正式權力與資源權力

定義：

> P_(formal)
> ≠
> P_(resource)

即使提案程序公開，能夠：

- 實作大型功能；
- 維持多平台；
- 整合 Toolchain；
- 承擔多年遷移；

的組織仍具有較高實際議程能力。

SECTION

五十六、Swift 的混合治理

可表示為：

> G_(Swift)
> =
> OpenEvolution
> +
> SteeringAuthority
> +
> CorporateEngineeringCapacity
> +
> CommunityImplementation

SECTION

五十七、Vision document 與方向設定

Swift 不只逐案審查功能，也使用：

- Focus area；
- Vision document；
- Steering group；
- Workgroup；
- Release goal；

  建立中程方向。

  這降低完全由零散提案塑造語言的風險。

SECTION

五十八、混合制的主要優勢

- 有大型工程資源；
- 公開設計討論；
- 提案與發布目標連接；
- 可維持平台級品質；
- Steering Group 可保存方向；
- 社群能提出與審查語言功能。

SECTION

五十九、混合制的主要風險

- 公司需求可能影響議程；
- 外部貢獻者難以匹配實作資源；
- 正式公開與實際決策能力可能不對稱；

- 產品發布節奏限制設計時間；
- 公司與社群目標不一致時，權力邊界需重新檢驗。

SECTION

六十、不能簡化為「企業控制」

若所有核心能力都由單一企業秘密決定，才接近封閉企業語言。

Swift 已具有：

- 公開 Repository；
- 公開 Proposal；
- 公開 Review；
- 公開 Steering Group；
- 外部貢獻；
- 多平台擴張。

更精確的判定是：

> 企業資源高度集中的公開社群治理。

SECTION

六十一、標準委員會的對象不同

開源語言專案常同時控制：

- Reference implementation；
- Release；
- Specification。

ISO C++ WG21 主要產出：

> InternationalStandard

實際 Compiler 由多個實作者完成。

SECTION

六十二、WG21 的組成

WG21 由 ISO／IEC JTC1／SC22 下的國家成員與認可專家構成，並透過：

- Study Group；
- Evolution Working Group；
- Library Working Group；
- Core Working Group；
- National body；
- Plenary；

  推進標準工作。[R14][R15]

SECTION

六十三、書面提案政治

C++ 提案以 Paper 進入程序。

提案者通常需要：

- 描述問題；
- 提供設計；
- 比較替代；
- 回應既有標準；

- 準備實作經驗；
- 參與會議；
- 修訂多輪；
- 尋求子群共識。

SECTION

六十四、共識不是簡單多數

WG21 由 Convener、Subgroup chair 與投票程序判定是否形成足夠共識。[R15][R16]

其目標不是：

> 51% wins

而是：

- 廣泛支持；
- 重要反對已處理；
- 多實作者可接受；
- 國家代表能支持；
- 標準文字可整合。

SECTION

六十五、多實作約束

C++ 標準不能只對一個 Compiler 方便。

必須考慮：

- GCC；
- Clang；

- MSVC；
- EDG；
- Embedded；
- 不同 ABI；
- 不同 OS；
- 大量歷史程式。

  這增加證據，也增加速度成本。

SECTION

六十六、標準文字與語言實作分離

WG21 通過功能後：

- 標準文字需整合；
- Compiler 各自實作；
- Library Vendor 實作；
- Conformance 逐步提升；
- 使用者面對版本差異。

  因此：

> StandardAccepted
> not⇒
> UniversalAvailability

SECTION

六十七、委員會模型的主要優勢

- 多實作代表；
- 國際參與；
- 長期相容責任；
- 高規格精度；
- 多領域證據；
- 不易由單一公司永久壟斷正式標準；
- 可處理產業基礎設施級承諾。

SECTION

六十八、委員會模型的主要風險

- 高參與成本；
- 公司與國家代表資源不對稱；
- 旅行與會議門檻；
- 決策時間長；
- 提案累積；
- 折衷功能；
- 整體品味弱化；
- 標準、Compiler 與使用者部署不同步。

SECTION

六十九、委員會不等於無設計者

C++ 仍受到：

- Bjarne Stroustrup；

- Herb Sutter；
- 各 Working Group 主席；
- 提案 Champion；
- Compiler／Library 領袖；
- 大型企業實作者；

的強影響。

委員會只是把個人影響放入更複雜的正式程序。

SECTION

七十、Go 的混合特性

Go 由小型創始設計團隊形成，後來建立公開 Proposal process，但核心語言變更仍由資深設計與 Review 群體高度克制。[R17][R18]

因此：

> G_(Go)
> =
> SmallDesignCore
> +
> OpenProposalIntake
> +
> ConservativeReview

SECTION

七十一、Proposal 不等於 RFC 議會

Go Proposal process 接收重要語言、Library 與 Tool 變更。

但其主要目標是：

- 收集具體問題；
- 公開追蹤；
- 避免重複；
- 由 Review group 決定；
- 保持語言穩定。

SECTION

七十二、拒絕作為治理能力

Go 官方曾明確表示，由於語言改變成本高、收益常不確定，多數語言變更提案最終會被拒絕。[R18]

這顯示：

> GovernanceQuality
> ≠
> AcceptanceRate

SECTION

七十三、Go 模型的優勢

- 強烈保守性；
- 小核心方向清楚；
- 公開提案紀錄；
- 決策速度通常高於國際委員會；
- 相容與工具一致性高；
- 不需要每個改變都建立龐大 RFC 政治。

SECTION

七十四、Go 模型的風險

- 最終裁決圈較小；
- 外部社群可能感到意見已被聽見但未真正改變議程；
- 拒絕理由品質取決於維護帶寬；
- 公司資源與語言方向具有結構性關聯；
- 創始設計品味制度化程度有限。

SECTION

七十五、個人工作室

提案通常來自：

- 設計者；
- 近身協作者；
- 實作問題；
- 教學觀察；
- 系統整合需求。

准入窄，但轉換快速。

SECTION

七十六、BDFL＋PEP

任何人理論上可形成 PEP，但需要：

- Sponsor／核心互動；

- 完整規格；
- 社群共識工作；
- 最終裁決。

SECTION

七十七、Rust RFC

提案入口公開，並以 Git Pull Request 記錄。

但成功需要：

- Champion；
- 領域團隊注意；
- 長期討論；
- 實作可能性；
- 社群需求。

SECTION

七十八、Swift Evolution

Pitch 與論壇討論公開，正式 Proposal 需成熟到可 Review，並由 Steering Group 決定。

SECTION

七十九、WG21

理論上可透過國家成員或相關參與路徑加入，但有效提案需要：

- Paper；
- 會議；

- Presenter；
- Subgroup；
- 多輪修訂；
- 共識。

  形式入口與實際能力門檻差距較大。

SECTION

八十、開放性的正確公式

> EffectiveAccess
> =
> FormalAccess
> ×
> DocumentationAbility
> ×
> SocialAccess
> ×
> ImplementationCapacity
> ×
> Time

只看「任何人都可以提出」會高估實際開放性。

SECTION

八十一、議程權常比投票權重要

> P_(agenda)
> >
> P_(vote)

因為大量提案在進入正式裁決前便會：

- 無人回應；
- 被判定非目標；
- 缺乏 Champion；
- 延期；
- 要求先實驗；
- 被既有 Roadmap 排除。

SECTION

八十二、個人設計者的議程

設計者直接決定：

- 什麼是問題；
- 哪些問題值得解；
- 哪些限制可接受；
- 何時重寫。

一致性最高，代表性最低。

SECTION

八十三、BDFL 的議程

社群可提出，但 BDFL 的興趣、拒絕信號與品味會影響：

- 作者是否投入；
- Core developer 是否支持；
- 討論是否繼續。

SECTION

八十四、RFC 聯邦的議程

Rust 等專案的議程由：

- Roadmap；
- Team capacity；
- Project goal；
- Maintainer interest；
- 實作壓力；
- 社群需求；

  共同形成。

  沒有中央獨裁，不等於沒有議程中心。

SECTION

八十五、企業—社群混合制的議程

公司產品、平台期限與工程團隊能力可能強烈決定：

- 哪些功能有完整實作；
- 哪些方向進入 Release goal；
- 哪些問題獲得專職人力。

SECTION

八十六、委員會議程

WG21 的議程由：

- Paper volume；
- Subgroup priority；
- Chair scheduling；
- 國家與公司投入；
- 實作者關切；
- 標準週期；

  共同形成。

SECTION

八十七、個人設計者

> D=1

優點是清楚。

風險是單點與不可申訴。

SECTION

八十八、BDFL

> D=1
>
> P_(input)≫1

多人輸入、一人收斂。

SECTION

八十九、Steering Council

> D=n

其中 n 為小型、任期制、可選舉或任命的群體。

SECTION

九十、領域團隊

> D=D_(purview)

不同技術範圍由不同團隊負責。

跨團隊問題再上升至 Council。

SECTION

九十一、Steering Group

正式裁決由專業 Steering Group 承擔，公開 Review 提供證據與意見。

SECTION

九十二、標準委員會

決策經多層：

Study Group
→ Evolution／Library／Core subgroup
→ Plenary
→ National body ballot
→ ISO publication
最後權威分散於制度鏈。

SECTION

九十三、最終決定的必要性

若制度沒有可辨認的結束點：

> DiscussionTime→∞

語言便無法演化。

治理必須允許：

- 接受；
- 拒絕；
- 延期；
- 撤回；
- 要求實驗；
- 限縮範圍。

SECTION

九十四、品味不是可以完全投票的量

語言品味包含：

- 功能是否像這門語言；
- 概念是否過度一般；
- 表面是否太密；
- 例外是否可接受；
- 是否破壞教學模型；
- 是否與未來方向衝突。

它常無法由單一 Benchmark 決定。

SECTION

九十五、個人品味

保存方式：

> Taste
> =
> DesignerMemory

最強也最脆弱。

SECTION

九十六、文件化品味

PEP、Design FAQ、Rationale、Style guide 把部分品味轉成：

> Taste
> →
> RecordedPrinciple

SECTION

九十七、團隊化品味

Rust Language team、Swift Language Steering Group、Go Design group 等以：

- 長期參與；
- 同行評審；
- 共同案例；
- Mentorship；
- Review practice；

形成群體品味。

SECTION

九十八、委員會品味

委員會較難保持單一美學，往往轉向：

- 可實作；
- 不破壞；
- 多方接受；
- 與標準一致；
- 有足夠支持。

  其產物可能更穩健，也更折衷。

SECTION

九十九、品味制度化的損耗

> TasteTransmission
> =
> Principles
> +
> Examples
> +
> RejectedCases
> +
> Mentorship
> +
> ImplementationExperience

只有抽象口號不足以保存設計風格。

SECTION

一百、提出者不一定是實作者

一項功能可由：

- 使用者提出；
- 設計團隊批准；
- Compiler team 實作；
- Library team 整合；
- Release team 發布；
- Tool team 支援。

因此：

> ProposalAuthorship
> ≠
> ImplementationOwnership

SECTION

一百零一、個人工作室的實作一致

設計者可能直接撰寫 Compiler，使：

> Specification
> ≈
> Implementation

優點是語義與可行性靠近。

缺點是缺乏獨立驗證。

SECTION

一百零二、BDFL＋Reference implementation

Python 長期由 CPython 提供事實中心。

PEP 可由 Guido 或 Delegate 接受，但若無人實作：

> AcceptedDesign
> not⇒
> ReleasedFeature

SECTION

一百零三、Rust 的分階段實作

RFC 接受、Nightly 實作、Feature gate、Tracking issue、Stabilization 是不同階段。

這降低文字提案直接永久化的風險。

SECTION

一百零四、Swift 的資源集中

大型語言功能常需：

- Parser；
- Type checker；
- SIL；
- LLVM；
- IDE；
- Debugger；
- Migration；

- Apple platform integration。

  能提供完整鏈條者具有高度實作權。

SECTION

一百零五、C++ 的多實作驗證

標準提案理想上需實作經驗，但最終由多 Compiler 各自落地。

這增加可攜性證據，也延長普及時間。

SECTION

一百零六、相容是治理債務

> Debt_(compatibility)
> =
> Users
> ×
> CodeAge
> ×
> EcosystemSize
> ×
> DeploymentLifetime

語言越成功，治理越不自由。

SECTION

一百零七、個人設計者可重寫

小型研究語言可：

- 發布 Oberon 新版本；
- 重做 Compiler；

- 修改 Module；
- 以新系統取代舊系統。

  但採用規模也較小。

SECTION

一百零八、Python 的相容治理

PEP、Deprecation、Release cycle 與 Steering Council 必須考慮：

- Python 2／3 歷史；
- Library；
- 教材；
- Packaging；
- C API；
- 多實作。

SECTION

一百零九、Rust 的 Edition 機制

Rust 以 Edition 在保持 Crate 間互通的同時，允許部分語法與慣例演化。

這是治理與語言機制結合的例子：

> CompatibilityPolicy
> →
> LanguageFeature

SECTION

一百一十、Swift 的 Source／ABI 約束

Swift 需要考慮：

- Source compatibility；
- ABI stability；
- Standard Library；
- Apple SDK；
- Migration tooling；
- 多平台。

治理不能只判定功能是否漂亮。

SECTION

一百一十一、C++ 的歷史包袱

C++ 面對數十年程式與多實作，功能移除極難。

委員會的保守與複雜，部分是其責任規模的結果。

SECTION

一百一十二、公開討論的價值

公開流程使未來設計者知道：

- 為何加入；
- 哪些替代被拒；
- 哪些風險已知；
- 哪些問題未解；
- 誰承擔實作；
- 何時可以重新提出。

SECTION

一百一十三、文件也可能製造假透明

大量公開文字不保證：

- 關鍵決策不在私下；
- 資源分配公開；
- 反對者有相同時間；
- 最終理由完整；
- 提案不因社會位置而受差別待遇。

  因此：

> Transparency
> ≠
> DocumentVolume

SECTION

一百一十四、可追蹤性指標

> T_G
> =
> T_(proposal)
> +
> T_(discussion)
> +
> T_(decision)
> +
> T_(implementation)
> +
> T_(release)

五段皆能追蹤，才形成完整治理記憶。

SECTION

一百一十五、拒絕紀錄的重要性

被拒功能可在未來重新出現。

若沒有拒絕理由：

> RepeatedProposalCostuparrow

所以成熟制度應保存：

- Rejected；
- Deferred；
- Withdrawn；
- Superseded；
- Implemented；
- Reverted。

SECTION

一百一十六、接班不是人事問題而已

接班必須轉移：

- 技術權威；
- 社群信任；
- 品味；
- 密碼與資產；
- Release control；

- 品牌；
- 法律實體關係；
- 衝突處理。

SECTION

一百一十七、個人設計者接班

最常見路徑：

> Founder
> →
> Maintainer

但 Maintainer 未必具有重新設計權。

SECTION

一百一十八、BDFL 接班

指定下一名 BDFL 容易製造：

- 合法性不足；
- 模仿創始者；
- 權力競爭；
- 風格漂移。

Python 選擇 Council，而非新 BDFL，正是避免人格複製。

SECTION

一百一十九、RFC 聯邦接班

團隊制度透過：

- Membership；
- Lead；
- Delegate；
- Council representative；
- Mentorship；
- Emeritus；

  讓角色逐步轉移。

  其風險是人員名單存在，實際知識卻未轉移。

SECTION

一百二十、委員會接班

WG21 等制度依靠：

- Chair；
- Convener；
- National body；
- Working Group；
- Paper history；
- 標準程序。

  制度壽命可超過個人，但整體方向可能變得慣性化。

SECTION

一百二十一、接班成熟度公式

> S_G
> =
> DocumentedAuthority
> +
> DistributedKnowledge
> +
> LegitimateSelection
> +
> AssetContinuity
> +
> ConflictProcedure

SECTION

一百二十二、企業參與不是單純污染

企業可提供：

- 全職 Compiler engineer；
- CI；
- Security；
- Release；
- Hardware；
- Cloud；
- Legal；
- 標準會議；
- Long-term support。

  沒有資源，治理可能只存在於文件。

SECTION

一百二十三、企業權力的主要形式

> P_(corporate)
> =
> P_(employment)
> +
> P_(implementation)
> +
> P_(infrastructure)
> +
> P_(distribution)
> +
> P_(agenda)

不必擁有正式多數，也能形成實質影響。

SECTION

一百二十四、治理防護

可降低集中風險的機制包括：

- Conflict-of-interest；
- 任期；
- 公司席次限制；
- 公開投票；
- 多實作；
- 獨立基金會；
- 商標與資產分離；

- Fork freedom；
- 外部 Core member。

SECTION

一百二十五、企業—社群同構與衝突

若公司產品成功依賴語言健康：

> CorporateGoal
> ≈
> CommunityGoal

治理可高效。

但在：

- 平台優先序；
- 封閉產品整合；
- 商業期限；
- 競爭策略；
- 非核心平台；

上可能分離。

SECTION

一百二十六、Fork 是最後否決權

開源語言使用者理論上可：

- Fork Compiler；
- 建立方言；

- 另建標準；
- 保持舊版本；
- 建立新社群。

因此：

> ExitPower
> ≠
> VoicePower

即使無法改變原專案，仍可能退出。

SECTION

一百二十七、Fork 的實際成本

> C_(fork)
> =
> Compiler
> +
> Library
> +
> Tooling
> +
> Packages
> +
> Brand
> +
> Users
> +
> Governance
> +
> Security

大型語言的 Fork 並不容易。

SECTION

一百二十八、Fork 威脅的治理作用

可行 Fork 能限制中央任意權力。

但過度依賴 Fork 也表示制度無法吸收合理分歧。

SECTION

一百二十九、標準語言的分叉

C++ 可由 Compiler extension、Dialect、Vendor feature 形成局部分叉，但正式標準仍提供重聚中心。

SECTION

一百三十、治理矩陣

軸　個人設計者　BDFL＋文件制　RFC＋團隊聯邦　Steering Group 混合制　標準委員會

主要案例　Wirth／Oberon　Guido 時期 Python　Rust　Swift　C++／WG21

議程權　設計者　BDFL＋核心圈　Team／Roadmap／Champion　Steering Group＋工程資源　Paper／Subgroup／Chair

提案入口　窄　公開但需成熟　公開 RFC　公開 Evolution　形式開放、實際高門檻

最終裁決　個人　個人／Delegate　領域團隊　Steering Group　多層共識與表決

實作　設計者／小團隊　Core developer　Compiler／Library team　Apple＋社群團隊　多 Vendor

一致性　很高　高　中高、靠 Team　中高、靠 Steering　中，較多折衷

決策速度　快　中快　中慢　中　慢

治理記憶　低至中　高，PEP　高，RFC／Issue　高，Proposal／Forum　高，Paper／Minutes

接班　弱　中，需制度轉換　中高，團隊化　中高　高

主要優勢	整體作品　品味＋社群　公開責任與分工　資源＋公開演化　多實作與國際標準
主要風險	單點、不可擴張　過載、不可替代　耗竭、停滯、權責模糊　企業資源不對稱　高成本、折衷、緩慢

SECTION

一百三十一、治理目標函數

> $$J_G = \alpha Q_{decision} + \beta C_{coherence} + \gamma T_{traceability} + \delta S_{succession} + \varepsilon I_{implementation} - \lambda L_{latency} - \mu K_{coordination} - \nu R_{capture}$$

其中 $R_{capture}$ 表示個人、公司、派系或程序被俘獲的風險。

SECTION

一百三十二、沒有全域最優治理

最適治理取決於：

> $$G^* = f($$

研究語言與產業基礎設施需要不同制度。

SECTION

一百三十三、誤判一：RFC 就是民主

RFC 是：

- 提案格式；
- 討論容器；
- 決策記錄；
- 共識工具。

  它不是自動的全民公投。

SECTION

一百三十四、誤判二：BDFL 就是獨裁

若 BDFL：

- 依公開文件決策；
- 接受社群論證；
- 委派專業；
- 可被 Fork；
- 願意退出；

  其實際治理可能比名義委員會更可預測。

SECTION

一百三十五、誤判三：委員會沒有設計品味

委員會仍由：

- 強設計者；
- Champion；
- 實作者；
- Chair；
- 歷史標準；

  形成方向。

  只是品味變成協商後的合成結果。

SECTION

一百三十六、誤判四：開源代表權力平等

Source available 不表示：

- 合併權平等；
- 發布權平等；
- CI 權平等；
- 商標權平等；
- 全職時間平等；
- 社群聲望平等。

SECTION

一百三十七、誤判五：投票可以解決技術真理

投票可決定制度行動，不能改變：

- Soundness；

- Performance；
- Compatibility；
- Implementation feasibility；
- Security。

專業證據仍必要。

SECTION

一百三十八、誤判六：共識表示所有人同意

成熟共識是：

> 重大反對已被辨認、回答或明確記錄，負責團隊願意承擔結果，程序可以結束。

SECTION

一百三十九、誤判七：創始者退出後風格會自然保存

若沒有：

- 案例庫；
- 拒絕理由；
- Design rationale；
- Mentorship；
- 接班程序；

風格可能迅速漂移。

SECTION

一百四十、創始者瓶頸

症狀：

- 所有決策等一人；
- 無人敢拒絕創始者；
- 創始者疲勞；
- 小問題也上升；
- 接班停滯。

SECTION

一百四十一、RFC 墳場

症狀：

- 大量提案無回應；
- 無 Champion；
- 無人關閉；
- 狀態不明；
- 作者反覆催促；
- 討論與實作脫節。

SECTION

一百四十二、程序俘獲

熟悉程序者可：

- 控制模板；
- 延長討論；
- 要求無限證據；

- 以程序阻擋新參與者；
- 將價值判斷包裝為格式問題。

SECTION

一百四十三、公司俘獲

正式制度仍存在，但：

- 核心人員同公司；
- Roadmap 由產品決定；
- CI／Release 由公司控制；
- 外部提案無實作資源；
- 品牌與商標限制 Fork。

SECTION

一百四十四、委員會堆疊

每個利益群體加入一項功能，最終：

> Language
> =
> Σ☒ Compromise☒

但缺乏刪除與整體重構。

SECTION

一百四十五、品味神秘化

決策只說：

不像這門語言

卻不提供：

- 原則；
- 例子；
- 代價；
- 替代方案；
- 可重複判準。

這讓品味變成不可質疑權力。

SECTION

一百四十六、無責任民意

大量使用者要求功能，但不承擔：

- Compiler；
- Documentation；
- Migration；
- Security；
- 十年維護。

治理不能把反應數量直接當作設計證據。

SECTION

一百四十七、維護者寡頭化

維護者具有正當專業權威，但若：

- 新人無進入路徑；
- Membership 不透明；

- 反對被社會性排除；
- 任期無限；
- 決策理由不足；

  專業團隊可能封閉化。

SECTION

一百四十八、十四個治理軸

PLDST 後續應為每位設計者或語言記錄：

- Founder centrality；
- Agenda openness；
- Proposal accessibility；
- Decision finality；
- Delegation；
- Implementation ownership；
- Release ownership；
- Compatibility burden；
- Transparency；
- Dissent preservation；
- Corporate concentration；
- Succession；
- Fork viability；
- Governance adaptability。

SECTION

一百四十九、評分不是道德排名

可使用：

> v⊠∈[0,5]

但每一軸必須附：

- 史實；
- 文件；
- 時期；
- 案例；
- 反例；
- 信心水平。

SECTION

一百五十、時間切片

Python 至少應分為：

早期個人設計
BDFL＋社群
PEP 制度成熟
後 BDFL 過渡
Steering Council
Rust、Swift、C++、Go 也需分期。

SECTION

一百五十一、治理狀態轉移

定義：

> G⊠
> xrightarrowEvent
> G_(t+1)

Event 可包括：

- 創始者退出；
- 生態爆發；
- 重大分裂；
- 安全事件；
- 企業收購；
- 新基金會；
- 標準化；
- Release crisis；
- 維護者耗竭。

SECTION

一百五十二、治理風格指紋

可建立：

FounderCentrality: 4
ProposalOpenness: 5
DecisionConcentration: 2
TeamFederation: 5
CorporateConcentration: 2
SuccessionFormalization: 4
HistoricalTraceability: 5
ImplementationCoupling: 4
但指紋只在指定時期有效。

SECTION

一百五十三、MVP 階段不需要假裝議會

新語言只有一至三名實作者時，建立龐大委員會常是形式主義。

較合理：

創始設計原則
公開 Issue
決策紀錄
Reference implementation
版本政策

SECTION

一百五十四、先建立治理記憶，再擴權

在社群尚小時，最重要的不是投票，而是保存：

- 問題；
- 選擇；
- 拒絕；
- 實驗；
- 版本；
- 實作證據。

SECTION

一百五十五、何時從個人轉向團隊

觸發條件包括：

DecisionLoad
+
EcosystemRisk
+

SECTION

一百五十六、何時需要 RFC

RFC 適合：

- 語義改變；
- Syntax；
- Type system；
- Standard library 核心；
- Compatibility；
- Security model；
- Governance；
- Package protocol。

  不適合所有小型 Bugfix。

SECTION

一百五十七、RFC 必須有 Champion

沒有 Champion 的提案容易成為：

> DocumentWithoutAgency

Champion 應負責：

- 回答；
- 修訂；
- 協調實作；

- 整理異議；
- 推進狀態；
- 必要時撤回。

SECTION

一百五十八、AI 可協助但不能自動合法化決策

AI 可以：

- 搜尋先例；
- 整理討論；
- 比較替代；
- 找出未回答問題；
- 模擬遷移；
- 產生測試；
- 建立決策語料。

但：

> AIAnalysis
> ≠
> GovernanceLegitimacy

SECTION

一百五十九、AI 代理提案風險

若 AI 大量產生高品質表面 RFC：

> ProposalVolumeuparrow
>
> HumanReviewCapacity=constant

可能造成治理拒絕服務攻擊。

SECTION

一百六十、提案配額不如證據門檻

可要求 AI／人類提案附：

- Minimal prototype；
- Compatibility scan；
- Alternatives；
- Negative cases；
- Tooling impact；
- Migration plan；
- Maintainer sponsor。

SECTION

一百六十一、AI 模擬設計者品味

PLDST 可讓 AI 預測：

Guido-style likely objection
Wirth-style simplification
Hickey-style decomplecting
Stroustrup-style compatibility concern
Rust-team-style RFC questions
WG21-style implementation evidence
但預測不能冒充真實裁決者。

SECTION

一百六十二、AI 治理需要來源可追蹤

任何 AI 建議應輸出：

> Recommendation
> +
> Evidence
> +
> Uncertainty
> +
> AffectedStakeholders
> +
> Reversibility

SECTION

一百六十三、提案狀態機

> Draft
> →
> Sponsored
> →
> Discussion
> →
> Review
> →
> {
> Accepted,
> Rejected,
> Deferred,
> Withdrawn
> }

Accepted 後：

> Accepted
> →
> Experimental
> →
> Implemented
> →
> Stabilized
> →
> Released

SECTION

一百六十四、每一狀態的責任人

Draft：Author
Sponsored：Sponsor
Discussion：Moderator／Champion
Review：Decision body
Experimental：Implementation owner
Stabilized：Language／Library owner
Released：Release manager

SECTION

一百六十五、決策紀錄格式

每項決策至少保存：

Problem
Context
Proposal
Alternatives
Evidence
Compatibility
Security
Implementation
Dissent
Decision
Decision authority

Review date
Revisit trigger

SECTION

一百六十六、可逆性分級

> Reversibility∈
> {
> High,
> Medium,
> Low,
> NearZero
> }

語法與穩定 ABI 通常低可逆。

Tooling 實驗通常高可逆。

治理門檻應與不可逆性成正比。

SECTION

一百六十七、權力—責任配對

權力　對應責任

接受功能　維護與相容

拒絕功能　提供可理解理由

控制發布　提供品質與安全

控制議程　公開優先序

指定團隊　提供接班與撤換

代表社群　披露利益衝突

控制實作　接受可攜性與外部審查

SECTION

一百六十八、Governance Budget

每個 Release 應估算：

> B_G
> =
> ReviewerHours
> +
> ImplementationHours
> +
> DocumentationHours
> +
> MigrationHours
> +
> CommunityHours

沒有預算的開放治理只是把成本隱藏給志願者。

SECTION

一百六十九、PLDST 已超越人物傳記

第四部前三篇比較：

- Wirth、Ritchie、Stroustrup 的機器現實主義；
- Guido、Matz、Wall 的三種人本語言設計；
- Backus、McCarthy、Hickey 的三種簡單性。

本篇再證明：

> 同一位設計者的技術風格，只有放進權力、制度與接班中，才形成完整的設計風格。

SECTION

一百七十、設計風格包含治理風格

> DesignerStyle
> =
> TechnicalPreference
> +
> BurdenAllocation
> +
> EvidenceStandard
> +
> DecisionStyle
> +
> GovernanceStyle

SECTION

一百七十一、個人風格如何變成共同體風格

轉換鏈為：

> PersonalTaste
> →
> RepeatedDecision
> →
> WrittenRationale
> →
> CommunityNorm
> →
> FormalProcess
> →
> Institution

每一步都可能失真。

SECTION

一百七十二、制度化不等於去人格化

任何制度仍需要：

- 判斷；
- 勇氣；
- 拒絕；
- 綜合；
- 說服；
- 責任；
- 信任。

程序只能規範權力，不能取代設計智慧。

SECTION

一百七十三、人格化不等於無制度

優秀個人設計者也可依賴：

- 原則；
- 實驗；
- 文件；
- Compiler；
- 教學；
- 反例；

- 長期修正。

SECTION

一百七十四、第四部最終命題

> 【ProgrammingLanguage
> =
> Design
> +
> Implementation
> +
> Community
> +
> Governance】

若只研究人物思想，不研究制度，PLDST 會退化為設計者傳記。

若只研究制度名稱，不研究實際權力，PLDST 會退化為治理組織圖。

SECTION

一百七十五、五種治理憲法

個人設計者

> 由理解整體系統的人負責完整取捨，以作品的一致性證明其權威。

BDFL

> 讓社群探索設計空間，由可信且負責的最終裁決者維持語言品味與決策終止。

RFC＋團隊聯邦

讓重大改變留下公開論證，由承擔實作與維護責任的領域團隊形成決策。

Steering Group 混合制

以公開演化程序吸收社群智慧，由專業 Steering Group 與集中工程資源把設計轉成平台能力。

標準委員會

以多實作、多組織與國家代表的正式共識，交換決策速度以取得長期產業相容性。

SECTION

一百七十六、沒有制度可以同時最大化全部目標

Fast
+
Open
+
Coherent
+
Representative
+
LowCost
+
HighlyCompatible

不可能全部無條件最大化。

SECTION

一百七十七、治理選擇是一種複雜度配置

【Individual：把複雜度集中到設計者;
BDFL：把最終性集中、把探索分散;
RFC：把論證公開、把責任分配到團隊;
Steering：把方向集中、把意見與實作部分開放;
Committee：把權威分散、把協調成本制度化】

SECTION

一百七十八、治理的真正單位

治理不是：

一人
或
多人
而是：

WhoCanCauseWhichChange
UnderWhatEvidence
WithWhoseResources
AndWhoPaysLater

SECTION

一百七十九、最終 PLDST 判定

本文將五種治理風格判定為：

【IndividualDesigner
：
Coherent Studio Governance;
BDFL
：
Central-Taste Deliberative Governance;
RFCFederation
：
Documented Team-Responsibility Governance;

SECTION

一百八十、本文最後命題

> 語言治理的目的，不是讓所有人都擁有同樣的決定權，而是讓提出權、裁決權、實作權、發布權與長期責任之間形成可理解、可追蹤、可接班的關係。

因此：

> 【GoodLanguageGovernance
> =
> DistributedKnowledge
> +
> ExplicitAuthority
> +
> AccountableDecision
> +
> SustainableLabor
> +
> Succession】

第四部至此完成。

PLDST 下一階段將不再只分析歷史人物與語言，而要把比較結果轉成：

- 評估矩陣；
- 決策語料庫；
- SKILL；
- AI 模擬；
- 總論。

研究單元：語言治理風格

比較類型：
1. Coherent Studio Governance
2. Central-Taste Deliberative Governance
3. Documented Team-Responsibility Governance
4. Directed Open-Evolution Governance
5. Multi-Implementation Consensus Governance
核心問題：
誰能提出？
誰能設定議程？
誰能裁決？
誰能實作？
誰能發布？
誰承擔相容？
誰保存品味？
誰接班？
個人設計者：
優勢＝一致、快速、可刪除
風險＝單點、不可擴張、知識不可轉移
BDFL：
優勢＝公開探索＋最終收斂
風險＝過載、人格依賴、接班困難
RFC 聯邦：
優勢＝公開理由、團隊責任、治理記憶
風險＝耗竭、停滯、程序門檻、權責邊界
Steering 混合：
優勢＝公開演化、平台資源、方向能力
風險＝企業資源不對稱、正式與實際權力差距
標準委員會：
優勢＝多實作、國際合法性、長期相容
風險＝緩慢、昂貴、折衷、複雜堆疊
PLDST 核心結論：
治理是設計風格的制度化版本。

案例 原始問題 制度決策 權力配置 主要收益 主要代價 標記

Wirth／Oberon 系統複雜且難以整體理解 小型設計—實作閉環 高度集中 一致與可驗證 單點與小規模 GOV-STUDIO

Python／BDFL 社群擴大但需維持品味 PEP＋BDFL／Delegate 公開輸入、單點收斂 決策終止與一致 過載與接班 GOV-BDFL

Python／PEP 13 Guido 退出 五人 Council、選舉與委派 憲法式最終權力 可接班、低單點 Council 負擔 GOV-COUNCIL

Rust／RFC 2 早期功能加入過度自由 重大變更必須 RFC 公開提案、團隊裁決 設計記憶與成熟度 程序成本 GOV-RFC

Rust／RFC 1068　Core team 負擔與領域增長　多子團隊　Purview 分權　專業責任　邊界協調　GOV-FED

Rust／RFC 3392　跨團隊領導空隙　Leadership Council　團隊代表＋委派　問責與協調　代表性設計成本　GOV-LC

Swift Evolution　開源後需公共演化　Pitch／Proposal／Review　公開討論、LSG 裁決　社群輸入與方向　資源不對稱　GOV-HYBRID

C++／WG21　多實作與國際標準　Paper／Subgroup／Consensus　多層委員會　相容與代表性　慢與折衷　GOV-ISO

Go Proposal　使用者提案增長但需克制　公開 Issue＋Review group　開放入口、小核心裁決　穩定與簡潔　決策圈較小　GOV-DESIGNTEAM

[R1] Niklaus Wirth, The Programming Language Oberon, ETH Zürich.

— Oberon 語言報告、設計與規格集中性。

[R2] Niklaus Wirth, “The History of Modula-2 and Oberon,” ETH Zürich.

— Modula-2／Oberon 的設計演化、模組化與簡化歷史。

[R3] ETH Zürich Department of Computer Science, “Niklaus Wirth and the Art of Simplicity” and Project Oberon historical materials.

— 小型整體系統、簡潔、語言—Compiler—OS 共同設計。

[R4] Python Enhancement Proposals, PEP 1 – PEP Purpose and Guidelines.

— PEP 的設計文件、規格、理由、共識責任與歷史紀錄功能。

[R5] Python Enhancement Proposals, PEP 8000 – Python Language Governance Proposal Overview.

— Guido 退出後治理模式選擇的總覽。

[R6] Python Enhancement Proposals, PEP 8001 – Python Governance Voting Process.

— 新治理模型的投票與轉換程序。

[R7] Python Enhancement Proposals, PEP 13 – Python Language Governance and PEP 8016 – The Steering Council Model.

— 五人 Steering Council、任務、廣泛但低頻使用的權力、委派、選舉與公開原則。

[R8] Rust RFC Book, RFC 0002 – RFC Process.

— 重大變更的 RFC 准入、公開 PR、共識與接受／拒絕程序。

[R9] Rust RFC Book, RFC 1068 – Rust Governance.

— Core team 向多領域子團隊擴張的治理結構。

[R10] Rust RFC Book, RFC 3392 – Leadership Council, and Rust official Governance page.

— Leadership Council、團隊 Purview、委派、跨團隊協調與現行團隊結構。

[R11] Swift.org, Swift Evolution.

— 公開 Pitch、討論、Proposal、Review 與 Release goal。

[R12] Swift.org, Contributing／Swift Evolution Process.

— 語言與 Standard Library 公開介面變更的演化範圍。

[R13] Swift.org, Language Steering Group and “Evolving the Swift Project Workgroups.”

— Language Steering Group 的演化權威與工作群組制度。

[R14] Standard C++, The Committee: WG21.

— WG21 的 ISO／IEC 結構、國家成員與專家參與。

[R15] Standard C++, Meetings and Participation.

— Subgroup、會議、提案、共識與參與方式。

[R16] Standard C++, SD-4: WG21 Practices and Procedures.

— Paper、Presenter、程序、國家代表與會議規則。

[R17] Go Project, Proposal Process.

— 語言、Library 與 Tool 的重要改變提案流程。

[R18] Go Blog, “Go 2, Here We Come,” “Proposals for Go 1.15,” and related language change process materials.

— Review group、語言變更保守性與多數提案被拒的治理理由。

[R19] PLDST-023, Wirth、Ritchie 與 Stroustrup：簡潔、機器控制與相容性之間的三種系統語言倫理.

— 第四部前置比較研究。

[R20] PLDST-024, Guido、Matz 與 Larry Wall：可讀性、幸福與多義性之間的三種人本語言設計.

— 人本價值與治理風格連接。

[R21] PLDST-025, Backus、McCarthy 與 Hickey：函數、符號與簡單性的不同道路.

— 簡單性與權力准入的前置比較。

資料查核日期： 2026-07-30。

[G-STUDIO] individual coherent studio governance
[G-BDFL] central-taste deliberative governance
[G-PEP] documented proposal governance
[G-COUNCIL] elected steering council
[G-RFC] request-for-comments process
[G-FED] team federation
[G-LC] leadership council
[G-HYBRID] corporate-community hybrid
[G-ISO] standards committee
[G-DESIGNTEAM] small design-team governance
[P-A] agenda-setting authority
[P-P] proposal access
[P-D] decision authority
[P-I] implementation ownership
[P-R] release authority
[P-C] compatibility obligation
[P-S] succession
[P-T] traceability
[P-E] exit/fork
[R-C] coherence responsibility
[R-M] maintenance responsibility
[R-G] governance labor
[R-X] conflict handling
[R-B] burden allocation

SECTION

E.1 Python 現行治理不是 BDFL

截至查核日，PEP 13 明確規定 Python 由五人 Steering Council 治理。

本文將 Python 分期：

創始者設計
BDFL
PEP 制度

後 BDFL 過渡

Steering Council

沒有把歷史 BDFL 狀態誤寫為現行狀態。

SECTION

E.2 PEP Editor 不等於 PEP 裁決者

PEP 1 說明 Editor 主要檢查：

- 格式；
- 完整性；
- 結構；
- 行政要求。

  Editor 接受文件進入 PEP Repository，不表示接受功能設計。

SECTION

E.3 Steering Council 權力與克制

PEP 13 同時包含：

- 廣泛正式權力；
- 優先共識；
- 優先委派；
- 盡量少直接使用權力；
- 盡可能公開。

  本文沒有只寫「Council 擁有全部權力」，也沒有把它描述成純象徵角色。

SECTION

E.4 Rust RFC 不是直接民主

Rust 官方治理資料指出：

- 大型變更由 RFC 公開討論；
- 領域團隊對 Purview 責任；
- Team 可形成決定；
- Leadership Council 處理跨團隊與專案級協調。

  本文因此區分：

> PublicInput
> ≠
> FinalAuthority

SECTION

E.5 Rust Core Team 與 Leadership Council 的時間差

早期 RFC 使用 Core Team 語彙；RFC 3392 後由 Leadership Council 接替專案級領導功能，並將大部分權力委派給團隊。

本文保留歷史分期，沒有把早期文件的 Core Team 直接當成 2026 年現行組織圖。

SECTION

E.6 Swift 的公司影響是結構分析

本文沒有主張所有 Swift Evolution 決策由 Apple 私下決定。

第一手資料支持：

- Proposal 與 Review 公開；
- Language Steering Group 具有演化權威；
- Apple 仍提供重要工程、平台與發布資源。

  所以判定為：

SECTION

E.7 WG21 不是一般開源維護團隊

WG21 的正式產物是 ISO C++ 標準，不直接等同任一 Compiler Repository。

本文分開：

> StandardDecision
> ≠
> CompilerMerge
> ≠
> ReleaseAvailability

SECTION

E.8 共識與投票

Rust、Python、Swift、WG21 都重視討論或共識，但「共識」的制度含義不同。

本文沒有把它們合併為單一 Voting model。

SECTION

E.9 個人設計者不是單人完成所有成果

Oberon 由 Wirth 與 Gutknecht 等協作者共同完成語言與系統。

本文以「個人設計者工作室」描述高度集中且小型的設計核心，不把全部成果錯歸單人。

SECTION

E.10 Go 作為補充混合案例

Go 不屬於標題中的三個純原型。

本文加入 Go，是為展示：

小型設計團隊
+
公開 Proposal
+
高拒絕率
+
相容保守
如何形成第四種實務組合。

SECTION

E.11 治理分類不是價值排名

下列分類不是道德排序：

Studio
BDFL
RFC federation
Steering hybrid
Standards committee
每種制度只在特定規模、風險與責任條件下成立。

第四部共四篇：

- PLDST-023：Wirth、Ritchie 與 Stroustrup；
- PLDST-024：Guido、Matz 與 Larry Wall；
- PLDST-025：Backus、McCarthy 與 Hickey；
- PLDST-026：個人設計者、BDFL 與 RFC 制度。

  第四部完成後，PLDST 已建立四類跨案例能力：

  技術現實比較
  人本價值比較
  簡單性比較
  治理比較
  第五部將把這些結果轉成可重複執行的方法。

  下一篇預定為：

  PLDST-027：PLDST 評估矩陣與設計決策語料庫規格。