Niklaus Wirth：簡潔、教育與可實作性的設計倫理

Niklaus Wirth: Simplicity, Education, and the Ethics of Implementability

v1.0 / 2026-07-30 / Neo.K / 公開版／第三部設計師個案正式研究

https://thisoneisneok.com/html/papers/pldst-014.html

英文名稱： Niklaus Wirth: Simplicity, Education, and the Ethics of Implementability

系列： Programming Language Designer Style Taxonomy（PLDST）

文件編號： PLDST-014

版本： v1.0

日期： 2026-07-30

作者： Neo.K

文件狀態： 公開版／第三部設計師個案正式研究

SECTION

摘要

Niklaus Wirth 常被簡化為「Pascal 的發明者」「極簡主義語言設計者」或「軟體膨脹的批評者」。這些標籤只描述了結果，卻未說明其極簡從何而來，也容易將他誤寫成反對所有功能與大型系統的保守派。

Wirth 實際設計了一條持續修正的語言—系統鏈：

> Euler
> →
> ALGOL W
> →
> Pascal
> →

Pascal 將結構化程式設計與資料結構帶入教學；Modula／Modula-2 回應並行、模組、獨立編譯與系統程式需求；Oberon 則在對 Cedar 等大型軟體系統的實際經驗後，嘗試提高 Modula-2 的能力並同時減少複雜度，以少量型別延伸和模組機制支撐完整工作站系統。[R1][R2][R3]

這表示 Wirth 的核心風格不是「永遠保留原語言」，而是：

> 當既有語言的概念邊界不再適合新問題時，寧可建立更小、更一致、可完整實作的新系統，也不願在舊核心上無限堆疊補丁。

Wirth 的教育觀同樣不是把語言變成只適合學生的玩具。他把程式設計視為：

- 將模糊任務逐步精化；
- 同時設計演算法與資料結構；
- 以型別及結構表達不變量；
- 將程式、語言、編譯器和機器共同理解；
- 透過完整實作驗證設計是否真的清楚。

其著名的 Stepwise refinement 強調把設計決策逐步分解，盡量解除彼此交纏；《Algorithms + Data Structures = Programs》則把資料表示與算法視為共同決定程式結構的兩部分。[R4][R5] Project Oberon 更把語言、編譯器、作業系統、圖形環境及後來的 RISC5 處理器放在同一可閱讀、可教學、可重建的工程中。[R3][R6]

本文以 PLDST 方法把 Wirth 的生涯分為六個相位：

- ALGOL 概念提煉期：Euler 與 ALGOL W；
- 結構化教育期：Pascal、資料結構及逐步精化；
- 模組與系統程式期：Modula、Modula-2、Lilith；
- 整體可理解系統期：Oberon、Ceres 與 Project Oberon；
- 軟體膨脹批判期：Lean Software；
- 語言—編譯器—硬體再統一期：Oberon-07、RISC5、Lola 及晚期教材。

本文核心判斷為：

> 【Wirth 的簡潔不是功能數量崇拜，
>
> 而是要求整個計算系統仍可被人完整理解、實作和教授。】

其設計倫理可以表示為：

> 【概念必須清楚
> +
> 語言必須可實作
> +
> 實作必須可閱讀
> +
> 系統必須能被重建】

這種風格的代價也很明確：

- 願意捨棄向上相容；
- 可能低估大型生態的異質需求；
- 有時把完整性和便利性留給外部實作者；
- 小型研究團隊可掌握的整體性，不一定能直接擴展到全球產業；
- Pascal、Modula-2 及 Oberon 的實際生態又常超出 Wirth 本人的設計與治理。

關鍵詞： Niklaus Wirth、Pascal、Modula-2、Oberon、結構化程式設計、逐步精化、Compiler Construction、Project Oberon、Lean Software、PLDST

SECTION

一、本文研究範圍

本文主要分析：

- Euler；
- ALGOL W；

- Pascal；
- Stepwise refinement；
- Modula／Modula-2；
- Lilith；
- Oberon；
- Project Oberon；
- Ceres／RISC；
- A Plea for Lean Software；
- Oberon-07、RISC5 與晚期 Compiler Construction。

本文不把下列內容全部歸入 Wirth：

- UCSD Pascal；
- Turbo Pascal；
- Object Pascal／Delphi；
- ISO Pascal；
- ISO Modula-2；
- Oberon-2；
- Component Pascal；
- 所有後代方言及編譯器。

SECTION

二、語言創始與系統共同建造需要分離

Wirth 對 Pascal、Modula-2、Oberon 的語言核心具有高度個人設計權重，但其系統工作仍包含重要共同作者和團隊：

- Pascal 的早期編譯器、Pascal-P 可攜實作與公共報告涉及不同協作者；其中 Urs Ammann 與可攜編譯工作密切相關，Kathleen Jensen 是《Pascal User Manual and Report》共同作者，其他移植及編譯專案亦由 ETH 成員參與；
- Lilith 是最多約七名助手參與的整合硬體—軟體計畫；
- Project Oberon 由 Wirth 與 Jürg Gutknecht 共同完成；
- Oberon-2 與 Hanspeter Mössenböck 等後續研究相關；
- 工作站、板卡及控制系統亦有硬體共同設計者和實作者。[R2][R3][R7]

因此：

核心語言取捨：Wirth 高
編譯器與教材：Wirth 高，但有共同作者
完整系統：Wirth＋Gutknecht／研究團隊
後代生態：其他機構與社群

SECTION

三、Turing Award 的歷史定位

ACM 於 1984 年授予 Wirth 圖靈獎，表彰他發展 Euler、ALGOL W、Modula、Pascal 等一系列創新語言，以及 Pascal 對教育和後續語言、系統、架構研究的基礎作用。[R8]

值得注意的是：

- 獎項強調的是「一系列語言」；
- Wirth 的風格應從連續修正中辨識；
- 不能只以 Pascal 一個成功作品定義其一生。

SECTION

四、Euler：尋找基本且一般的語言概念

ETH 專案回顧將 Euler 描述為：

- 從 ALGOL 60 中辨識並組合本質概念；

- 探索語法分析與語義解釋的系統連接；
- 建立高階語言實作方法；
- 研究 Stack-oriented interpreter。[R2]

這顯示 Wirth 的早期問題不是單純增加語法，而是：

> 哪些概念是語言真正需要的，如何使它們有清楚語義並可有效實作？

SECTION

五、ALGOL W：標準爭議中的個人設計

ALGOL W 起源於對 ALGOL 後繼設計的分歧。

Wirth 與 C. A. R. Hoare 提出較克制、可實作的方向，但 ALGOL 68 委員會採取更廣泛設計。Wirth 隨後推進自己的 ALGOL W。

這形成其治理風格的一個早期特徵：

若委員會折衷無法形成一致系統
→
建立一個可完整實作的替代語言

SECTION

六、設計者的責任不止寫規格

Wirth 不滿足於：

- 語法建議；
- 委員會報告；
- 抽象功能清單。

他要求：

- 完整定義；
- Compiler；

- 教材；
- 真實使用；
- 由實作反饋設計。

這一原則貫穿 Pascal 到 Oberon。

SECTION

七、Pascal 的原始定位

Pascal 約於 1970 年形成，反映：

- Structured programming；
- 明確控制結構；
- 靜態型別；
- Record、Set、Array、Pointer 等資料結構；
- 可由學生及實作者理解的語言核心。[R1][R9]

它的教育目標不是：

讓初學者最快寫出任何程式

而是：

讓學生學習如何系統性建構算法與資料

SECTION

八、教學語言不等於不實用

Pascal 後來廣泛用於：

- 大學課程；
- 編譯器；
- 作業系統；

- 個人電腦；
- 軟體工具；
- 出版及研究。

但 Wirth 的核心評價標準仍是：

- 是否支持良好方法；
- 是否讓語言本身不鼓勵混亂；
- 是否能建立可教學 Compiler；
- 是否使程式結構與思考結構一致。

SECTION

九、結構化控制

Pascal 偏好：

- if；
- case；
- while；
- repeat；
- for；
- Procedure／Function；

而非任意 goto 驅動的控制圖。

這將控制責任從：

程式設計者手工維護任意跳轉

改為：

語言提供具有局部入口和出口的控制構造

SECTION

十、資料結構是語言核心

Wirth 的設計不只追求控制結構，也重視：

- Type；
- Array；
- Record；
- Set；
- Pointer；
- Enumeration；
- Subrange。

《Algorithms + Data Structures = Programs》的含義不是簡單加法公式，而是：

> ProgramStructure
> =
> f(
> Algorithm,
> DataRepresentation
>)

資料設計不是演算法完成後的附加工作。

SECTION

十一、從任務到可執行細節

Stepwise refinement 將程式開發理解為：

粗略任務
→
分解子任務

→
選擇表示
→
逐步加入細節
→
形成可執行程式

每一步應：

- 保持先前目的；
- 引入有限新決策；
- 避免同時處理所有層級；
- 盡量解除互相交纏的設計面向。[R4]

SECTION

十二、精化不只是 Top-down 口號

Wirth 強調：

- 資料和操作相互影響；
- 某些決策需要返回修正；
- 表示選擇會改變算法；
- 每一步應能被人理解和檢查。

因此它不是機械樹狀分解，而是一種控制認知複雜度的方法。

SECTION

十三、責任配置

逐步精化把正確性責任放在：

- 設計過程；
- 每一層抽象；

- 清楚的不變量；
- 可追蹤決策。

它不假設 Compiler 能自動推導全部正確程式，而要求人類將問題以可審查階段轉化。

SECTION

十四、教育倫理

其教育立場可以表示為：

> 不要只教會學生操作語法；應教會他們如何將問題逐步轉成結構清楚的演算法和資料。
> 語言是思考訓練工具，而不只是市場技能。

SECTION

十五、Pascal 的不足

Pascal 的原始核心適合結構化程式與教學，但大型系統需要：

- 獨立編譯；
- Module；
- 明確介面；
- 隱藏實作；
- 低階硬體存取；
- 並行與同步；
- 分離式開發。

Wirth 沒有持續把所有功能塞入 Pascal，而是建立新語言系列。

SECTION

十六、Modula：並行作為問題來源

Modula 約自 1973 年起探索：

- Concurrent process；
- Process generation；
- Synchronization；
- 最小支援語言；
- PDP-11 上的實際系統。[R2]

這顯示其極簡不是拒絕並行，而是：

> 只有在問題真正需要時，才將一組具有清楚模型的機制納入語言。

SECTION

十七、Modula-2：模組成為第一級構造

Modula-2 增加：

- Definition module；
- Implementation module；
- Separate compilation；
- Information hiding；
- System programming facilities；
- Coroutine／Process support。

它把大型系統責任由非正式文件轉入語言結構。

SECTION

十八、語言與工作站共同設計

Lilith 計畫整合：

- Hardware；
- Microcode；
- Operating system；
- Modula-2 Compiler；
- Editor；
- Application。

  這讓語言機制接受完整系統壓力測試。

  Wirth 的「可實作」不是 Compiler 能跑即可，而是：

> Language
> +
> OS
> +
> Tools
> +
> Hardware

能否形成可靠整體。

SECTION

十九、Cedar 的反面教材

Wirth 在 Xerox PARC 使用 Cedar 後，認為其雖具先進概念，卻呈現大型群體軟體的問題：

- 體積龐大；
- 結構交纏；
- 難以理解；

- 可靠性不足；
- 完整系統無人能掌握。[R2]

因此 Oberon 的目標不是建立另一個功能更完整的 Cedar，而是：

> 保留重要觀念，但讓系統小到仍能被完整理解、記載和教授。

SECTION

二十、先刪後加

Wirth 在 Modula-2／Oberon 回顧中說明，設計 Oberon 時先決定：

- 哪些 Modula-2 功能可以刪除；
- 哪些是真正缺少的基本能力。

最重要新增是：

- Type extension。

同時移除或簡化多種被視為次要或交互成本高的機制。[R1]

這是 刪減式極簡：

> Poweruparrow
>
> FeatureCountdownarrow

理想上可以同時成立。

SECTION

二十一、型別延伸而非完整類別系統

Oberon 使用：

- Record；

- Pointer；
- Type extension；
- Type test；
- Procedure；
- Module。

  建立可擴展資料抽象，而沒有直接採用：

- 完整類別語法；
- 多重繼承；
- Overload；
- 大型 Metaobject system。

  這是高覆蓋機制優先於功能清單的典型案例。

SECTION

二十二、單一處理程序與事件系統

早期 Oberon System 採相對簡單的執行模型，仍能支撐：

- 編輯器；
- 網路；
- 檔案；
- 圖形；
- 印表；
- 郵件；
- 互動命令。

這展示 Wirth 偏好先使用最簡單可行模型，再以清楚組合擴展。

但它不表示單一處理程序適合所有現代多核心和分散式工作負載。

SECTION

二十三、可重建性

Project Oberon 的核心目標是：

- 從零設計；
- 完整實作；
- 以書籍逐層解釋；
- 讓讀者取得原始碼；
- 使系統能在教學中重建。[R3][R6]

  這把文件責任提升到設計本體。

SECTION

二十四、編譯器必須能被完整閱讀

Wirth 的 Compiler Construction 以精簡語言與 RISC 目標機器說明：

- Scanner；
- Parser；
- Symbol table；
- Type checking；
- Code generation；
- Loader；
- Target architecture。

編譯器不是黑箱，而是教育與可信性的核心。

SECTION

二十五、作業系統不是語言外部世界

Oberon 把：

- Module loader；
- File；
- Text；
- Viewer；
- Command；
- Input；
- Display；

  用同一語言和設計原則建構。

  因此：

> LanguageQuality
> 由完整系統驗證

而非只以小型範例驗證。

SECTION

二十六、與 Jürg Gutknecht 的共同成果

Project Oberon 由 Wirth 和 Gutknecht 共同完成。

一般可粗略區分：

- Wirth：語言、Compiler、部分系統與後來硬體；

- Gutknecht：作業系統、顯示與系統結構的重要共同設計；
- 具體模組仍需依原始文件逐項歸因。

不能把 Project Oberon 簡化成 Wirth 單人作品。

SECTION

二十七、軟體為何膨脹

〈A Plea for Lean Software〉批評：

- 硬體成長掩蓋軟體低效率；
- 功能堆疊超過實際需要；
- 使用者難以分辨必要與裝飾；
- 大型工具鼓勵更大型軟體；
- 複雜系統難以可靠理解。[R10]

SECTION

二十八、「Wirth 定律」的歸因邊界

「軟體變慢的速度快於硬體變快」常被稱為 Wirth's Law。

但 Wirth 在文中將相關觀察歸於 Martin Reiser 的 Oberon System 前言。較準確的寫法是：

- Wirth 普及並系統論證該問題；
- 該句式不應被當成無來源的 Wirth 單人原創定律。

SECTION

二十九、極簡不等於回到低階

Wirth 並不主張：

- 捨棄高階語言；
- 手寫組合語言；
- 不使用圖形；
- 不建立完整系統。

相反地，他主張以：

- 更好語言；
- 模組；
- GC 或自動管理的適當使用；
- 清楚工具；
- 簡潔系統；

提高生產力。

批判對象是無法以功能收益解釋的體積與交纏。

SECTION

三十、可理解性是一種安全

當系統大到無人理解：

- 漏洞難發現；
- 修改不可預測；
- 故障難定位；
- 知識只能依賴少數人；
- 驗證成本爆炸。

所以：

> Understandability
> ⊂
> SystemSafety

至少是安全的必要工程條件之一。

SECTION

三十一、RISC 與 Oberon

Wirth 後期以 FPGA 建立 RISC5，並更新 Project Oberon，使：

- 處理器；
- Compiler；
- Operating system；
- Application；

再次形成完整教學鏈。

這不是追求市場最強性能，而是使所有層級可閱讀和實驗。

SECTION

三十二、Lola

Lola 是硬體描述語言，延續 Wirth 的方法：

- 少量語言構造；
- 清楚結構；
- 可編譯；
- 與實際 FPGA 連接；
- 作為教育工具。

他把語言設計倫理從 Software 延伸至 Digital circuit。

SECTION

三十三、Oberon-07

Oberon-07 再次修訂 Oberon：

- 精簡語言；
- 調整型別、迴圈及實作規則；
- 與新 Compiler 和 RISC5 共同驗證；
- 維持簡潔 Reference report。[R9]

  這表明 Wirth 並未把 1988 Oberon 化石化，而是持續修改。

SECTION

三十四、為什麼稱為「設計倫理」

Wirth 的方法隱含一個責任主張：

> 設計者不能只把功能交給使用者，再把所有理解與實作成本推給其他人。

語言設計者應能：

- 解釋核心；
- 寫出 Compiler；
- 展示 Runtime；
- 建立真實系統；
- 提供教材；
- 說明功能的必要性。

SECTION

三十五、功能的舉證責任

在 Wirth 風格中：

> BurdenOfProof(feature)
> >
> BurdenOfProof(omission)

新增功能必須證明：

- 不只是便利；
- 無法由現有機制清楚表達；
- 不會造成過度交互；
- 能被實作與教授；
- 對完整系統有實際價值。

SECTION

三十六、整體理解的門檻

Wirth 不要求每位一般使用者都理解所有晶片細節，但他希望：

- 至少有人能從語言追到機器；
- 教材可以建立完整因果鏈；
- 系統不依賴無人能說明的魔法層；
- 實作與規格相互校驗。

SECTION

三十七、複雜度配置

Wirth 傾向降低：

- 核心語言功能；
- 特徵交互；
- Runtime 魔法；
- 工具鏈依賴；
- 不可理解框架；
- 過度向後相容。

將必要複雜度放入：

- 程式結構；
- Module；
- Type；
- 明確資料表示；
- 可讀 Compiler；
- 可理解系統。

SECTION

三十八、責任配置

語言與 Compiler 負責：

- 靜態檢查；
- 結構化控制；
- 模組邊界；
- 型別；

- 清楚翻譯。

  程式設計者負責：

- 分解；
- 資料表示；
- Algorithm；
- Module design；
- 逐步精化。

  Wirth 不追求讓 Compiler 猜完一切，而是以語言約束協助人類形成可驗證設計。

SECTION

三十九、安全—自由配置

Wirth 偏好：

- Safe structured subset；
- 強型別；
- 清楚 Pointer；
- 模組封裝；
- 受控低階 SYSTEM 能力。

  低階能力存在，但應集中並與一般語言分離。

SECTION

四十、相容性配置

他願意：

- 以 Modula-2 修正 Pascal；

- 以 Oberon 修正 Modula-2；
- 以 Oberon-07 再修正 Oberon。

  因此：

> Coherence
> >
> UpwardCompatibility

在創建新語言時尤其明顯。

這與 C++、Java 或 Web 語言的長期相容制度不同。

SECTION

四十一、Euler／ALGOL W 期

目標：提煉 ALGOL 的基本概念並證明可實作

SECTION

四十二、Pascal 期

目標：讓結構化程式和資料結構可被系統教學

SECTION

四十三、Modula-2／Lilith 期

目標：將語言擴展到模組化系統與硬體控制

SECTION

四十四、Oberon／Ceres 期

目標：建立完整、可理解、可延展工作站系統

SECTION

四十五、Lean Software 期

目標：反駁以硬體成長合理化軟體膨脹

SECTION

四十六、RISC5／Oberon-07 期

目標：重新統一語言、Compiler、OS 與處理器教育

SECTION

四十七、問題 framing

Wirth 的核心問題是：

> 如何使程式、語言與系統的結構清楚到人能逐步建造、完整理解並可靠實作？

SECTION

四十八、價值優先序

> V_(Wirth)
> ≈
> (
> Simplicity,
> Structure,
> Understandability,
> Implementability,
> Education,
> Efficiency,
> WholeSystemIntegrity
>)

SECTION

四十九、核心—擴張偏好

偏好：

- 小核心；
- 正交機制；
- Module；
- Type extension；
- 明確系統邊界；
- 以新語言修正舊核心。

不偏好：

- 功能聯合體；
- 大量語法糖；
- 為相容保留所有歷史；
- Runtime 黑箱。

SECTION

五十、顯式—推導偏好

偏好明示：

- Type；
- Module；
- Data structure；
- Control；
- Low-level boundary。

自動化主要交給 Compiler：

- 解析；

- 型別檢查；
- Code generation；
- 部分配置。

SECTION

五十一、效率—可讀性偏好

希望：

- Source 結構直接對應清楚實作；
- Compiler 小而有效；
- Hardware model 可理解；
- 性能不是依巨大最佳化黑箱獲得。

但這不代表拒絕 Compiler 最佳化，而是拒絕以不可理解性作為性能前提。

SECTION

五十二、治理偏好

Wirth 適合：

- 小型專業團隊；
- 集中設計；
- 完整重建；
- 快速修正；
- 教學與研究。

較不適合：

- 全球多方委員會；

- 永久向後相容；
- 大型異質生態需求整合。

SECTION

五十三、Pascal 的缺陷不能忽略

原始 Pascal 在實務中曾受到批評，例如：

- 字串；
- 分離編譯；
- I/O；
- 陣列邊界參數；
- 系統程式能力；
- 互操作。

Modula-2 的出現本身就是 Wirth 對部分不足的回答。

SECTION

五十四、小型設計可能轉移生態成本

若核心不提供常見能力，使用者可能建立：

- 方言；
- 非標準 Library；
- Vendor extension；
- 不相容 Compiler；
- 外部 Framework。

Pascal 及 Modula-2 的多方言歷史證明，小核心不能自動保證小生態。

SECTION

五十五、完整可理解系統難以全球化

Project Oberon 在小團隊、單一工作站與教學範圍內極具說服力。

但全球系統還需要：

- 國際文字；
- Accessibility；
- 安全隔離；
- 多核心；
- 分散式；
- 大型 Driver；
- 供應鏈；
- 多組織治理。

整體可理解性與完整功能之間仍有張力。

SECTION

五十六、教育價值不等於普遍產業最優

一種語言非常適合教導：

- 資料結構；
- Compiler；
- Module；
- 系統；

不表示它在所有商業生態中都優於大型平台語言。

SECTION

五十七、Lean Software 可能低估需求異質性

某些軟體體積來自：

- Accessibility；
- Localization；
- 安全；
- 向後相容；
- 多硬體；
- 法規；
- 可觀察性；
- 雲端容錯。

不能把所有大型軟體都歸為無意義肥胖。

SECTION

五十八、Wirth 不是所有工作之單一作者

Oberon System、Lilith、Pascal Compiler、教材和後續語言均有共同作者及實作者。PLDST 必須在每項決策上重新歸因。

時期	問題	決策	複雜度去向	風格
1960s	ALGOL 概念過多且規格分歧	Euler／ALGOL W	Compiler／形式語義	概念提煉
1970	缺乏結構化教學語言	Pascal	Type／control structure	教育結構
1971	程式開發決策交纏	Stepwise refinement	分階段設計	方法紀律
1973–79	Pascal 不足以支援系統	Modula／Modula-2	Module／SYSTEM	問題驅動擴張

1977–81 語言需經完整硬體驗證 Lilith 全棧共同設計 可實作性

1986–90 Cedar 過大且難理解 Oberon 刪減＋Type extension 整體簡潔

1995 軟體膨脹被硬體掩蓋 Lean Software 功能舉證責任 反膨脹

2007–2013+ 教學鏈再次分裂 Oberon-07／RISC5 語言—硬體統一 可重建性

SECTION

五十九、主要原型

Niklaus Wirth 同時屬於：

- 概念刪減式語言建築師；
- 結構化教育設計者；
- 完整實作驗證者；
- 語言—系統—硬體整合者；
- 軟體膨脹批判者。

SECTION

六十、不適合的簡單標籤

不應只稱：

Pascal 發明者
極簡派
反功能派
教學語言設計者
Wirth 定律提出者

較精確的描述是：

> 把可理解性視為語言與系統的設計責任，並反覆以新語言和完整實作檢驗哪些概念真正必要的工程教育者。

SECTION

六十一、最重要的連續性

Euler 到 Oberon-07 的共同規則是：

> 【以更少、更正交的概念
>
> 覆蓋當前真正需要的問題】

SECTION

六十二、最重要的不連續性

各語言使命不同：

- Pascal：教學與結構化程式；
- Modula-2：模組化系統；
- Oberon：可延展完整工作站；
- Lola：Digital circuit；
- PICL：Microcontroller。

  Wirth 的風格穩定，不代表功能集合固定。

SECTION

六十三、最重要的倫理判斷

一個語言若需要：

- 無法解釋的 Compiler；
- 無人能完整理解的 Runtime；
- 無限歷史補丁；

- 僅因硬體便宜而存在的膨脹；

  在 Wirth 看來就不只是技術醜陋，而是設計者逃避責任。

  Niklaus Wirth 的程式語言設計不是單純做減法。

  他的真正方法是：

- 先明確定義要解決的問題；
- 從已有語言中保留必要概念；
- 刪除不具足夠覆蓋率的機制；
- 加入少量能改變整體能力的新構造；
- 親自或與小團隊建立 Compiler；
- 用真實作業系統、工具與硬體驗證；
- 將結果寫成可教、可閱讀及可重建的材料；
- 當新問題出現時，允許新語言取代舊核心。

  本文對 Wirth 的 PLDST 判定為：

> 【Conceptual Reductionist
> →
> Educational Systems Engineer
> →
> Whole-System Implementability Critic】

他的核心優勢是：

- 語言概念與教學方法一致；
- 功能需通過完整實作測試；
- 模組與型別承擔結構責任；
- Compiler 和 OS 可被閱讀；

- 願意修正自己的舊語言；
- 將可理解性視為性能、可靠與教育的共同地基。

  其主要代價是：

- 低相容容忍度；
- 小團隊模型難以直接擴張；
- 可能低估生態和多樣需求；
- 完整系統示範與現代全球平台仍有距離；
- 使用者可能以方言和 Vendor extension 補回被刪除能力。

  最終原則為：

> 【簡潔不是功能貧困
>
> ∧
> 教育不是降低要求
>
> ∧
> 可實作不是只要編譯成功】

Wirth 所要求的是一種更嚴格的設計責任：

> 語言提出的每一個概念，都應能被清楚定義、有效編譯、完整解釋，並在真實系統中證明自己值得長期存在。

人物：Niklaus Wirth
主要語言／系統：Euler、ALGOL W、Pascal、Modula-2、Oberon、Lola
核心時期：1960s–2010s
主要問題：語言與系統複雜度超過人類可理解範圍
主要策略：結構化、型別、模組、刪減、完整實作
複雜度去向：清楚程式結構、Compiler 與可讀系統
責任去向：設計者需證明功能可教、可實作
主要保護對象：學習者、實作者、系統維護者
主要限制：相容、生態、現代平台規模與異質需求
歸因信心：高

[R1] Niklaus Wirth, “Modula-2 and Oberon,” HOPL III, revised 2006.

— Pascal、Modula-2、Oberon 的設計演化、先刪後增、Type extension、硬體與系統背景。

[R2] Niklaus Wirth, “Summary of Projects, 1962–1999,” ETH Zürich.

— Euler、Modula、Lilith、Ceres、Oberon、教學及硬體計畫的時間線與團隊背景。

[R3] Niklaus Wirth and Jürg Gutknecht, Project Oberon: The Design of an Operating System, a Compiler, and a Computer, 1992／2005／2013.

— 完整工作站系統、Compiler、OS、工具及後期 RISC5。

[R4] Niklaus Wirth, “Program Development by Stepwise Refinement,” Communications of the ACM 14(4), 1971.

— 分解設計決策、逐步精化及程式建構方法。

[R5] Niklaus Wirth, Algorithms + Data Structures = Programs, 1976; later Oberon edition.

— 算法、資料結構、程式表示及系統化教材。

[R6] Niklaus Wirth, Compiler Construction, ETH Zürich editions.

— Scanner、Parser、Type checking、Code generation、RISC 目標及可閱讀 Compiler。

[R7] ACM Turing Award Oral History Interview with Niklaus Wirth, 2018; ETH historical records.

— Pascal、個人電腦、語言標準爭議、教育與系統工作的後期回顧。

[R8] ACM, Niklaus Wirth A.M. Turing Award materials, 1984.

— Euler、ALGOL W、Modula、Pascal 及其教育、系統與架構影響。

[R9] Niklaus Wirth, “The Programming Language Pascal,” 1971; “The Programming Language Oberon／Oberon-07,” ETH reports.

— 語言正式定義、資料型別、結構化控制與 Type extension。

[R10] Niklaus Wirth, “A Plea for Lean Software,” Computer 28(2), 1995.

— 軟體膨脹、硬體掩護、Oberon 經驗與簡潔工程。

[R11] Kathleen Jensen and Niklaus Wirth, Pascal User Manual and Report, 1974.

— Pascal 的公共定義、教材與共同作者歸因。

[T-A] ALGOL concept-refinement phase
[T-P] Pascal educational phase
[T-M] Modula systems phase
[T-O] Oberon whole-system phase
[T-L] Lean-software critique phase
[T-H] Language–hardware reunification phase
[S-R] Conceptual reduction
[S-E] Educational structure
[S-I] Implementability
[S-W] Whole-system understanding
[S-C] Compatibility sacrifice
[S-L] Lean-software discipline

附錄 D　第二輪史實與歸因校對紀錄
D.1 語言系列與時間
第二輪重新核對 ACM、ETH 專案史及 Wirth 的 HOPL 回顧：
Euler：1960s 初

ALGOL W：1960s 中期

Pascal：1970

Modula：1973–1976

Modula-2：1977–1980，定義報告 1979 前後

Oberon：1986–1988 形成

Oberon-07：2000s 後期起持續修訂

本文使用的是思想及工程相位，不把一個出版年份當作唯一設計日期。

D.2 Pascal 的作者與編譯器歸因
已重新檢查 ETH 專案史與 Wirth 的 Pascal 回顧：
- Pascal 語言的核心設計主要歸於 Wirth；
- 第一批 Compiler、後來 Pascal-P／P-code、移植和公共文件由 ETH 多位成員共同推進；
- Kathleen Jensen 的明確角色是《Pascal User Manual and Report》共同作者，不應籠統寫成第一 Compiler 實作者；
- Pascal 的廣泛流通又受到 UCSD Pascal、微型電腦及外部廠商重大影響。
本文因此把：
語言核心

編譯器工程

公共規格

可攜移植

後期生態

分開歸因。

D.3 Modula-2 與向上相容

ETH 專案史及 Wirth 原文明確說明：

- Modula-2 被視為 Pascal 的更新版本；
- Wirth 願意犧牲 Upward compatibility，以避免 Pascal 的缺陷；
- 主要新增包括 Separate-compilation module、Coroutine 及封裝 Machine-specific object 的 System facility；
- 他後來承認自己曾天真地以為熟悉 Pascal 的使用者會自然接受轉換。

本文因此沒有將低相容偏好描述成零成本美德，也將 Modula-2 未取得 Pascal 同等普及視為反例。

D.4 Oberon 的「先刪後加」

第二輪直接核對〈Modula-2 and Oberon〉：

- Wirth 明確把 Simplicity of design 稱為最重要 Guiding principle；
- 其後果包括概念清楚、功能經濟、實作效率與可靠；
- Oberon 策略是先決定從 Modula-2 省略什麼，再決定需要增加什麼；
- 其目標是提高能力並同時降低複雜度；
- Type extension 是核心新增能力。

本文的「刪減式極簡」是對原始設計程序的結構化名稱，而非後人僅根據功能數量貼標籤。

D.5 Project Oberon 的共同作者

已重新核對 2013 Revised Edition Preface：

- Project Oberon 作者為 Niklaus Wirth 與 Jürg Gutknecht；
- 1986–1989 年間兩人不只構想，也編寫整個書中系統及更多部分；
- 主要目標是讓系統可被整體描述、解釋與理解；
- Program listing 被視為最終解釋的重要部分；
- Oberon 同時被用作 Implementation vehicle 與 Algorithm publication medium。

本文因此不將 Oberon System 寫成 Wirth 單人作業系統。

D.6 Lilith 與小團隊效應

ETH 專案史記錄：

- Lilith 是硬體、Microcode、OS、Compiler 和應用整合工程；
- 最多約七名助手的密集工作使其在三年內完成；
- Wirth 自己編寫部分 Editor 及工具；
- 具體處理器、Compiler 和應用另有共同設計者。

本文把「小團隊促進概念一致」視為該專案情境，不把它推廣成所有大型組織必然無法設計好系統。

D.7 Cedar 批判的範圍

Wirth 的 ETH 專案史明確把 Cedar 描述為先進但在日常使用中顯得：

- Bulky；
- Unreliable；
- 結構交纏；
- 難以整體理解。

本文保留這是 Wirth 的使用經驗和設計動機，不把它升格為對 Cedar 所有技術貢獻的客觀否定。

D.8 「Wirth 定律」與 Martin Reiser

第二輪核對〈A Plea for Lean Software〉：

- Wirth 以軟體體積及遲鈍速度超過硬體進展的問題為文章核心；
- 文章相關引文和參考將著名觀察連到 Martin Reiser 的 Oberon System；
- 後來「Wirth's law」是對 Wirth 普及、系統論述該問題的命名；
- 本文因此沒有把精確句子寫成 Wirth 無前例的單人發明。

D.9 高階工具與簡潔

第二輪核對 Lean Software 原文後確認：

- Wirth 的方案不是返回 Machine code；
- 他仍支持 High-level language、良好 Compiler、Structured methodology 和適當工具；
- 批判目標是功能和體積超出可證成收益，以及硬體進步掩護缺乏紀律；
- 本文因此避免將他分類為反抽象或反自動化。

D.10 Oberon-07、RISC5 與晚期工程

ETH 官方頁面保存：

- Oberon-07 Language report；
- Compiler；
- Project Oberon 2013；
- RISC5 Verilog；
- FPGA 工具；
- Lola；
- 完整 System module。

這支持「晚期重新統一語言—Compiler—OS—Hardware」的判定。

但具體版本和模組有後續修改者，本文只將 Wirth 主導的整合問題意識歸為其個人風格。

D.11 PLDST 推論邊界

下列名稱為本文分析原型，而非 Wirth 正式自稱：

概念刪減式語言建築師

結構化教育設計者

完整實作驗證者

語言—系統—硬體整合者

其證據來自跨語言重複決策，信心為高；「設計倫理」則是本文對責任配置的規範性詮釋，信心為中高。