Anders Hejlsberg：工具驅動設計、漸進型別與平台折衷

Anders Hejlsberg: Tool-Driven Design, Gradual Typing, and Platform Compromise

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-020.html

英文名稱： Anders Hejlsberg: Tool-Driven Design, Gradual Typing, and Platform Compromise

系列： Programming Language Designer Style Taxonomy（PLDST）

文件編號： PLDST-020

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

Anders Hejlsberg 橫跨四十多年的語言與工具工作，常被濃縮成一串產品名稱：

> Turbo Pascal
> →
> Delphi
> →
> C#
> →
> TypeScript

這種履歷式敘述容易掩蓋一個高度穩定的設計風格。Hejlsberg 並不主要以建立最純粹、最小或最激進的語言聞名；他的核心能力是把語言、Compiler、IDE、Runtime、Library、平台和遷移路徑設計成一個可以被真實開發者快速採用的整體。

Turbo Pascal 的突破不只是 Pascal 方言，而是把編輯器、極快 Compiler、錯誤定位和執行流程壓縮成低摩擦回饋循環。Delphi 將這種整合擴張到視覺元件、事件、資料庫、Native code 和 Rapid Application Development。C# 則在 Windows／.NET 轉型中，重新配置 C／C++ 的熟悉表面、Java 式 Managed runtime、Component software、Versioning、Metadata、Delegate、Property、Event、Generics、LINQ 和非同步能力。TypeScript 更進一步：它沒有要求 JavaScript 生態改用一門不相容的新語言，而是以可擦除的型別註記、Structural typing、Inference 和編輯器服務，建立「可逐步加入的靜態證據」。[R1][R2][R3][R4]

本文將 Hejlsberg 的設計生涯分成七個相位：

- 微型電腦 Compiler 期：Blue Label／Compas／PolyPascal 與資源限制下的快速 Compiler；
- Turbo Pascal 回饋循環期：Compiler、Editor、低價格和個人開發工具的整合；
- Delphi 元件平台期：Object Pascal、Visual design、Native performance 與 Component model；
- C#／.NET 平台重建期：Managed runtime、Metadata、Component-oriented language 和版本治理；
- LINQ 與語言—資料統一期：Lambda、Generic、Expression tree、Query comprehension 和型別推導；
- TypeScript 漸進建模期：尊重 JavaScript 語義，以 Structural type system 建模既有慣用法；
- TypeScript 7 原生工具鏈期：為超大型程式庫重寫 Compiler／Language service 基礎，將工具延遲本身視為語言規模問題。

  本文核心判斷為：

> 【Hejlsberg 的主要設計對象不是孤立語法，
>
> 而是開發者與整個平台之間的回饋迴路。】

其深層配置可以表示為：

> 【Familiar surface
> +
> Rich tooling

這種風格並不等於一味追求便利。Hejlsberg 的 C# 設計論述反覆關注：

- Versioning；
- Component boundaries；
- Checked exception 的可擴展性；
- Virtual／Override 的演化風險；
- Delegate 與 Event 的一級支援；
- Runtime／Language 分工；
- 可用性測試與設計品味。[R5]

  TypeScript 的官方設計目標更明確規定：

- 靜態識別可能錯誤；
- 為大型元件提供結構；
- 不對輸出加入 Runtime overhead；
- 發出 Clean、Idiomatic JavaScript；
- 保留 JavaScript Runtime 行為；
- 採用 Gradual／Structural type system；
- 不以完全 Sound type system 為設計目標；
- 不複製 Runtime type information；
- 不以平台或 Library 為特定目標。[R4]

  因此 TypeScript 並不是「C# 編譯到 JavaScript」。它的型別系統刻意接受 JavaScript 世界中的開放物件、Prototype、Duck typing、Callback、Dynamic property 和漸進遷移；型別資訊主要服務於錯誤檢查、導航、Completion、Refactoring 和人機協作，Runtime 仍是 JavaScript。

  這種設計的力量與代價同樣來自折衷。TypeScript 可以在不重寫程式的情況下逐步增加工具價值；但 any、Assertion、Declaration file、Structural compatibility 和刻意不健全規則意味著型別正確不是完整 Runtime 安全證明。C# 具有更強的 Managed contract，卻也必須承擔 CLR、Framework、ABI、版本及平台整體複雜度。

截至 2026 年 7 月，TypeScript 7 已以 Go 原生重寫 Compiler 和 Language service，官方報告典型完整建置可達 8–12 倍加速；但 7.0 尚未提供穩定 Programmatic API，部分 Vue、Angular、Svelte、MDX 等嵌入式工具鏈仍需與 TypeScript 6 並行。[R6] 這正好呈現 Hejlsberg 風格的最新階段：當語言服務延遲開始妨礙互動時，他不只微調型別規則，而會重新設計整個工具基礎，同時以 Compatibility package 和 Side-by-side transition 限制生態破壞。

因此，Hejlsberg 不應只被分類成「商業語言設計者」或「語法保守派」。更精確的判定是：

> 他以開發者回饋速度和平台採用為中心，反覆在更強型別、熟悉語法、Runtime 能力、工具可用性及既有生態之間建立可部署折衷。

關鍵詞： Anders Hejlsberg、Turbo Pascal、Delphi、C#、TypeScript、漸進型別、結構型別、工具鏈、IDE、平台、PLDST

SECTION

一、本文研究範圍

本文主要分析：

- Blue Label／Compas／PolyPascal；
- Turbo Pascal；
- Delphi；
- J++／WFC 作為過渡背景；
- C# 及 .NET；
- LINQ；
- TypeScript；
- TypeScript 原生 Go 重寫與 7.0。

本文不把以下成果全部歸於 Hejlsberg：

- Pascal；
- Borland IDE 的全部產品策略；

- Delphi 全部元件和後續版本；
- CLR；
- .NET Framework；
- C# 所有語言功能；
- LINQ 全部研究；
- TypeScript 全部型別功能；
- Visual Studio／VS Code；
- TypeScript 7 全部實作。

SECTION

二、個人權重與團隊權重

Hejlsberg 對以下事項具有高度個人權重：

- 早期 Pascal Compiler；
- Turbo Pascal Compiler 架構；
- Delphi 首期總體架構；
- C# Lead architect；
- TypeScript 原始問題及型別／工具方向；
- TypeScript 7 原生 Port 的倡議和架構參與。

但官方 C# 規格明確把 Principal inventors 列為：

- Anders Hejlsberg；
- Scott Wiltamuth；
- Peter Golde。[R2]

C# 與 TypeScript 長期又涉及：

- Mads Torgersen；
- Eric Lippert；
- Neal Gafter；
- Peter Hallam；
- Erik Meijer；
- Don Syme；
- Luke Hoban；
- Jonathan Turner；
- Ryan Cavanaugh；
- Daniel Rosenwasser；
- TypeScript Core team；
- ECMA／ISO；
- .NET Runtime／Library team；
- IDE 和生態作者。

因此：

Turbo Pascal Compiler 原型：Hejlsberg 極高
Turbo Pascal 產品／IDE：Borland 團隊
Delphi 架構：Hejlsberg 高，產品團隊共同
C# 核心：Hejlsberg、Wiltamuth、Golde 等
C# 後續：Microsoft 團隊＋ECMA／ISO＋社群
TypeScript 原始方向：Hejlsberg 高
TypeScript 實作／演化：開源 Core team
TypeScript 7：Hejlsberg倡議＋原生 Port 團隊

SECTION

三、資源限制塑造方法

Hejlsberg 的早期 Compiler 面對：

- 64KB 等級記憶體；
- 慢速磁碟；
- 微型電腦；
- 個人使用者；
- 不成熟 Toolchain。

Compiler 必須：

- 小；
- 快；
- 可在同一機器立即使用；
- 生成實際可執行程式；
- 提供低摩擦 Debug 循環。

SECTION

四、從教材 Compiler 到產品

其早期 Pascal Compiler 受到 Wirth 教材中小型 Compiler 的啟發，但 Hejlsberg 逐步建立：

- Blue Label Pascal；
- Compas Pascal；
- PolyPascal；
- 多平台版本。

這個歷史顯示，他的第一項設計能力不是發明新語法，而是把語言規格轉成極高可用性的實際工具。

SECTION

五、第一個深層風格：Feedback latency 是語言成本

令一次開發循環為：

> T_(loop)
> =
> T_(edit)
> +
> T_(compile)
> +
> T_(diagnose)
> +
> T_(run)

Hejlsberg 的工具設計反覆試圖最小化 T_(loop)。

語言若理論優秀，但每次修改等待太久，其可學習性和探索性都會降低。

SECTION

六、Turbo 不只表示生成程式快

Turbo Pascal 的突破包括：

- Compiler 快；
- 編輯器整合；
- 錯誤定位；
- Compile／Run 循環；
- 低價格；
- 單一 Package；

- 個人電腦可用。

  因此其實際設計單位是：

> PascalDialect
> +
> Compiler
> +
> Editor
> +
> RuntimeLibrary
> +
> Distribution

SECTION

七、開發者體驗先於完整性

Turbo Pascal 沒有先等待：

- 完整標準覆蓋；
- 巨大 Framework；
- 多層工具；
- Enterprise deployment。

  它先使個人能迅速建立可執行程式。

  這種「先完成完整回饋閉環，再擴張能力」的模式延續至 Delphi 和 TypeScript。

SECTION

八、Compiler speed 的認知作用

快速 Compiler 讓使用者：

- 更敢試驗；
- 更快發現錯誤；
- 以小步重構；
- 將語言當互動工具，而非批次提交系統。

  速度因此是一種教育及設計特徵。

SECTION

九、從語言產品到應用平台

Delphi 結合：

- Object Pascal；
- Visual form designer；
- Component palette；
- Event；
- Property；
- Native Compiler；
- Database；
- Windows API；
- IDE；
- Debugger。

其問題是：

> 如何讓 Windows GUI／Database 應用可以像組合元件一樣快速建造，又保留 Native performance 和靜態語言工具？

SECTION

十、設計時與執行時的連接

Delphi 元件具有：

- Property；
- Event；
- Design-time metadata；
- Serialization；
- Visual representation；
- Runtime behavior。

這預示 C#／.NET 對 Property、Event、Attribute、Metadata 和 Component 的重視。

SECTION

十一、視覺工具不是取消程式

Form designer 處理：

- Layout；
- Component placement；
- Property editing；
- Event wiring。

程式碼處理：

- Domain logic；
- Data；
- Algorithm；

- Custom behavior。

  工具將部分宣告性工作移出文字語法。

SECTION

十二、RAD 的風險

快速建立介面可能造成：

- 邏輯散落在 Event handler；
- Visual state 和 Code 耦合；
- Vendor platform 綁定；
- 大型應用架構不足；
- 元件市場品質差異。

  因此 Tool productivity 不等於 System design 自動良好。

SECTION

十三、C# 不是只為改善 Java

C# 的問題背景包括：

- Windows API 的複雜性；
- COM 的 Component／Versioning 問題；
- C++ 開發安全和生產力；
- Java／J++ 經驗；
- 需要新的 Managed platform；
- 多語言 Runtime；
- Metadata 和 Deployment。

SECTION

十四、語言與 CLR 分工

CLR 提供：

- Garbage collection；
- Verification；
- Metadata；
- JIT；
- Exception；
- Reflection；
- Common type system；
- Interlanguage boundary。

C# 提供：

- 熟悉的 C-family 表面；
- Static type；
- Class／Struct；
- Delegate；
- Property；
- Event；
- Attribute；
- Generics；
- Async 等後續能力。

SECTION

十五、Component-oriented 設計

Property、Event、Attribute 和 Delegate 不只是 Syntax sugar，而是把 Component 慣例寫入語言和 Metadata。

例如：

> Event
> =
> TypedPublisherContract
> +
> SubscriberList
> +
> ToolVisibility

它能被 IDE、Designer、Reflection 和 Framework 共同理解。

SECTION

十六、Simplexity

Hejlsberg 在 C# 訪談中用「Simplexity」說明：表面簡單的 Component 操作可能由 Runtime、Metadata 和 Tool 承擔內部複雜度。[R5]

這與 Ruby 的「Natural not simple」不同，但配置相似：

> UserSurfaceComplexitydownarrow
>
> PlatformComplexityuparrow

SECTION

十七、Checked exception 的拒絕

Hejlsberg 對 Java Checked exception 的批評包括：

- Versionability；
- API 演化；
- 大型程式 Catch／Rethrow；
- 開發者可能用空 Catch 逃避；
- Function composition 困難。[R5]

  C# 選擇：

- Exception type 不進入 Method signature 強制契約；
- 依文件、分析和慣例治理。

  這提高演化彈性，降低靜態 Error effect 可見性。

SECTION

十八、Virtual／Override 的版本策略

C# 要求 virtual 和 override 明示，新增 Base method 時減少意外改寫。

這反映 Hejlsberg 對 Library evolution 的關注：

> LocalConvenience
> <
> FutureVersionSafety

SECTION

十九、資料查詢的斷裂

在 LINQ 前，程式經常在：

- Object；

- SQL；
- XML；
- Collection；

  之間切換不同語言、型別和工具。

SECTION

二十、LINQ 的共同基礎

LINQ 由多項能力組合：

- Generics；
- Lambda；
- Extension method；
- Type inference；
- Anonymous type；
- Expression tree；
- Query syntax。

  Query syntax 不是直接把 SQL 寫入 C#，而是翻譯成一般方法組合。

SECTION

二十一、語言功能服務 Tooling

LINQ 讓 Compiler 和 IDE 能：

- 檢查名稱；
- 推導型別；
- Completion；

- Refactoring；
- 跨資料來源提供共同模型。

  它延續「讓工具理解領域操作」的風格。

SECTION

二十二、抽象洩漏

不同 Provider 仍可能具有：

- 不同可翻譯 Operation；
- 不同性能；
- Remote query；
- N+1；
- Runtime exception。

  語言統一表面不能消除資料源差異。

SECTION

二十三、問題不是替換 JavaScript

2012 年的 TypeScript 面對：

- JavaScript 已是 Web Runtime；
- 巨大既有程式和 Library；
- Prototype 與 Dynamic pattern；
- 大型應用工具不足；
- Refactoring 和 Navigation 困難。

  建立完全不同的新語言會失去生態。

SECTION

二十四、Strict superset 與 Clean JavaScript

TypeScript 原始承諾：

- JavaScript 程式可逐步成為 TypeScript；
- Type annotation 可擦除；
- 生成可讀 JavaScript；
- Runtime behavior 保持 JavaScript；
- 不強制 TypeScript Runtime。

  這是相容性優先的語言增量。

SECTION

二十五、Structural typing

TypeScript 根據物件 Shape 判斷相容，而非要求共同名義宣告。

這與 JavaScript Duck typing 對齊：

> Compatible(A,B)
> ⇐
> RequiredMembers(B)
> ⊆
> Members(A)

SECTION

二十六、Type inference

TypeScript 盡量從：

- Initializer；
- Return；
- Context；
- Generic call；
- Control flow；

  推導型別，避免把 Java／C# 式完整 Annotation 樣板帶入 JavaScript。

SECTION

二十七、Contextual typing

Callback parameter 可以由使用位置反向取得型別。

這使既有 JavaScript Pattern 在不增加大量宣告下得到工具支援。

SECTION

二十八、any 與漸進遷移

any 允許：

- 先接入未知 Library；
- 逐步加入型別；
- 處理動態模式；
- 繞過 Checker。

  但也會傳播不安全。

  TypeScript 的選擇是：

> PartialEvidence
> >
> NoAdoption

SECTION

二十九、不追求完整 Soundness

官方設計目標明確把「建立一個可靠或可證明正確的 Type system」列入 Non-goal。[R4]

原因不是安全不重要，而是：

- JavaScript 行為本身高度動態；
- 完全 Sound 會拒絕大量既有慣用法；
- TypeScript 首要目標是錯誤偵測和工具；
- Runtime 不由 TypeScript 控制。

SECTION

三十、TypeScript 型別的主要消費者

型別資訊同時服務：

- Compiler error；
- Completion；
- Hover；
- Navigation；
- Rename；
- Refactoring；
- Quick fix；
- Documentation；
- Language service。

因此：

> TypeSystemValue
> ≠
> OnlyProgramRejection

SECTION

三十一、錯誤恢復

Editor 中的程式經常：

- 未完成；
- 有語法錯誤；
- 缺少 Import；
- 處於重構中。

  Language service 必須在錯誤狀態仍提供有用結果。

  這使工具型 Compiler 與 Batch theorem checker 的目標不同。

SECTION

三十二、Proposal 從使用情境開始

TypeScript 官方的設計提案指南要求先描述：

- 使用者有何困難；
- 哪項常見任務需要改善；
- 現有 Workaround；
- 誰受影響；
- 真實範例。[R7]

  這與 Hejlsberg 跨產品的情境驅動設計一致。

SECTION

三十三、為何用 TypeScript 重寫仍不夠

原有 TypeScript Compiler 以 TypeScript／JavaScript 實作，具有：

- 自舉；
- 可攜；
- 社群熟悉；
- 單一 Codebase。

但大型專案面臨：

- Type checking 延遲；
- Language server 記憶體；
- 單執行緒限制；
- CI 等待；
- Editor 不可用。

SECTION

三十四、Go 原生 Port

2025 年，Hejlsberg 宣布將 Compiler 和 Language service Port 到 Go，目標利用：

- Native code；
- Shared-memory multithreading；
- 較低 Memory；
- Parallel parsing／checking／emitting；
- 更快 Language server。[R8]

SECTION

三十五、相容優先的重寫

TypeScript 7 並未以重新設計語言為主要目標，而是：

- 保持原 Codebase 結構和邏輯；
- 使用數萬測試；
- 比對兩個 Compiler；
- 在真實大型 Codebase 驗證；
- 提供 TypeScript 6 Compatibility package；
- 支援 Side-by-side。

這是一種實作重寫、語義保守。

SECTION

三十六、2026 年 TypeScript 7 邊界

官方 7.0 Release 表示：

- 典型完整 Build 約 8–12 倍加速；
- Language server crash 和失敗命令下降；
- 已在多個大型組織驗證；
- 7.0 尚未提供穩定 Programmatic API；
- 依賴 Compiler API 或 Plugin 的 Angular、Vue、Svelte、MDX 等工作流可能暫時仍需 TypeScript 6；
- 7.1 預計提供新 API。[R6]

所以：

> CompilerSpeedup
> not⇒
> EcosystemMigrationComplete

SECTION

三十七、Turbo Pascal：對微型電腦折衷

- 小記憶體；
- 快 Compiler；
- Pascal 方言；
- 單體 IDE。

SECTION

三十八、Delphi：對 Windows／Native 折衷

- 視覺元件；
- Object Pascal；
- Windows API；
- Native code；
- Vendor Toolchain。

SECTION

三十九、C#：對 Managed platform 折衷

- C-family 熟悉性；
- CLR；

- GC；
- Metadata；
- Component；
- 跨語言 Runtime。

SECTION

四十、TypeScript：對 Web 生態折衷

- JavaScript 語義；
- Structural types；
- Unsoundness；
- Erasure；
- Open-source Compiler；
- 任意 JavaScript Runtime。

SECTION

四十一、共通模式

> NewCapability
> =
> PlatformRespect
> +
> IncrementalImprovement

而不是：

> NewCapability
> =
> PlatformReplacement

SECTION

四十二、微型 Compiler 期

問題：個人電腦缺乏快速、完整 Compiler
策略：小型一遍式 Compiler 和短回饋

SECTION

四十三、Turbo Pascal 期

問題：語言工具碎片化且昂貴
策略：Editor＋Compiler＋Run 整合產品

SECTION

四十四、Delphi 期

問題：Windows 應用開發過慢
策略：Visual component＋Native Object Pascal

SECTION

四十五、C#／.NET 期

問題：Native Windows Component／Versioning 複雜
策略：Managed Runtime＋Component language

SECTION

四十六、LINQ 期

問題：程式與資料語言斷裂
策略：Type inference＋Lambda＋Query abstraction

SECTION

四十七、TypeScript 期

問題：大型 JavaScript 缺乏靜態工具
策略：可擦除漸進結構型別

SECTION

四十八、TypeScript 7 期

問題：工具延遲妨礙超大型程式
策略：Native Go Compiler／LSP＋相容遷移

SECTION

四十九、問題 framing

Hejlsberg 的核心問題是：

> 如何讓開發者在既有平台上更快獲得錯誤、理解和重構回饋，而不用先放棄現有程式、工具及生態？

SECTION

五十、價值優先序

```
V_(Hejlsberg)
≈
(
DeveloperFeedback,
Tooling,
Usability,
TypeInformation,
PlatformFit,
Performance,
Compatibility,
Evolution
)
```

SECTION

五十一、核心—擴張偏好

偏好：

- 熟悉表面；
- 高品質 Compiler；
- IDE；
- Library／Platform；
- Type inference；
- 元資料；
- 由工具擴張體驗。

SECTION

五十二、顯式—推導偏好

C#：

- 強型別；
- 顯式 Public contract；
- Local inference；
- Attribute／Metadata。

TypeScript：

- 可選 Annotation；
- 強推導；
- Contextual typing；
- Structural compatibility；
- any Escape。

SECTION

五十三、效率—可讀性偏好

Compiler／Tool speed 是核心體驗。

Runtime 策略依平台：

- Turbo／Delphi：Native；
- C#：JIT／Managed；
- TypeScript：JavaScript Runtime、型別擦除；
- TypeScript 7：Native Toolchain。

SECTION

五十四、安全—自由偏好

C# 偏向 Managed safety，但保留 unsafe。

TypeScript 偏向漸進證據，明確不追求完全 Soundness。

共同點是：

> 提高安全預設，但保留平台必要能力和遷移出口。

SECTION

五十五、相容性偏好

Hejlsberg 通常避免要求一次全面重寫：

- Turbo Pascal 延續 Pascal；
- Delphi 延續 Object Pascal；
- C# 採 C-family 語法及多語言 CLR；

- TypeScript 是 JavaScript Superset；
- TypeScript 7 提供 Side-by-side transition。

SECTION

五十六、治理偏好

早期產品：

- 小型集中團隊。

C#：

- Microsoft 團隊＋ECMA／ISO＋公開設計。

TypeScript：

- Open-source repository；
- Design goal；
- Issue／Proposal；
- Core team；
- 高度實作與相容測試。

SECTION

五十七、工具好不等於語言好

IDE 可以隱藏：

- 複雜語法；
- 隱式規則；
- Framework 魔法；
- Vendor lock-in。

語言離開工具後仍需可理解。

SECTION

五十八、TypeScript 不保證 Runtime 資料

外部 JSON、Network 和 JavaScript 可以不符合 Declaration。

需要 Runtime validation。

SECTION

五十九、Structural typing 會接受意外相容

兩個物件 Shape 相同，不表示語義相同。

Brand／Nominal pattern 有時仍必要。

SECTION

六十、漸進型別可能永久停在 any

若沒有：

- Strict mode；
- Boundary validation；
- Migration policy；
- CI；

  TypeScript 只會提供局部工具收益。

SECTION

六十一、C# 的平台整合增加重量

CLR、Framework、Metadata、JIT、NuGet 和 IDE 帶來：

- 學習；
- Deployment；
- Version；
- Startup；
- Toolchain；

成本。

SECTION

六十二、RAD 容易把架構延後

快速視覺或框架開發可能讓系統在規模增長後才暴露結構問題。

SECTION

六十三、TypeScript 7 仍是遷移中制度

7.0 的 Compiler 和 LSP 已穩定發布，但 Programmatic API 和部分 Framework 嵌入尚未完成。

不能把性能發布等同整個生態已完全切換。

時期 問題 決策 複雜度去向 風格

1980s 初 微型電腦 Compiler 過慢 Poly／Compas Pascal Compiler 快回饋

1983+ 工具昂貴且分離 Turbo Pascal 整合 IDE 產品化

1995 Windows 應用開發複雜 Delphi Component／Tool RAD

1999–2000 Native Component／Versioning 困難 C#／CLR Managed platform 平台重建

2004–07 Data query 與語言分裂 LINQ Compiler／Provider 語言—資料統一

2012 大型 JavaScript 缺工具 TypeScript Checker／Language service 漸進建模

2025–26 TypeScript Tooling 達規模極限 Go Native Port／TS7 新 Compiler／LSP 基礎重寫

SECTION

六十四、主要原型

Anders Hejlsberg 同時屬於：

- 回饋循環導向工具設計者；
- 語言—IDE—平台共同建築師；
- 實用型別系統設計者；
- 漸進遷移設計者；
- 平台折衷與重寫工程師。

SECTION

六十五、不適合的簡單標籤

不應只稱：

C# 單一發明者
TypeScript 單一作者
商業語言設計者
Java 模仿者
靜態型別純粹主義者
較精確的描述是：

> 反覆在既有平台上加入足以支撐大型開發的型別和工具能力，並把低延遲回饋、遷移及生態相容視為語言核心問題的設計者。

SECTION

六十六、最重要的連續性

從 Turbo Pascal 到 TypeScript 7：

> 【縮短從意圖到可靠回饋的距離】

SECTION

六十七、最重要的變化

型別模型逐步從：

Pascal／C# 的 Nominal static contract
轉向：

TypeScript 對動態 JavaScript 的 Structural、Gradual model
這證明 Hejlsberg 不把自己的舊語言模型當成普遍答案。

SECTION

六十八、最重要的工具判斷

當 Tool latency 破壞使用時，他願意：

- 優化 Compiler；
- 重建 Language service；
- 甚至用另一門語言重寫整個工具基礎；

  但優先保持使用者語義和遷移路徑。

  Anders Hejlsberg 的語言設計不能只用功能清單理解。

  其完整方法是：

- 從真實平台與開發流程開始；
- 使編輯、編譯、診斷及執行形成短回饋；
- 將語言、IDE、Runtime 和 Library 共同設計；
- 使用熟悉表面降低遷移；
- 讓型別資訊同時服務 Compiler 和工具；

- 不追求超出平台現實的形式純度；
- 在新能力與既有生態間建立漸進路徑；
- 當實作架構達到規模極限時，重寫基礎但保護外部行為。

本文對 Hejlsberg 的 PLDST 判定為：

> 【Fast-Feedback Compiler Engineer
> →
> Integrated Platform Language Architect
> →
> Gradual Tooling Systems Designer】

其核心優勢是：

- Compiler 和 IDE 回饋速度被視為一級設計目標；
- 語言與平台能力高度協同；
- C# 對 Component、Versioning 和 Runtime 分工有明確思考；
- LINQ 讓型別和工具跨越資料邊界；
- TypeScript 尊重 JavaScript 生態並提供漸進工具價值；
- TypeScript 7 展示大型工具可在保持語義的前提下重建。

其核心代價是：

- 工具可能遮蔽語言本身複雜度；
- 平台整合造成 Lock-in 和系統重量；
- TypeScript 的不健全性需要 Runtime validation；
- Structural typing 可能混淆形狀與語義；
- 漸進遷移可以永久停在弱保證；
- 原生重寫仍需處理巨大 API 和 Framework 生態。

最終原則為：

> 【語言不是只有能編譯的文本
>
> ∧
> 型別不是只有拒絕程式的證明
>
> ∧
> 相容不是拒絕重寫內部】

Hejlsberg 的歷史提出了一個極具工程性的判斷：

> 一門語言真正的使用介面，是開發者每次輸入之後得到的全部回饋；如果 Compiler、IDE、Runtime 或遷移成本讓回饋失效，再優雅的語法也沒有完成設計。

人物：Anders Hejlsberg
主要語言／系統：Turbo Pascal、Delphi、C#、TypeScript
核心時期：1980s–至今
主要問題：語言能力與開發工具／平台之間的斷裂
主要策略：快速 Compiler、IDE、型別推導、平台 Metadata、漸進型別
複雜度去向：Compiler、IDE、Runtime、Library、Language service
責任去向：工具提前回饋，平台承擔執行與版本
主要保護對象：應用程式開發者、大型程式團隊、既有生態
主要限制：平台重量、不健全型別、工具依賴、遷移和 API 相容
歸因信心：高

[R1] Anders Hejlsberg historical interviews and 2026 retrospective interviews on Turbo Pascal, Delphi, C# and TypeScript.

— Compiler 回饋、產品形成、小型團隊、平台和長期設計方法；後期訪談需與同期文件交叉校對。

[R2] Microsoft／ECMA, C# Language Specification, Introduction.

— C# 由 Microsoft 發展，Principal inventors 為 Anders Hejlsberg、Scott Wiltamuth 和 Peter Golde。

[R3] Microsoft, TypeScript launch talks and “What is TypeScript and Why?” interviews, 2012.

— Application-scale JavaScript、可選型別、Tooling 和 JavaScript 生成。

[R4] Microsoft TypeScript repository, “TypeScript Design Goals.”

— 靜態錯誤、大型元件、Clean JavaScript、Runtime 保持、Structural／Gradual typing 及 Non-goals。

[R5] Bill Venners, Artima C# Design Interviews with Anders Hejlsberg.

— C# 設計程序、Usability／Taste、Checked exception、Delegate、Component、Versioning 及 CLR 折衷。

[R6] Microsoft TypeScript Blog, “Announcing TypeScript 7.0,” 2026.

— Go 原生 Compiler、8–12 倍典型加速、真實專案驗證、7.0 API 缺口及 Side-by-side transition。

[R7] Microsoft TypeScript repository, “Writing Good Design Proposals.”

— 問題情境、受影響使用者、Workaround 及實例優先的提案方法。

[R8] Anders Hejlsberg, “A 10x Faster TypeScript,” Microsoft TypeScript Blog, 2025.

— Native Go Port 的動機、Compiler／Language service、Parallelism 和開源實作。

[R9] Microsoft, articles on LINQ evolution and C# design.

— Peter Golde、Anders Hejlsberg、Compiler extensibility、Query syntax 及多團隊貢獻。

[R10] Borland／Delphi historical materials and Hejlsberg interviews.

— PolyPascal、Turbo Pascal、Delphi 及產品／團隊歸因；部分史料需注意回顧與公司宣傳性質。

[T-P] Microcomputer Pascal phase
[T-T] Turbo feedback-loop phase
[T-D] Delphi component-platform phase
[T-C] C#／.NET phase
[T-L] LINQ phase
[T-S] TypeScript gradual-modeling phase
[T-7] TypeScript 7 native-toolchain phase
[S-F] Fast feedback
[S-I] IDE／language integration
[S-P] Platform fit
[S-T] Tool-driven type system
[S-G] Gradual adoption
[S-R] Internal rewrite with external compatibility

附錄 D　第二輪史實與歸因校對紀錄
D.1 Turbo Pascal 的作者與產品歸因
第二輪重新核對 Hejlsberg 的歷史訪談與 Turbo Pascal 資料：
- Hejlsberg 先開發 Blue Label／Compas／PolyPascal Compiler；
- Borland 授權其 Compiler core，並將 Editor、Packaging、Distribution 和產品策略整合成 Turbo Pascal；
- Hejlsberg 是 Compiler 架構的核心作者及後續版本的重要工程師；
- Philippe Kahn 與 Borland 團隊對產品化、低價策略、IDE 和全球傳播具有獨立作用。

本文因此沒有把 Turbo Pascal 的全部產品成功歸為 Hejlsberg 單人工作。

D.2 Delphi 的歸因邊界

Hejlsberg 是 Delphi 初期 Chief architect，但 Delphi 是：

- Object Pascal 語言；
- Native Compiler；
- Visual Component Library；
- Form designer；
- Database tool；
- IDE；
- Windows platform integration；

的團隊產品。

本文只把「語言—Compiler—IDE—Component 平台共同設計」視為 Hejlsberg 的風格證據，不把全部 VCL、Database、Debugger 或後續 Delphi 版本歸於個人。

D.3 C# 的正式主要發明者

第二輪直接核對 Microsoft C# Language Specification：

- C# 在 Microsoft 內部發展；
- Principal inventors 是 Anders Hejlsberg、Scott Wiltamuth 和 Peter Golde；
- 第一個廣泛發布實作在 2000 年 7 月作為 .NET Framework 計畫的一部分出現；
- C# 後續規格由 Microsoft、ECMA、ISO、設計團隊和開源社群持續演化。

本文因此將 Hejlsberg定位為 Lead architect／主要發明者之一，而非唯一作者。

D.4 Checked exception、Versioning 和 Simplexity

第二輪核對 2003–2004 年 Artima 系列訪談：

- Hejlsberg 對 Checked exception 的主要疑慮確實是 Scalability 和 Versionability；
- C# 對 `virtual`／`override` 的明示要求與 Library evolution 及意外 Override 有關；
- Delegate、Property、Event 和 Metadata 反映 Component software 的一級支援；
- “Simplexity” 表示以平台內部複雜度換取使用者表面簡潔。

這些內容是 Hejlsberg 的一手訪談觀點，不代表所有 C# 團隊成員或研究界對 Checked exception 的唯一結論。

D.5 LINQ 的多作者性

第二輪核對 Microsoft 的 LINQ 歷史：

- Anders Hejlsberg 和 Peter Golde 早期討論語言內查詢；
- Erik Meijer、Peter Drayton、Matt Warren、Mads Torgersen 及多位 Compiler／Library 研究者具有重要作用；
- LINQ 依賴 Generics、Lambda、Extension method、Expression tree、Anonymous type、Type inference 等多項共同成果；
- Query syntax、Provider architecture 和各資料來源實作不可歸為 Hejlsberg 單人設計。

D.6 TypeScript 官方設計目標

第二輪直接核對 TypeScript Design Goals：

正式目標包括：

- 靜態識別可能錯誤；
- 組織大型程式；
- 不增加輸出程式 Runtime overhead；

- 生成清楚、慣用且可辨識的 JavaScript；
- 對齊 ECMAScript；
- 保留所有 JavaScript Runtime 行為；
- 使用可擦除的 Structural type system；
- 跨平台；
- 避免對 TypeScript 1.0 造成重大破壞。

正式 Non-goals 包括：

- 不精確模仿現有語言；
- 不以 Runtime 性能最佳化為 TypeScript Compiler 的主要任務；
- 不追求完全 Sound／可證明正確的 Type system，而是在 Correctness 與 Productivity 間取平衡。

本文因此沒有把 TypeScript 描述為 C# 語法移植或完整安全證明系統。

D.7 Structural／Gradual typing 的邊界

TypeScript 使用：

- Structural compatibility；
- Inference；
- Contextual typing；
- Control-flow narrowing；
- `any`／`unknown`；
- Declaration file。

這使其能建模 JavaScript 生態。

但：

- Type assertion 不會驗證 Runtime 資料；
- `any` 可關閉檢查；
- 外部 JSON 仍需 Runtime validation；
- 形狀相容不保證領域語義相同。

本文把其定位為工具與漸進證據，而非完整 Runtime contract。

D.8 TypeScript 7 的當代狀態

截至 2026 年 7 月 30 日：

- TypeScript 7.0 已於 2026 年 7 月 8 日正式發布；
- 官方 npm 版本在發布材料中為 7.0.2；
- 新 Compiler 和 Language server 以 Go 實作；
- 典型完整 Build 的官方觀察範圍約為 8–12 倍加速；
- 7.0 目標是盡可能相容 TypeScript 6 的 Type checking 和 CLI 行為；
- 7.0 尚未提供穩定 Programmatic API；
- `@typescript/typescript6` 支援 6／7 並行；
- 依賴 Compiler API 或特殊 Language-service plugin 的部分 Framework 仍可能需 TypeScript 6；
- 7.1 預計建立新的 API。

本文因此沒有把 7.0 寫成整個 JavaScript／TypeScript 生態已完成遷移。

D.9 TypeScript 7 的性能數字

官方性能結果來自：

- TypeScript 自身測試；
- Microsoft 及外部大型 Codebase；

- 個別組織回饋；
- Native code 和 Shared-memory multithreading。

不同專案、硬體及工作流的加速不同。

本文使用「典型 8–12 倍完整 Build」作有來源的官方範圍，而不是宣稱所有命令、Editor operation 或專案都固定快十倍。

D.10 工具驅動設計的推論邊界

「回饋循環導向工具設計者」是 PLDST 綜合推論，證據包括：

- Turbo Pascal 快速 Compiler／IDE；
- Delphi RAD；
- C#／Visual Studio／Metadata；
- LINQ 的 Editor／Type checking；
- TypeScript Language service；
- TypeScript 7 Native Port。

它不表示所有具體 IDE 功能都由 Hejlsberg 設計，也不表示 Tooling 可以取代清楚語言規則。

D.11 PLDST 原型邊界

下列名稱為本文分析原型：

回饋循環導向工具設計者

語言—IDE—平台共同建築師

實用型別系統設計者

漸進遷移設計者

它們不是 Hejlsberg 自稱的正式學派名稱，而是跨四代產品決策形成的穩定風格判定。