Alan Kay：物件、訊息與對物件導向的歷史誤讀

Alan Kay: Objects, Messages, and the Historical Misreading of Object-Oriented Programming

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-013.html

英文名稱： Alan Kay: Objects, Messages, and the Historical Misreading of Object-Oriented Programming

系列： Programming Language Designer Style Taxonomy（PLDST）

文件編號： PLDST-013

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

Alan Kay 經常被描述為「物件導向程式設計之父」「Smalltalk 發明者」或「圖形介面先驅」。這些稱號雖能指向其歷史地位，卻也容易把一套原本服務於個人運算、教育、動態媒介、分散式訊息與可延展系統的整體設計，壓縮成「類別、封裝、繼承」的語言功能集合。

Kay 的原始問題並不是如何建立更好的企業資料模型，而是：

> 如何讓普通人，尤其兒童，擁有一種可以閱讀、創作、模擬、探索與改造世界的個人動態媒介？
> Dynabook 概念把電腦視為可攜、連網、可程式化的個人媒介，而不是只用來執行預先封閉應用程式的裝置。Smalltalk 則不是單獨語言，而是這一願景中的語言、圖形環境、物件世界、互動工具與學習系統。[R1][R2][R3]

在 Kay 的歷史回顧中，「物件」受到 Simula、Sketchpad、Burroughs B5000、Lisp、Logo、ARPANET 及生物細胞等多種來源啟發。其深層觀念不是把資料包進類別，而是把系統視為大量具有局部狀態、透過訊息協作、可延後綁定及可在運行中重組的自治單元。[R1] 他後來反覆強調訊息的重要性，正是因為主流物件導向往往把焦點轉到類別階層、資料抽象與靜態結構，而弱化了原先的網路式協作、晚期綁定和完整動態系統。

然而，將 Smalltalk 全部歸於 Kay 同樣構成創始人偏誤。Computer History Museum 保存資料明確指出：Kay 提供願景並領導 Learning Research Group；Dan Ingalls 是早期 Smalltalk 的關鍵實作者，並主導大量語言及系統設計變化；Adele Goldberg 共同開發 Smalltalk，管理後期團隊，並領導其文件、教育及對外傳播；Ted Kaehler、Diana Merry、Dave Robson、Peter Deutsch、Scott Wallace 等人亦參與語言、虛擬機器、介面與應用建設。[R3][R4][R5]

本文以 PLDST 方法將 Kay 的設計生涯分成五個相位：

- 個人反應式媒介形成期：FLEX、圖形互動與可延展語言；
- Dynabook 問題定義期：兒童、個人電腦及建構式學習；
- Smalltalk 訊息系統期：物件、訊息、晚期綁定與活系統；
- Smalltalk-80 公共化與制度轉換期：類別、瀏覽器、文件及外部發行；
- 後期反身批判期：Squeak、Etoys、Viewpoints 及對主流 OOP 和軟體規模的批判。

本文的核心判斷是：

> 【Kay 的主要設計對象不是「物件」本身，
>
> 而是可由訊息協調、可被使用者改造的完整動態媒介。】

因此，其風格不能被縮成：

封裝 + 類別 + 繼承

較精確的表示是：

> 【局部自治
> +
> 訊息協作
> +
> 極端晚期綁定
> +

其主要風險也來自相同設計：極端晚期綁定和運行期可修改性增加工具、追蹤、重現、安全及大型系統治理成本；「一切皆可改造」對研究與學習極具力量，對大規模長期部署則需要額外制度。

關鍵詞： Alan Kay、Smalltalk、物件導向、訊息傳遞、Dynabook、個人運算、動態媒介、晚期綁定、活系統、PLDST

SECTION

一、本文研究的不是全部「物件導向」

本文主要分析：

- Kay 在 Utah 時期的 FLEX 與 Reactive Engine；
- Dynabook；
- Xerox PARC Learning Research Group；
- Smalltalk-72、Smalltalk-76、Smalltalk-80 的主要轉變；
- Kay 對物件、訊息、教育與個人運算的後期回顧；
- Squeak、Etoys 與 Viewpoints 研究方向。

本文不把下列系統全部歸為 Kay 的直接設計：

- Simula；
- C++；
- Objective-C；
- Java；
- C#；
- Ruby；
- UML；
- Design Patterns；

- 所有現代「類別導向」語言。

SECTION

二、Kay 沒有從真空中發明物件

Kay 的〈The Early History of Smalltalk〉明確列出多種重要來源：

- Ivan Sutherland 的 Sketchpad；
- Ole-Johan Dahl 與 Kristen Nygaard 的 Simula；
- Lisp；
- Logo；
- Burroughs B5000；
- ARPANET；
- Doug Engelbart 的互動系統；
- 生物細胞及自治單元；
- 圖形、平板及個人互動裝置。[R1]

因此：

> Smalltalk
> =
> Synthesis(
> Simulation,
> Graphics,
> Messages,
> Networks,
> Learning,
> PersonalComputing
>)

而不是單一語法發明。

SECTION

三、Smalltalk 是團隊系統

角色至少可分為：

3.1 Alan Kay

- 問題設定；
- Dynabook 願景；
- Learning Research Group 領導；
- 物件及訊息的核心方向；
- 教育研究；
- 早期 Smalltalk 概念與語言設計。

3.2 Dan Ingalls

- 第一個 Smalltalk 的關鍵實作者；
- 虛擬機器、BitBlt、顯示與語言系統；
- 多個 Smalltalk 版本的重要設計改動；
- 後來 Smalltalk Zoo 與 Squeak。

3.3 Adele Goldberg

- Smalltalk 共同開發；
- 教育實驗；
- Smalltalk-80 團隊管理；

- 文件、書籍與對外推廣；
- 促使 Smalltalk 離開 PARC 成為外部語言與系統。

3.4 其他 LRG 成員

包括 Ted Kaehler、Diana Merry、Dave Robson、Scott Wallace、Peter Deutsch 等人在語言、系統、虛擬機器、介面與應用上的工作。[R3][R4][R5]

因此：

願景與創始方向：Kay 高
第一實作與技術演進：Ingalls 極高
成熟環境與傳播：Goldberg、Ingalls、團隊極高
Smalltalk 後期語族：多社群

SECTION

四、FLEX 不是 Smalltalk 的簡單前身

Kay 的 FLEX 論文與 Reactive Engine 研究已具有：

- 個人互動電腦；
- 顯示與平板輸入；
- 可延展語言；
- 程式與編輯環境整合；
- 以單一語言描述系統；
- 圖形與文字共同媒介；
- 反應式使用模型。[R6][R7]

其目標不是先設計類別，而是把：

> Language
> +
> Editor
> +
> Display

視為一個整體。

SECTION

五、使用者的計畫本身成為程式

FLEX 的重要方向是：

- 不把程式先寫在紙上；
- 使用者在互動式圖形終端直接組織計畫；
- 編輯、執行及表示不再是互相分離的階段；
- 語言可以描述並延展自身。

  這已預示後來 Smalltalk 的 Image、Browser、Inspector 與活系統文化。

SECTION

六、第一個深層風格：語言不是孤立文本

Kay 對語言的評價單位不是：

Syntax + Compiler

而是：

User + Medium + Language + Tools + Runtime + Community

因此，若只比較 Smalltalk 語法與 Java 語法，就會錯過其主要設計對象。

SECTION

七、Dynabook 不是只預測平板電腦

1972 年〈A Personal Computer for Children of All Ages〉描述一種可攜的個人資訊操縱器，但硬體外形不是核心。

Dynabook 的真正問題是：

- 兒童能否建立自己的模擬；

- 使用者能否創造而非只消費媒體；
- 文字、聲音、圖形、動畫與程式能否成為同一媒介；
- 個人能否保存、傳播及改造知識；
- 電腦能否成為新的讀寫工具。[R2]

SECTION

八、個人是控制中心

傳統大型主機模式：

機構控制機器
使用者提交工作
系統決定何時執行

Dynabook 模式：

個人擁有媒介
即時互動
自由創作
持續保存自己的知識世界

這是一種權力重新配置。

SECTION

九、兒童不是縮小版成人使用者

Kay 受 Seymour Papert、Logo、Jerome Bruner 等教育思想影響，把兒童視為：

- 模型建構者；
- 規則探索者；
- 媒體作者；
- 可以透過操作具體世界發展抽象概念的人。

因此 Smalltalk 需要：

- 即時回饋；

- 圖形；
- 動畫；
- 簡短可試驗程式；
- 可檢視和修改的系統；
- 低失敗成本。

SECTION

十、語言設計的保護對象

Kay 首先保護的不是：

- 編譯器；
- 企業資料庫；
- 標準委員會；

而是：

> 尚未成為專業程式設計者、但需要以電腦思考與創作的人。

這直接改變語言對簡潔、錯誤、圖形和互動的配置。

SECTION

十一、「物件」的生物與網路隱喻

Kay 後來回顧，他把物件想像為：

- 生物細胞；
- 網路上的獨立電腦；
- 具有內部狀態；
- 只能透過訊息與外界協作；

- 不需要知道對方內部實作。

  其核心不是被動資料紀錄，而是自治運算單元。

SECTION

十二、訊息而非直接操作

傳統資料結構模型：

外部程序知道資料布局
直接讀寫欄位
Kay 式物件模型：

發送訊息
接收者自行決定回應
內部狀態與策略保持局部
因此：

> Sender
> not⇒
> Knowledge(ReceiverImplementation)

SECTION

十三、晚期綁定

訊息含義不必在：

- 編譯期；
- 類別定義時；
- 連結時；

  完全固定。

  接收者可依：

- 當前物件；

- 方法字典；
- 系統狀態；
- 後續修改；

  決定回應。

  這提高：

- 可延展；
- 互動；
- 替換；
- 模擬；
- 活系統修改。

  同時增加：

- 錯誤延遲；
- 工具推理；
- 效能預測；
- 大型系統追蹤。

SECTION

十四、Smalltalk-72 的訊息模型

Smalltalk-72 的訊息處理比後來熟悉的固定方法查找更自由：

- 接收者參與解析訊息；
- 訊息模式具有可延展性；
- 物件更接近微型語言處理器；

- 使用者可以建立局部語言。

  這種自由極強，但也難以：

- 最佳化；
- 統一工具；
- 解釋大型系統；
- 建立穩定語法。

SECTION

十五、Smalltalk-76 的制度化

後續版本逐步引入：

- 更規則的方法字典；
- 更固定、可工具化的類別組織；
- Simula 式繼承及差異式定義；
- 更一致的語法與虛擬機器；
- Browser；
- Inspector；
- Debugger。

  這表示：

> MessageFreedomdownarrow
>
> SystemRegularityuparrow

Smalltalk 的成熟並不是單純把 Kay 原始觀念完整放大，而是由團隊在自由、實作、性能與教學間重新配置。

SECTION

十六、消除語言內部階級

若數字、類別、方法、視窗、錯誤等都能作為物件參與系統，使用者不必反覆切換：

- 特殊內建；
- 普通資料；
- Compiler-only entity；
- Runtime-only entity；
- Library entity。

  這提高概念一致性。

SECTION

十七、反射與活系統

Smalltalk 環境讓使用者能：

- 檢查物件；
- 修改類別；
- 修改方法；
- 在 Debugger 中修復；
- 保存整個 Image；
- 立即看到改動；
- 使用系統本身建造工具。

  語言與 IDE 不是分開產品，而是同一物件世界。

SECTION

十八、錯誤是探索入口

在傳統 Batch 模式中：

錯誤 → 程式終止 → 修改 → 重編譯 → 重跑

在活系統中：

錯誤 → Debugger 打開 → 檢查狀態 → 修改方法 → 繼續

這對教育、研究與探索具有巨大價值。

SECTION

十九、活系統的治理負擔

但 Image-based 系統可能造成：

- 來源與執行狀態不一致；
- 變更歷史難追蹤；
- 部署內容難重建；
- 隱藏全域狀態；
- 團隊整合困難；
- 安全邊界模糊。

現代版本控制、Package 與可重現建置需要額外制度補足。

SECTION

二十、從研究系統轉為可傳播制度

Smalltalk-80 不只需要技術成熟，還需要：

- 語言定義；
- 虛擬機器規格；

- 書籍；
- 教材；
- 瀏覽器；
- 移植；
- 對外授權；
- 外部社群。

Adele Goldberg 在此階段的管理、文件和傳播工作不可被視為次要附錄。[R4][R5]

SECTION

二十一、公共化改變設計

研究室內可依賴：

- 團隊口傳；
- 共同 Image；
- 即時協助；
- 特定 Alto；
- 兒童實驗。

對外系統則需要：

- 明確行為；
- 穩定工具；
- 可移植 VM；
- 可教學概念；
- 可重建版本。

公共化必然提高制度一致性，並降低部分局部自由。

SECTION

二十二、類別與繼承不是全部 OOP

Smalltalk-80 的類別模型後來被大量模仿，主流 OOP 因而常把：

```
Object-oriented
=
Class + Inheritance + Encapsulation + Polymorphism
```

視為定義。

但對 Kay 而言，更核心的是：

- 訊息；
- 局部狀態；
- 晚期綁定；
- 自治；
- 整體系統可延展。

  類別主要是實作與組織手段，不是終極觀念。

SECTION

二十三、誤讀一：物件就是資料加方法

此模型接近 Abstract Data Type：

```
資料
+
作用於資料的程序
+
存取限制
```

它可以是良好工程方法，卻弱化：

- 訊息協議；

- 自治；
- 動態替換；
- 網路式組合；
- 接收者決定含義。

SECTION

二十四、誤讀二：OOP 的核心是繼承

繼承可支援：

- 程式碼重用；
- 分類；
- 差異式定義。

但它也可能形成：

- 深層階層；
- 脆弱基類；
- 實作耦合；
- 靜態分類中心。

Kay 的歷史敘述更重視訊息和晚期綁定，而非把繼承視為本體中心。

SECTION

二十五、誤讀三：Smalltalk 只是語法更純的 Java

Java、C++ 等語言可以借用 Smalltalk 的：

- 類別；
- 方法；

- 動態派發；
- 垃圾回收。

但 Smalltalk 還包括：

- Image；
- Browser；
- Live debugging；
- Program as manipulable objects；
- 即時修改；
- 個人媒介；
- 教育願景。

只比較語法會遺失系統層。

SECTION

二十六、誤讀四：Kay 否定所有主流 OOP 都沒有價值

Kay 的批判主要指出：

- 觀念被縮小；
- 系統規模與可理解性未改善；
- 訊息及晚期綁定被弱化；
- 個人創造媒介變成封閉 App 消費。

這不代表：

- 類別毫無用途；
- 靜態型別必然錯誤；

- C++／Java 沒有工程價值；
- 所有物件都必須非同步；
- Smalltalk 已完成所有原始願景。

SECTION

二十七、Squeak

Squeak 延續：

- Smalltalk-80；
- 可移植 VM；
- Image；
- 多媒體；
- 教育；
- 系統可自我實作。

它同時是保存歷史與重新實驗的平台。

SECTION

二十八、Etoys

Etoys 把：

- 物件；
- 腳本；
- 圖形；
- 模擬；
- 即時回饋；

重新放回兒童的創作環境，直接連接 Dynabook 的原始教育問題。

SECTION

二十九、Viewpoints 與軟體規模批判

Kay 後期研究反覆關注：

- 軟體規模膨脹；
- 現代系統難以完整理解；
- 真正個人可控制電腦尚未完成；
- 語言應能以更少核心建立完整系統；
- 教育媒介仍未實現其潛力。

這使其風格從「發明一門語言」轉向：

> 重新尋找可以讓完整個人計算系統保持小、可理解、可塑及可學習的元系統。

SECTION

三十、複雜度配置

Kay 的設計希望降低：

- 使用者的機器操作；
- 顯式記憶體管理；
- 編譯—執行分離；
- 工具切換；
- 封閉應用限制；
- 預先固定的系統邊界。

但增加：

- Runtime；
- VM；
- 動態派發；
- Image 管理；
- 工具與系統整合；
- 變更治理；
- 性能與除錯的非局部性。

SECTION

三十一、責任配置

系統承擔：

- GC；
- Dynamic binding；
- Object representation；
- Tool integration；
- Interactive debugging；
- Graphics。

使用者承擔：

- 訊息協議設計；
- 動態變更紀律；
- 系統狀態理解；

- 元程式設計邊界；
- 長期可維護性。

SECTION

三十二、安全—自由配置

Smalltalk 的自由來自：

- 幾乎所有事物可檢視；
- 幾乎所有方法可修改；
- 訊息可動態處理；
- 程式與工具在同一世界。

其安全主要依賴：

- 對象邊界；
- 影像環境；
- 團隊慣例；
- Runtime；
- 部署控制。

它不是以現代 Capability、安全型別或 Sandbox 為核心的安全模型。

SECTION

三十三、FLEX 期

問題：互動電腦被語言與工具分裂
策略：可延展單一媒介

SECTION

三十四、Dynabook 期

問題：電腦由機構和專家控制
策略：兒童也能創作的個人動態媒介

SECTION

三十五、Smalltalk-72 期

問題：系統需由自治單元動態協作
策略：訊息、局部狀態、極端晚期綁定

SECTION

三十六、Smalltalk-76／80 期

問題：自由模型難以實作、最佳化與傳播
策略：類別、方法字典、規則化 VM、公共文件

SECTION

三十七、後期反身期

問題：主流 OOP 與個人電腦只採用表面
策略：Squeak、Etoys、小型完整系統、重新發明

SECTION

三十八、問題 framing

Kay 的核心問題不是：

> 如何組織程式碼？

而是：

> 如何建立一種每個人都能用來建模、創造和擴展思想的動態媒介？

SECTION

三十九、價值優先序

> V_(Kay)
> ≈
> (

SECTION

四十、核心—擴張偏好

核心偏好：

- 少量一致物件規則；
- 訊息；
- 動態派發；
- 活環境。

擴張主要透過：

- 新物件；
- 新訊息協議；
- 系統內工具；
- 元層修改；
- Image 與教育應用。

SECTION

四十一、顯式—推導偏好

Kay 偏好讓：

- 物件自行決定訊息含義；
- 綁定推遲；
- 系統在 Runtime 保持可塑。

這犧牲部分靜態顯式性，換取系統延展性。

SECTION

四十二、效率—可讀性偏好

意圖可讀性高於：

- 固定布局；
- 靜態派發；
- 預先封閉最佳化。

  但 PARC 團隊仍大量投入 VM、BitBlt、Bytecode 與硬體，以使高階互動媒介實際可用。

SECTION

四十三、治理偏好

研究期偏好：

- 小團隊；
- 可快速重寫；
- 實際使用者試驗；
- 完整系統；
- 工作原型。

  公共化後則需要：

- 文件；
- VM 規格；
- 外部實作；
- 教材；
- 社群。

SECTION

四十四、Simula 的優先性

Simula 在 Smalltalk 前已具有 Class、Object、Process、Virtual procedure 等重要機制。

Kay 的貢獻是：

- 綜合與重新 framing；
- 訊息和自治的強化；
- 完整個人互動媒介；
- 「Object-oriented」名稱與新系統方向。

  不能把物件全部起源歸於 Kay。

SECTION

四十五、Smalltalk-80 與 Kay 原始模型不完全相同

Smalltalk-80 的：

- 同步訊息；
- 類別；
- 方法字典；
- 工具；
- 公共規格；

  是團隊成熟化結果。不能將每個細節直接解釋成 Kay 最初意圖。

SECTION

四十六、晚年「訊息」說法具有回顧性

Kay 後期對訊息的強調具有高度解釋價值，但仍需與：

- Smalltalk-72 文件；
- HOPL 回顧；
- 實際版本；
- 團隊證詞；

交叉使用，避免把晚年一句話回寫為所有時期唯一完整定義。

SECTION

四十七、兒童可編程不等於無需教學

建構式環境仍需要：

- 教師；
- 任務設計；
- 數學與媒體素養；
- 工具可用性；
- 漸進概念；
- 社群。

媒介能力不能自動產生教育成果。

SECTION

四十八、活系統不天然適合所有部署

安全關鍵、可重現、分散式企業與長期維護場景可能需要：

- 不可變發行物；
- 審批；

- 靜態契約；
- 重建；
- 權限；
- 版本化。

Kay 式可塑性需要制度化限制。

時期 問題 決策 複雜度去向 風格

1960s 語言、編輯器、機器分裂 FLEX／Reactive system 系統整合 媒介統一

1968–1972 個人無法創作運算媒體 Dynabook 裝置與環境 個人賦權

1972 系統需動態自治 Smalltalk-72 message model Runtime 訊息自治

1976 原模型過於自由 類別、方法字典、VM 系統規則 制度化

1980 需要對外傳播 Smalltalk-80、文件與移植 團隊／標準 公共化

1990s 後 主流只採用表面 Squeak、Etoys、VPRI 小型完整系統 反身再造

SECTION

四十九、主要原型

Alan Kay 同時屬於：

- 個人動態媒介設計者；
- 訊息自治架構師；
- 活系統建築師；
- 教育導向語言設計者；
- 反身式計算媒介批判者。

SECTION

五十、不適合的簡單標籤

不應只稱：

OOP 發明者
Smalltalk 單一作者
GUI 發明者
類別與繼承設計者
平板電腦預言者
較精確的描述是：

> 以兒童和個人創作者為保護對象，將語言、圖形、工具、物件、訊息與電腦整合成一種可被使用者持續改造之媒介的設計者。

SECTION

五十一、最重要的連續性

從 FLEX、Dynabook 到 Smalltalk、Etoys：

> User
> →
> Author
> →
> SystemBuilder

使用者不只是消費功能，而能修改媒介本身。

SECTION

五十二、最重要的制度轉換

Smalltalk 從：

Kay 與 LRG 的研究媒介
轉為：

由 Ingalls、Goldberg、PARC 團隊成熟、文件化和向外傳播的公共語言系統

SECTION

五十三、最重要的歷史誤讀

主流 OOP 保存：

- 類別；
- 方法；
- 封裝；
- 動態派發。

  卻常弱化：

- 訊息作為協議；
- 自治單元；
- 活系統；
- 個人媒介；
- 使用者改造完整系統。

  Alan Kay 的設計生涯顯示，程式語言可以是整個媒介革命的一部分，而不是編譯器前端的語法選擇。

  他的核心鏈條是：

> 【個人擁有電腦
> →
> 個人能建立模型
> →
> 模型由自治物件組成
> →
> 物件以訊息協作
> →
> 整個系統可在運行中被理解與改造】

本文對 Kay 的 PLDST 判定為：

> 【Personal Dynamic-Medium Visionary
> →
> Message-Oriented System Architect

他的主要優勢是：

- 以人的創造能力而非機器規格定義語言問題；
- 將語言、工具、圖形及 Runtime 統合；
- 以訊息和局部狀態降低直接耦合；
- 讓系統能在自身內部被檢視及修改；
- 將兒童納入真正的計算媒介設計。

其主要代價是：

- 極端動態性增加非局部推理；
- 活系統增加重現與部署難度；
- 元層自由需要安全與治理；
- 類別和訊息成熟化後可能偏離最初自治隱喻；
- 願景規模遠超單一語言可以完成的範圍。

最終原則為：

> 【物件不是被包裝的資料
>
> ∧
> 訊息不是普通函式呼叫的另一個名字
>
> ∧
> Smalltalk 不是孤立語言】

它們共同服務於一個更大的設計：

> 讓電腦成為人能閱讀、創造、試驗、溝通並重新發明的動態媒介。

人物：Alan Kay
主要語言／系統：FLEX、Smalltalk、Dynabook、Squeak、Etoys
核心時期：1960s–2000s

主要問題：個人與兒童缺乏可創作的計算媒介
主要策略：物件、訊息、晚期綁定、活系統、圖形互動
複雜度去向：Runtime、VM、工具與系統整合
責任去向：物件自行響應，系統承擔記憶體與工具
主要保護對象：學習者、個人創作者、系統探索者
主要限制：動態追蹤、安全、部署、重現、規模
歸因信心：高

[R1] Alan C. Kay, “The Early History of Smalltalk,” HOPL II／ACM SIGPLAN Notices, 1993.

— Smalltalk 的前代來源、物件觀念、Smalltalk-72／76、PARC 團隊及後期回顧。

[R2] Alan C. Kay, “A Personal Computer for Children of All Ages,” ACM National Conference, 1972.

— Dynabook、兒童、個人資訊媒介與建構式使用情境。

[R3] Computer History Museum, “Smalltalk at 50” and “Introducing the Smalltalk Zoo.”

— Learning Research Group、Smalltalk 多版本、Kay 願景、Ingalls 實作和 Goldberg 傳播角色。

[R4] Computer History Museum profiles of Alan Kay, Adele Goldberg and Dan Ingalls.

— Smalltalk 共同開發、團隊角色、個人運算與教育貢獻。

[R5] Adele Goldberg and David Robson, Smalltalk-80: The Language and Its Implementation, 1983; Smalltalk historical preservation collections.

— Smalltalk-80 語言、虛擬機器、工具與對外制度化。

[R6] Alan C. Kay, FLEX—A Flexible Extendable Language, University of Utah, 1968.

— 可延展語言、統一互動媒介與系統整合。

[R7] Alan C. Kay, The Reactive Engine, University of Utah, 1969.

— 個人反應式電腦、圖形互動、FLEX Machine 與媒介願景。

[R8] ACM, Alan Kay A.M. Turing Award materials, 2003.

— Smalltalk、物件導向、個人運算及圖形介面的獎項歷史定位。

[R9] Alan Kay, OOPSLA talks and later essays on messaging, personal computing and software scale.

— 後期對主流 OOP、訊息與尚未完成之電腦革命的反身批判；使用時須標記為後期回顧。

[T-F] FLEX／reactive-medium phase
[T-D] Dynabook phase
[T-M] Message-oriented Smalltalk phase

[T-P] Public Smalltalk-80 phase
[T-R] Reflexive reinvention phase
[S-U] User agency
[S-M] Messaging
[S-L] Late binding
[S-V] Live system
[S-E] Educational medium
[S-X] Extensible whole-system design

附錄 D　第二輪史實與歸因校對紀錄
D.1 Dynabook 的設計範圍
第二輪重新核對〈A Personal Computer for Children of All Ages〉與 Computer History Museum 的 Smalltalk 歷史材料：
- Dynabook 確實被描述為可攜、個人、可操作文字、圖形、聲音及程式的媒介；
- 兒童和成人都在原始使用情境中；
- 使用者能維護及編輯自己的文字與程式，而不是只執行封閉應用；
- Smalltalk 是支撐此動態媒介願景的軟體環境之一；
- 本文沒有把 Dynabook 簡化成現代平板外形預測，也沒有聲稱今日平板已實現其教育及可程式化願景。

D.2 Smalltalk 的團隊歸因
已重新核對 CHM 的 Smalltalk at 50、Smalltalk Zoo 及人物資料：
- Kay 主要負責願景、問題 framing 與 Learning Research Group 領導；
- Dan Ingalls 是早期 Smalltalk 的 Lead programmer，並負責多項語言與系統變化；
- Adele Goldberg 是共同開發者，後來管理 LRG 延伸團隊，並領導文件、教育和對外傳播；
- Ted Kaehler、Dave Robson、Diana Merry、Scott Wallace、Peter Deutsch 等人亦參與不同版本與環境；
- 本文因此沒有使用「Kay 單人實作 Smalltalk」或「Smalltalk-80 全部是 Kay 設計」的說法。

D.3 Simula、Sketchpad 與「物件發明者」
第二輪核對 Kay 的 HOPL 回顧：
- Simula 的 Class、Process、Virtual procedure 及配置模型是直接重要來源；
- Sketchpad 對 Master／Instance、圖形約束及互動物件提供另一條來源；
- Lisp、Logo、B5000、ARPANET 和生物隱喻亦參與其綜合；
- Kay 可以被描述為創造「object-oriented」術語及 Smalltalk 核心方向的重要人物；
- 不應說他從無到有發明所有 Object、Class 或 OOP 機制。

D.4 Smalltalk-72 與 Smalltalk-76
已重新核對 Kay 的 HOPL 論文及 Dan Ingalls 對六代 Smalltalk 的回顧：
- Smalltalk-72 的訊息語法和求值模型與 Smalltalk-80 有顯著差異；
- 接收物件在早期模型中可參與解析訊息，具備近似局部語言處理器的自由；
- 為提高速度、規則性與工具支援，後續版本限制部分自由；
- Smalltalk-76 建立更規則的 Class、Method dictionary、Inheritance 與開發環境；
- 本文將這一過程寫成「自由降低、制度規則提高」，而不是宣稱 Smalltalk-72 完全沒有 Class 概念。

D.5 「訊息才是大觀念」的時間邊界
Kay 2003 年對 OOP 的說明，把 Messaging、局部保存及隱藏 State-process、Extreme late binding 列為核心。
第二輪校對採取以下分層：

早期文件：證明訊息、物件、晚期綁定確實自一開始重要

1993 HOPL：Kay 對歷史整體的系統回顧

2003 郵件及後期演講：對主流 OOP 誤讀的反身性濃縮

本文沒有把 2003 年一句話當成 1972 年全部團隊成員共同接受的正式規格，也沒有將它擴張成「所有合法 OOP 都必須與 Smalltalk 完全相同」。

D.6 Smalltalk-80 的公共化

第二輪核對 CHM 與 Smalltalk-80 歷史資料：

- Smalltalk-80 需要 Language／VM 定義、書籍、移植與外部發行；
- Goldberg 對公共文件和推廣的作用具有獨立重要性；
- Ingalls 對 VM 與多版本演化具有獨立重要性；
- 公共化後的 Smalltalk 是制度和團隊成果，不是單純將 Kay 的口頭願景寫成標準。

D.7 活系統與歷史記錄

本文對 Image-based 系統可能造成版本、重建和狀態治理困難的判斷，是根據其系統特性作出的 PLDST 推論。

它不表示 Smalltalk 生態沒有：

- Change set；
- Source repository；
- Package；
- Image segment；
- 後來的版本控制工具。

更精確的結論是：活系統把「執行狀態即開發狀態」的能力提高，因此需要額外工具把可塑性轉化為可審計歷史。

D.8 後期研究邊界

Squeak、Etoys 與 Viewpoints 延續 Kay 的：

- 教育；
- 個人媒介；
- 小型完整系統；
- 元系統；
- 軟體規模批判。

但各專案都有新的共同作者及團隊。本文只把其反覆問題設定歸為 Kay 的長期風格，不把全部程式與成果歸於個人。

D.9 PLDST 推論標記

下列名稱是本文分析原型，不是 Kay 自稱的正式學派：

個人動態媒介設計者

訊息自治架構師

活系統建築師

反身式計算媒介批判者

它們由多時期決策共同推導，信心為中高至高。