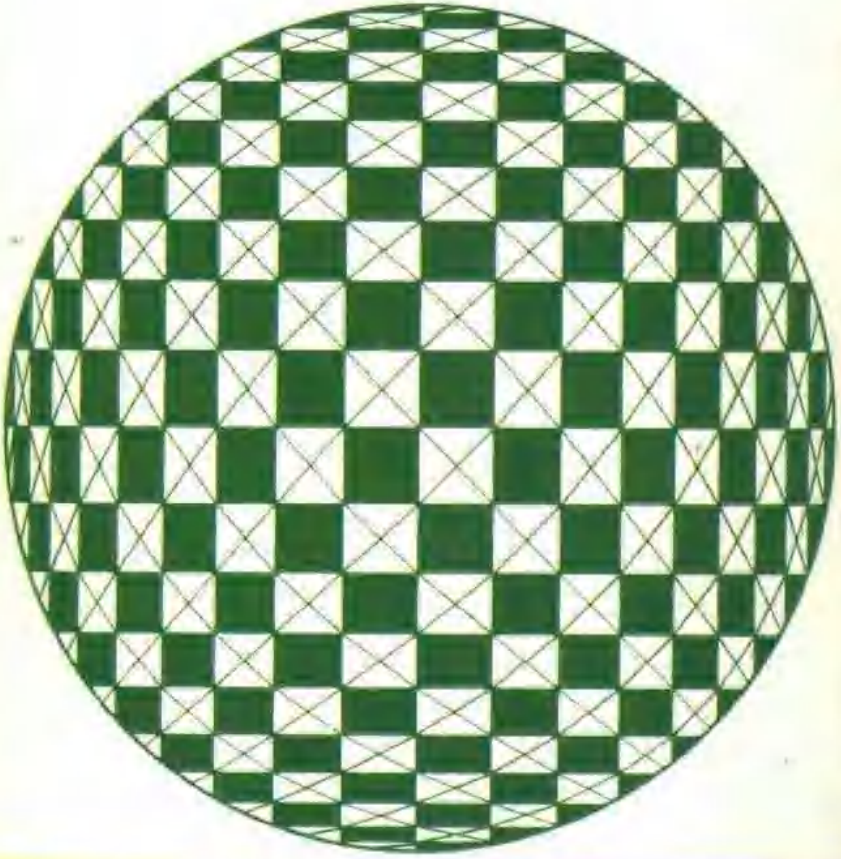
軟體的複合結構化設計

王學智 ■ 譯編



力新出版社

軟體的複合
—結構化設計

編譯・王 學 智

力新出版社

軟體的複合
—結構化設計

編譯者：王學智
出版者：
發行者：力新出版社
九龍新山道四三八號
承印者：力行印刷公司
香港柴灣保興工業大廈8樓A座

定價：港幣

前　言

我們當然可以知道1970年代是軟體比硬體受到更多開發的十年，當我們談論到有關資料系統的可靠性與經濟性時，與其將焦點對準硬體，不如將焦點對準軟體更恰當。

時至今日，軟體的成本已遠超過硬體的成本。因此，過去二十年來一直認爲有問題者（譬如硬體成本的最小化，或爲能有效應用於一切計算機所做的努力），已不再是那麼嚴重的問題，反而例如可靠性不如預期的高，超過了開發預算，或開發預定進度延後等，這些發生在軟體上的問題受到更多的矚目。

因爲這種情勢的變化，乃從事於改善程式計畫過程所用的一連串方法或構想。本書是這種方法論之一，將論及我們大家所熟悉複合設計的技巧。

在這些新的方法或構想（譬如，結構化程式計畫，高級程式設計師，團體寫碼工作）之中，也許以複合設計的知名度最低。其理由是因爲目前尚缺少有關複合設計的基礎數學式理論（因此，對抗許多學會的力量單薄）的緣故。此外，還可以舉出主題太大，無法在商用雜誌的記事欄或大約三頁程度的「程式計劃相關標準」說明完盡也是理由之一。另一方面，對大多數實際引進符合設計的人士來說，均感到

在新技巧裏這是最有價值的技巧。這是爲了要求更廉價，更高的可靠性，也設計擴大性更高的程式，因而所需解決的的基本問題，複合設計會給我們一些線索的緣故。

複合設計的主題從1976才開始引起世人的矚目。其濫觴就在於當年所進行的NCC(National Computer Conference)所作的有關此主題的三個小時討論會。在討論會中，此領域的四個主要貢獻者的Larry Constantine, Michael Jackson, Edward Yourdon, 和筆者，討論了各人不同的看法。此SETTOM在NCC邀請了最多的學者參加。

在討論複合設計時，導致混淆不清的原因就在於二個不同的名稱，也就是說，複合設計與結構化設計。筆者喜歡使用"複合設計"這一句專有名詞，而本書也使用此專有名詞。其理由是雖然曾經叫過以與"結構化設計"相同的專有名詞，但我卻知道，至少有六種完全沒有關係的設計技巧。

本書的目的就是要將中規模或大規模的程式結構設計所用的若干目標與方法，提供給各位程式設計師或系統分析師。本書的內容雖然以適用業務的程式設計爲中心，但其觀念也可在系統軟體中使用（譬如操作系統或各種編譯程式）或爲程式之設計。

其實，筆者於1974年曾寫了一本和複合設計有關的書。因此也許有人會懷疑，筆者爲什麼將相同的主題寫成另一本書也說不定。理由有三：其一就是這幾年來，筆者對複合設計累積了相當多的見解，也充實了有益的知識；其二筆者曾經從教授學生的經驗中獲得此主題；是透過與實際使用這種技巧的人士進行研討，因而累積了新知識等等。

筆者深信，凡是程式設計師、系統分析者、程式計劃管理者以及電腦科學的學生們，必須熟悉本書所說明的各種看法。爲什麼呢？這

此为试读,需要完整PDF请访问: www.ertongbook.com

是因爲要設計可提高可靠性，也容易維護的有擴大性的程式，在我們考慮計算系統之經濟性時會成爲基礎的緣故。

本書可採用於以程式開發實務參與的各階層人士爲對象的講習會，或大學課程的參考書、教科書。幾乎每一章後面都舉出了若干質疑與問題，這對於要將本書當作教科書使用時，也許會有所幫助。

在學習各章時，請各位務須解答問題。這是因爲有許多概念，需要實際動手做，否則很難了解。以複合設計爲主題的講習會，必須包括設計實際問題的一些事項。講師必須製作程式所需的現實規範，而受講者（個人或團體）必須設計該程式，而且必須評價該設計的結果。爲供參考起見，筆者經常使用第12章所說明的程式作爲習題。恐怕像這種程度大小的問題是最恰當的（爲了要設計第12章的程式，受講者平均花上15小時）。

本書共16章，區分爲三個範疇。前6章定義複合設計的目標，對於複合設計的各種目標，以及評價巧妙設計程式所用的尺度。從第7章起至第13章，將焦點放在設計方法（how-to）。換句話說，亦即當我們承認前6章所說明的例如目標或尺度時，對於爲獲得所希望之結果而應採取那一種思考過程（分割之法）的這個問題之檢討。最後分三章檢討引進複合設計時面臨的問題。例如，和程式計劃語言所處的關連性或結構化程式計劃或程式計劃團體和其他方法論之關係也都有詳細的說明。

在此感謝支援與複合設計相關的研究、講義、研討的IBM SRI（Systems Research Institute）。同時，也一併致謝許多受講者對各種構想進行有價值的討論、表示意見，並予以批評。而R. Goldberg對本書的製作也提供了許多寶貴的意見。筆者又對GUIDE International的PL／I提高生產性的許多人士的旺盛意欲，以及不但要使用複合設計，同時也將他們的經驗不吝告知於我的一切人士致

十二萬分的謝意。

Glenford J. Myers

於紐約

軟體的複合—結構化設計目錄

第1章　程式設計的著眼點……11

設計過程……17
參考文獻（有註釋）……19

第2章　定義與表記法……25

結構性的專有名詞……26
機能．邏輯．用法……27
表記法……29
程式結構圖的寫法……32
ＨＩＰＯ圖……33
參考文獻……34
習　題……35

第3章　良好設計的基礎……37

區分……37
階層結構化……39
獨立性……40
參考文獻……46
習　題……46

第4章 模式強度 ……48

暗合性強度……49
邏輯性強度……52
時間性強度……55
步驟性強度……56
連絡性強度……57
機能性強度……57
資訊性強度……59
參考文獻……63
習 題……63

第5章 模式結合度 ……66

內容結合……68
共同結合……70
外部結合……76
控制結合……77
打印結合……80
資料結合……82
打印結合的最小化……83
結合度的代替……85
參考文獻……87
習 題……87

第6章 其他的設計目標……89

模式大小……89
有關模式之獨立性的其它要因……90
分界面之重複……91
機能與分界面之統一……92
模式被呼叫之數的最大化……93
限定模式……94
有先見性的模式……95
遞歸……96
習 題……99

第7章 設計的思考過程……101

步驟的概要……103
停止……106
二條路……106
參考文獻……107

第8章 原始／變換／吸收分割……108

程式結構的概要……109
使主要資料的流程明確……110
找出最大抽象點……112
直接從屬模式之定義……114
技巧之修正……115

習　題……118

第9章　業務分割……121

習　題……123

第10章　共同機能分割……124

習　題……126

第11章　資料結構分割……127

結構的不一致……133
程式的轉化……136
參考文獻……139

第12章　分割之事例……140

狀況說明……140
問題……142
外部規範書……143
現有的模式……147
解決方案……148
設計的分析……163
習　題……163

此为试读，需要完整PDF请访问：www.ertongbook.com

第13章 最佳化與檢討……165

結構的最佳化……165
靜態性設計的檢討……168
動態性設計的檢討……170

第14章 程式計劃語言……174

語言的特性……175
PL/I……177
FORTRAN……178
COBOL……180
APL……181
RPGII……182
BASIC……182
語言的改良……183
習 題……188

第15章 和其他方法論所處的關係……190

結構化程式計劃……191
程式計劃團隊……193
上下開發……195
模式分界面的設計……197
參考文獻……199

第16章 整理和其他……201

性能……203
程式的維護……208
程式的修正……209
適用複合設計的出發點……210
參考文獻……212

習題解答……212

第 1 章

程式設計的著眼點

回顧過去，從程式開發開始到完畢爲止，透過全體放眼看整個過程時，即使是微乎其微，人類的成功與開發電腦的適用業務，毋寧說是令我們驚歎的。對些新的令人狂熱的程式或系統，成爲程式開發過程輸入之要件規範，至短而籠統，而且是不完整的。 開發的結果已產生爲數達數百萬的0與1位元串，而這些位元串將系統的動作指示給一組電子裝置。

人們會立即意識到要這與程式計劃有關的研究減低其複雜性，是導至軟體開發成功的關鍵。許多程式與組合語言的概念是針對這個目標的第一個步驟。即使程式設計師爲了要將二個暫存器的值互相比較，而必須將011010這個位元串寫碼，我們也不必記取這件事。

在此步驟之後，人人開始認識需要有比用程式設計師管理複雜程式更佳的方法或工具。於是所謂的結構化程式計劃、讀碼、開發援助公用程式等構想也就開始受到矚目。

這些構想所作的貢獻雖大，但也有很多令人遺憾的地方。譬如，

採用世界最麻煩的程式（爲大家所熟悉的操作系統），用高階語言與結構化程式計劃的概念改寫，其結果筆者也不會認爲有多大的改善。換句話說，只是將存在於一程式之各部份的局部性複雜抑制到最小化也無濟於事，而對複雜的問題，則存在更多的要因。複雜的而重要的要因，就是大局性的複雜。所謂大局性的複雜，是指程式或系統總體性結構的複雜（亦即存在於程式主要部份之間的結合程度或內部依存度）。

本書就是要提示對大局性或結構性複雜之問題的解決法（了解複雜的程度，將它處理抑制到最小化），所以在考慮到解決法之前，首先要檢討與這些複雜有性有關的問題，這是很重要的。下面讓筆者對程式設計領域比較基本的九個問題，簡單地檢討。

幾乎全然沒有結構被設計的較具代表性的程式。其結構通常是在寫碼過程中，以獨特的方法製作的。

這一點有我們所做的決定的許多事，從順序上來看，是相反的意思。在這裏筆者並不是最初要檢討系統的結構面，然後再檢討屬於步驟性之面，而是我們是任性的容易做其相反的事。換句話說，我們會患雖然一面把程式寫碼，卻一面決定重要的整體性結構的這種錯誤。

這一點當歸究於許多人。例如說，資料處理部門的管理者在設計過程時，實際上雖是沒有多大意義的事，但卻因爲不能做到“寫碼正完成了95％”或“消化了測試例子的82％”這種進展管理，所以，在這期間會被不安所侵襲。也就是說，設計的進展度很難定量化，因而許多生產管理者一心一意要儘可能把生產按照預定完成手續的設計局面把它做完。

關於教育的方法，也是有一個問題。程式設計師或學生所以要受有關程式計劃之教育，要受的就是程式計劃語言、良好的程式計劃書法、標準、文件化之技術、演算法等，有關於分析或設計的方法通常

都會脫離教育的對象。

所有的程式均被設計成不能對應要件或環境之變化的方法。

我們必須研究資料處理所面臨的比較基本的事實。程式設計師很少寫用畢而不用的程式。幾乎是所有的程式都是有長的壽命，通常會在這段期間受到很多的變更。可是，到目前爲止，程式的修正通常是昂貴的，是很容易帶來錯誤的作業。這一點有使長時間程式開動時，程式的擴大性或適應性是重要的經濟上要因的意思。

程式設計師爲了修正錯誤而耗費了大部分的時間。

所謂的錯誤的修正作業是無窮盡的。試以調查較具代表性的程式計劃，我們可以知道程式設計師在整個計劃的期間，把自己的作業時間的50％以上消磨在錯誤的發現與修正。這種問題的通常“解決法”值得我們矚目。那雖是在計劃結束時希望留下足夠的時間的緣故，但是，這個時間就使用於要發現因爲催促設計過程藉以發現所發生的

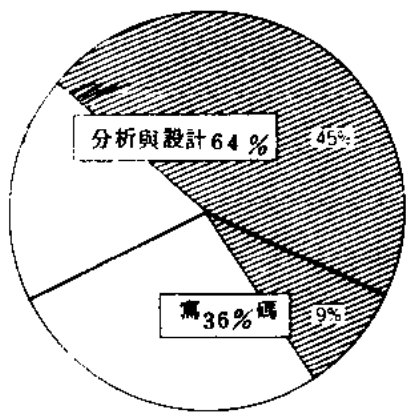


圖1．1 錯誤發生部位與利用測試發現的錯誤

錯誤。

由於不充分重視，因而所產生的問題，說明於圖1.1。圖1.1舉示了在一連串修正某大程式之後所發現的誤差[1]。分析與設計階段所產生的錯誤達到全體錯誤的64％。最重要的，這種錯誤非常難以檢出。此計劃的開發團體雖然能把寫碼錯誤的75％檢出來，但是，設計錯誤卻只能檢出30％而已。

軟體的可靠性曾是開玩笑的對象，但是，目前已經成爲嚴重的問題。

這種程式計劃錯誤曾經是茶餘飯後的有趣話題，佔去報紙或雜誌的許多篇幅。可是，如今錯誤已不是玩笑。在太空衛星程式的軟體錯誤，曾經使該太空的使命歸於失敗。軟體錯誤也讓人失去了做生意的機會。此外，軟體錯誤也曾使逮捕兇犯失敗，有時候竟然招致死亡。

程式計劃的過程雖然有許多可改善的餘地，但軟體的生產成本在過去20年並沒有減少多少。

硬體的成本雖降低了很多，但軟體的成本卻相對地呈繼續昇高的狀態。如果舉出在20年前所開發的適用業務，然後也作同樣的開發，也許人事費要差不多吧。

我們很少看到兩名程式設計者走到黑板，對於各種方案，判斷可靠性、複雜程度、成本、擴大性、維護性這方面的效果，並且爲了評價而互相討論的情形。所幸可以說經常看到的，就是設計上的決定是被團體的壓力（“多數決的原理”），“愚昧的迷信”（“一切程式必須有初期設計、處理、終止處理的部分”），個人的偏見（“我喜歡以這個方法來做”），感情（“按照我的作法做吧。否則的話……”）所決定的。

此为试读，需要完整PDF请访问：www.ertongbook.com