UNIDADE 04

Medidas Para Estrutura de Produtos de Software

META

Apresentar as principais medidas utilizadas para avaliar a estrutura de produtos de software.

OBJETIVOS

Geral

Capacitar o aluno a compreender e aplicar métricas estruturais em projetos de software.

Específicos

- Identificar os principais atributos estruturais de um produto de software;
- Compreender as propriedades e aplicações de métricas como complexidade, coesão, acoplamento e comprimento;
- Entender as medidas específicas em projetos orientados a objetos.

RESUMO

A unidade apresenta as principais medidas aplicadas à estrutura interna de produtos de software, com foco na avaliação de atributos como complexidade, acoplamento, coesão e comprimento. São abordadas diferentes perspectivas de análise, incluindo fluxo de controle, fluxo de dados e projetos orientados a objetos.

1. INTRODUÇÃO

Acredita-se que um produto de software bem projetado é caracterizado, em grande parte, por sua estrutura interna. De fato, a justificativa por trás da maioria dos métodos de engenharia de software é

garantir que os produtos de software sejam construídos com certos atributos estruturais desejáveis. Assim, é importante saber como reconhecer e medir esses atributos, pois eles podem fornecer indicadores importantes de atributos externos essenciais, como manutenibilidade, testabilidade, reutilização e até mesmo confiabilidade.

Nesta unidade, descrevemos como realizar medições daquilo que geralmente se acredita serem atributos estruturais internos essenciais, incluindo complexidade estrutural, acoplamento, coesão e modularidade. Esses atributos são relevantes tanto para documentos de projeto e seus modelos quanto para o código. De fato, ao conhecer esses atributos, podemos identificar componentes que provavelmente serão difíceis de implementar, testar e manter.

Para entender como medir a estrutura de produtos de software, começamos o estudo descrevendo aspectos das medidas estruturais com base na entidade de software a ser medida, seu nível de abstração e o atributo que se deseja medir. Em seguida, analisamos com mais detalhes a estrutura de fluxo de controle de software em funções, procedimentos e métodos individuais no nível de código. Na sequência, abordamos medidas baseadas na estrutura de fluxo de informação e de dados e métricas para projetos orientados a objetos.

2. CARACTERISTICAS DAS MEDIDAS DE ESTRUTURA

O tamanho de um produto de software diz muito sobre o esforço envolvido em sua criação. Considerando que todos os outros fatores sejam iguais, poder-se-ia presumir que um módulo grande leva mais tempo para ser especificado, projetado, codificado e testado do que um módulo menor. Porém, a experiência mostra que a estrutura do produto desempenha um papel relevante não apenas no esforço de desenvolvimento, mas também na forma como o produto é mantido. Assim, é importante investigar as características da estrutura do produto e determinar como elas afetam os resultados almejados. Para tal, é necessário representar a entidade de interesse em um modelo adequado

para análise dos aspectos relevantes do atributo de interesse.

A estrutura de produtos de software pode ser analisada, ao menos, por duas perspectivas: estrutura do fluxo de controle e estrutura do fluxo de dados. O fluxo de controle representa a estrutura na sequência na qual as instruções do programa são executadas. Por outro lado, muitas vezes, as operações aplicadas aos dados são mais complexas que as instruções que as implementam. Assim, a estrutura do fluxo de dados representa o percurso dos dados quando criado ou manipulado por um programa. As medidas de fluxo de dados descrevem o comportamento dos dados enquanto interagem com o programa.

Usaremos grafos para modelar produtos de software. Nesse modelo, um sistema é formado por um conjunto de módulos, no qual cada módulo contém um conjunto de elementos. Os elementos dos módulos são representados por nós do grafo. Há ligações entre os elementos dos nós, as quais são representadas por arestas no grafo. Um elemento pode aparecer em mais de um módulo e os módulos podem ser aninhados ou disjuntos. As medidas de estrutura serão definidas e avaliadas nesse modelo, aplicando as propriedades definidas por Briand et al (1996). Os atributos usados para caracterizar a estrutura de um produto de software são *complexidade estrutural, comprimento, coesão e acoplamento*. As propriedades definidas por Briand et al. para os atributos definem um relacionamento empírico e são condições necessárias, mas não suficientes para ser uma medição válida.

Complexidade estrutural do sistema captura a noção de quão complexa são as conexões entre os elementos do sistema. A complexidade do sistema depende do número de conexões entre os elementos e qualquer medida que capture essa noção: i) não deve ter sistema com valor negativo; ii) ter medida nula para sistemas sem conexões entre os elementos; iii) não depender de como as conexões são representadas; iv) ser monotônico, isto é, a complexidade de um sistema não pode ser menor que a soma da complexidade de 2 módulos disjuntos;

EXPLICA ATIVO

Um grafo é uma estrutura matemática usada para modelar relações entre objetos. Ele é composto por nós (ou vértices), que representam os elementos, e arestas, que representam as conexões entre esses elementos. Para os alunos não familiarizados com conceito, sugiro a leitura do livro Introduction to Graph Theory, do Robin J. Wilson.

e v) a complexidade de um sistema com módulos disjuntos deve ser a soma da complexidades dos módulos individualmente.

O comprimento é um atributo que representa a noção de distância entre elementos. Assim, uma medida do comprimento do software irá medir a distância entre as conexões dos elementos do sistema. Medidas de comprimento devem ter ao menos as propriedades: i) não tem comprimentos negativos; ii) sistemas sem conexões entre os elementos tem comprimento nulo; iii) comprimento não aumenta se incluir novas conexões a elementos já conectados; iv) comprimento não diminui caso se adicione conexões entre elementos não conectados; e v) o comprimento de um sistema composto por módulos disjuntos é igual ao maior comprimento entre os seus módulos. As quatro primeiras propriedades são semelhantes às da medida de tamanho, porém a quinta reflete a diferença entre esses dois atributos.

Acoplamento é um atributo de um módulo individualmente e depende das conexões desse módulo com elementos de módulos externos. Assim como a complexidade estrutural e o comprimento, medidas de acoplamento não devem ser negativas e módulos sem conexões com elementos de outros módulos deve ter acoplamento nulo. Adicionalmente, uma medida de acoplamento deve ser monotônica, ou seja, adicionar conexões do módulo para elementos externos não diminui o acoplamento; a junção de dois módulos cria um módulo cujo acoplamento será no máximo a soma do acoplamento dos dois módulos individualmente; e a junção de dois módulos disjuntos cria um módulo cujo acoplamento é a soma dos acoplamentos dos módulos individuais.

A coesão também é um atributo de um módulo individualmente, porém diz respeito a como os elementos internos do módulo estão relacionados. Medidas de coesão devem ser não negativas e normalizadas (variar num intervalo de 0 a 1). Bem como para medidas do demais atributos, um módulo sem conexões deve ter medida de coesão nula. Adicionalmente, adicionar conexões entre os elementos do módulo não

EXPLICA ATIVO

Um **bloco básico** é uma unidade

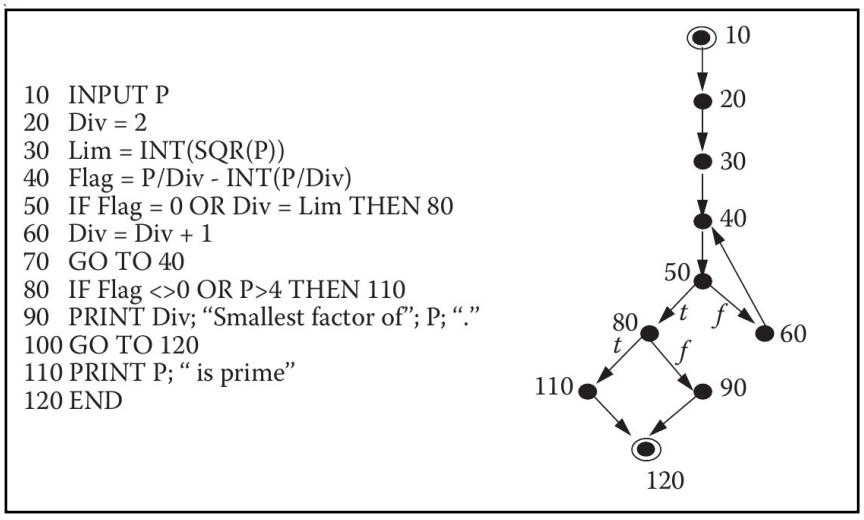
diminui a sua coesão e a junção de dois módulos não relacionados cria um módulo cuja coesão é no máximo igual a coesão do módulo de maior valor.

3. ESTRUTRA DO FLUXO DE CONTROLE DE PROGRAMAS

Medidas de fluxo de controle são, usualmente, modeladas com grafos dirigidos, nos quais os nós representam comandos ou blocos básicos e as arestas o fluxo de controle de um comando para outro. Esse tipo de grafo é conhecido como grafo de fluxo de controle, sendo um bom modelo para medidas da estrutura do fluxo de controle pois expõe muitas propriedades estruturais dos programas.

A Figura 1 apresenta um exemplo de um programa e seu correspondente grafo de fluxo de controle em que os nós denotam os comandos e as arestas mostram o fluxo do controle entre os comandos. No grafo de fluxo de controle, os nós iniciais e finais costumam serem diferenciados por um círculo em volta.

Figura 1- Exemplo de programa e seu grafo de fluxo de controle



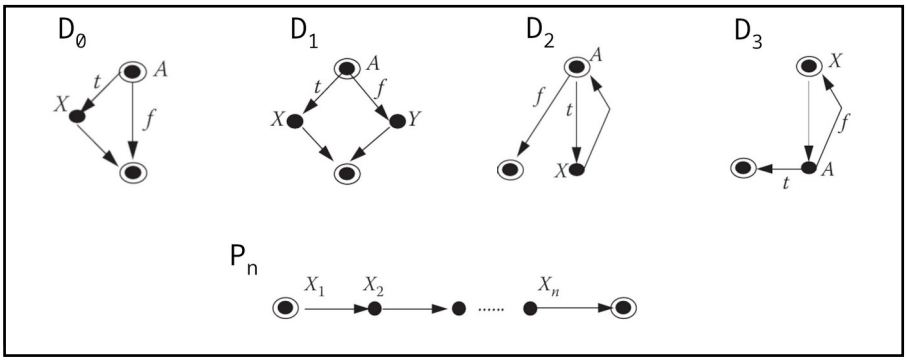
Fonte: Fenton (2014)

A análise discutida nesta Unidade da estrutura do fluxo de controle é fundamentada em fluxos de controle primitivos e operações de sequência e aninhamentos para formar grafos de fluxo de controle mais

fundamental na análise de programas. Trata-se de uma sequência contínua de instruções que são executadas de forma linear, ou seja, sem qualquer possibilidade de desvio interno — não há bifurcações, saltos ou chamadas que alterem o fluxo entre as instruções dentro do próprio bloco. Sua execução começa na primeira instrução e segue até a última, sendo que o controle só pode entrar no bloco por seu início e sair por seu fim. Essa característica torna os blocos básicos especialmente úteis para otimizações em compiladores e para a aplicação de métricas de software.

complexos a partir dos grafos primitivos. Os grafos de fluxo de controle primitivos são os apresentados na Figura 2. O grafo D_0 representa a estrutura de um comando if-then, D_1 o comando *if-the-else*, D_2 o comando *while*, D_3 o comando *do-while* e P_n uma sequência de *n* comandos.

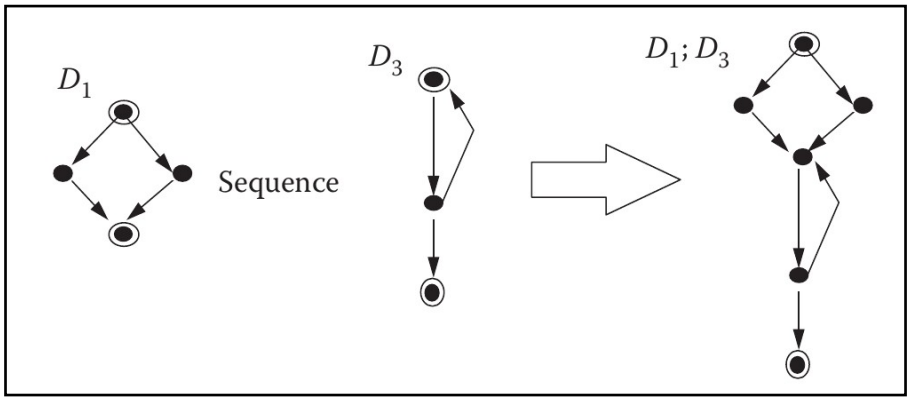
Figura 2- Grafos de fluxo de controle primitivos



Fonte: Fenton (2014)

Esses grafos primitivos podem ser combinados com as operações de sequência ou aninhamento para formar outros grafos de fluxo de controle mais complexos. A operação de sequência consiste na junção do nós de saída de um grafo de fluxo com o nó inicial do outro. Essa operação corresponde a concatenação de dois programas em linguagens imperativas. A Figura 3 ilustra a operação de sequência dos grafos de fluxo de controle D_1 e D_3.

Figura 3- Exemplo de aplicação da operação de sequência de D_1 e D_3

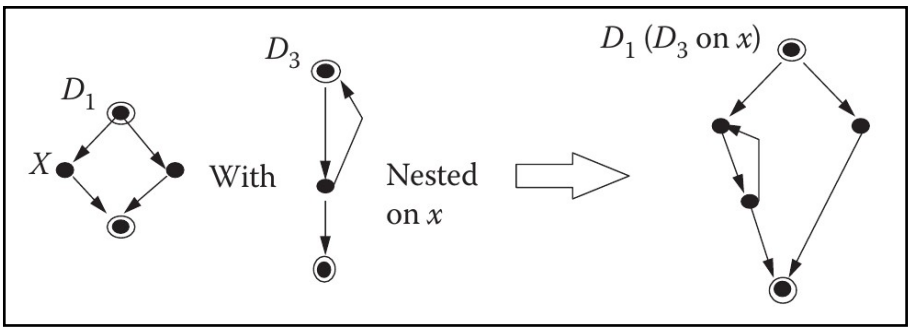


Fonte: Fenton (2014)

A operação de aninhamento é realizada em nós que contêm apenas uma aresta de saída, denominados de nós de procedimento. Nessa

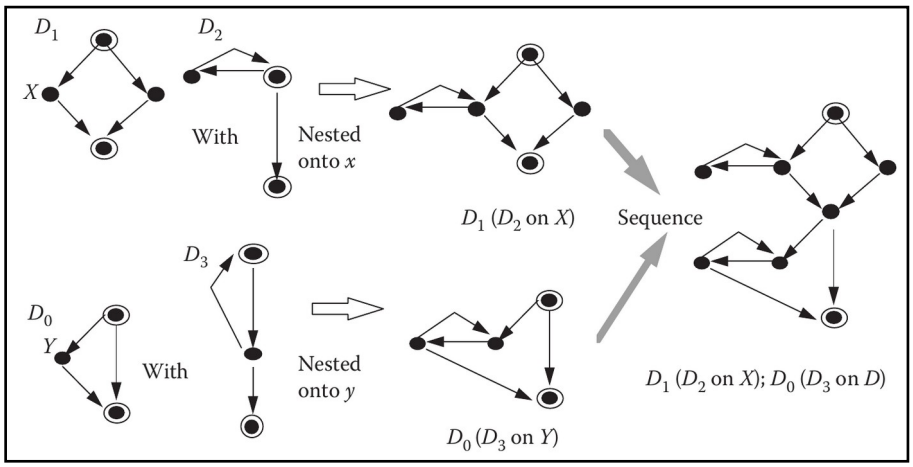
operação, um nó de procedimento do grafo de fluxo de controle é substituído pelo grafo de fluxo de controle aninhado. A Figura 4 mostra um exemplo da operação de aninhamento, em que o nó de procedimento X de D_1 é substituído por D_3. Uma sequência de operações de sequência e aninhamento para formar um grafo de fluxo mais complexo é apresentado na Figura 5.

Figura 4- Exemplo de aplicação da operação de aninhamento de D_1 e D_3



Fonte: Fenton (2014)

Figura 5- Combinação de operações de sequência e aninhamento



Fonte: Fenton (2014)

As medidas de estrutura de fluxo de controle serão definidas com bases na decomposição do grafo de fluxo de controle em grafos de fluxo de controle primitivos, pois a noção de programação estruturada é definida em termos da composição de programas a partir dessas construções primitivas (Böhm e Jacopini, 1966) e a decomposição de um programa em termos das operações de sequência e aninhamento dos

grafos de fluxo de controle primitivos é única (Fenton e Whitty, 1986).

A medição α mede a profundidade do aninhamento máximo dos grafos de fluxo de controle primitivos e é assim definido:

- $\alpha(P_1) = 0$ e $\alpha(D_i) = 1$;
- Sejam F_1 e F_2 dois grafos de fluxo, então α da sequência F_1 e F_2 é $\alpha(F_1 ; F_2) = max\{\alpha(F_1), \alpha(F_2)\}$;
- Sejam F_1 e F_2 dois grafos de fluxo, então α do aninhamento de F_1 e F_2 é $\alpha(F_1(F_2)) = 1 + \alpha(F_2)$.

Considerando o exemplo da Figura 5, tem-se que a medida α do grafo de controle de fluxo resultante, $D_1(D_2)$; $D_0(D_3)$ é 2. Essa medida indica que a profundidade máxima de um grafo de controle de fluxo primitivo é 2.

Note que a medição α é definida recursivamente, definindo-se o valor da medida para as primitivas e para as operações de sequência e aninhamento. Diversas outras medições foram propostas como medidas de complexidade estrutural e podem ser definidas de forma semelhante, sendo denominadas medições hierárquicas. Com essa representação, as medições hierárquicas podem ser computadas automaticamente.

Complexidade ciclomática é uma mede o número de caminhos lineares do grafo de fluxo de controle. Dado um grafo de fluxo de controle F, a complexidade ciclomática, ν , de F é:

$$\nu(F) = e - n + 2$$

Em que e é o número de arestas e n a quantidade de nós do grafo de fluxo de controle. A complexidade ciclomática também é uma medição hierárquica, podendo ser definida no mesmo arcabouço recursivo que a medição α como:

- $\nu(P_1) = 1$ e $\nu(D_i) = 2$;
- $\nu(F_1 ; F_2) = \nu(F_1) + \nu(F_2) - 1$;

CURIOSIDADE

Böhm e Jacopini demonstraram que qualquer algoritmo pode ser implementado somente usando estruturas de sequência (P_1), seleção (D_0) e iteração (D_2).

EXPLICA ATIVO

Fenton (2014) descreve um arcabouço para definição de medições hierárquicas. Apresenta, também, uma medida hierárquica de tamanho não ambígua.

- $\nu(F_1(F_2)) = \nu(F_1) + \nu(F_2) - 1.$

Dessa definição, podemos concluir que a “complexidade” dos programas básicos é a mesma e que a “complexidade” das operações de sequência e aninhamento é igual a complexidade da soma de seus componentes menos um. No entanto, não há consenso nessa noção intuitiva da complexidade ciclomática para ser considerada uma medida geral de “complexidade” de um programa. Porém, ela é uma medição muito valiosa para definir o qual difícil é para se testar um módulo.

Muitas das medições sobre estrutura enxergam o programa na perspectiva do quão complexo é entendê-lo. Porém, pode-se analisar a estrutura por outras óticas, em especial a do quão difícil é testar um programa. Nessa ótica, há medições que buscam medir o número mínimo de casos de testes para satisfazer a cobertura de uma dada estratégia de teste.

4. ATRIBUTOS DE PROJETO DE SOFTWARE

Algumas medições foram propostas para medir atributos de projetos de software e o relacionamento intermódulos. Nessa seção discutimos morfologia, impureza e acoplamento.

Morfologia é uma noção que se refere a forma geral da estrutura de um sistema quando representado graficamente e suas características podem ser usadas para descrever bons e maus projetos (Yourdon and Constantine, 1979). Projetos são representados usando grafos de dependência, nos quais os nós são conectados se houver dependência entre eles. As medições morfológicas são:

- Tamanho: medido pelo número de nós, arestas ou combinação de ambos;
- Profundidade: medição do maior caminho entre a raiz e uma folha;
- Largura: medição do maior número de nós em um nível;
- Razão de arestas por nó: medição de densidade que computo a

razão entre o número de arestas por nós.

Quanto mais a morfologia de um projeto se desvia de uma estrutura de árvore e se aproxima de um grafo, pior é considerado seu projeto. A impureza, ou tree impurity, é uma medição que indica o quanto a estrutura do projeto se desvia de uma estrutura de árvore. Podemos medir a impureza de um projeto como:

$$\frac{2(e-n+1)}{(n-1)(n-2)}$$

As medições *fan-in* e *fan-out* buscam medir o fluxo de informação que chegam e saem de um módulo (ou outra entidade de interesse), respectivamente. A métrica *fan-in* é calculada pelo número de informação que chega no módulo a partir dos módulos externos. De forma inversa, a métrica *fan-out* é calculada computando-se a quantidade de informação que sai do módulo para módulos externos. Essas duas medições costumam ser usadas como medidas de acoplamento.

5. MEDIÇÕES DA ESTRUTURA DE SOFTWARES ORIENTADOS A OBJETOS

Existem diversas medições propostas para medir atributos de softwares orientados a objetivos. Muitas delas não satisfazem as propriedades discutidas na Seção 1, assim restringimos a nossa discussão àquelas que satisfazem.

Message Passing Coupling (MPC) é uma medida de acoplamento que contabiliza o número de chamadas de métodos para classes externas. *Afferent Coupling* (C_a) e *Efferent Coupling* (C_e) são outras medições acoplamento de software orientado a objetos. C_a mede o número de classes de outros pacotes que dependem da classe analisada. C_e, por sua vez, mede o número de classes de outros pacotes que a classe depende. C_a é uma medida de *fan-in* e C_e de *fan-out* do pacote. Um alto valor de C_e torna a classe instável, pois depende de muitas classes externas. A seguinte métrica computa a instabilidade da classe:

SAIBA-MAIS

Chidamber and Kemerer (1994)

$$\frac{C_e}{C_a + C_e}$$

definem um conjunto de métricas orientadas a objetos (Métricas CK).

A coesão de um módulo depende dos relacionamentos existentes entre os elementos do módulo. Portanto, coesão é uma propriedade intramódulo. Em orientação a objetos, pode-se analisar a coesão de pacotes, classes ou métodos. A maioria das medições proposta são para coesão de classe, a qual mede o quão relacionados estão os elementos da classe (atributos, métodos e chamadas de métodos).

Tight Class Cohesion (TCC) e L*oose Class Cohesion (LCC)* são duas métrica de coesão de classe que medem a conexão entre os métodos via atributos. Dois métodos têm uma conexão direta se ambos acessam diretamente um mesmo atributo. Por sua vez, têm uma conexão indireta se um método compartilha um atributo via chamada a métodos que acessa diretamente o mesmo atributo. As medições TCC e LCC são definidas com base do número de conexões diretas (NDC), número de conexões indiretas (NIC) e número máximo de conexões (NP):

$$TCC = \frac{NDC}{NP} \quad \text{e} \quad LCC = \frac{NDC + NIC}{NP}$$

Depth of Inheritance Tree (DIT) é uma medição de comprimento que mede a distância máxima da classe até a raiz da hierarquia de classe. Essa medição informa o quão profundo é uma classe na hierarquia de herança.

6. CONSIDERAÇÕES FINAIS

Nesta unidade, abordamos as principais medidas utilizadas para avaliar a estrutura interna de produtos de software, fundamentais para a compreensão da complexidade de sistemas. Discutimos atributos como complexidade estrutural, acoplamento, coesão e comprimento. Apresentamos, também, métricas específicas para projetos orientados a objetos. Por fim, ressalta-se que não existe uma métrica única que irá expressar em um único valor todas as características da “complexidade de

software", como compreensividade, manutenibilidade, testabilidade e confiabilidade.

REFERÊNCIAS

BÖHM, C.; JACOPINI, G. *Flow diagrams, Turing machines and languages with only two formation rules*. Communications of the ACM, v. 9, n. 5, p. 366–371, 1966.

BRIAND, L. C.; MORASCA, S.; BASILI, V. R. Property-based software engineering measurement. IEEE Transactions on Software Engineering, v. 22, n. 1, p. 68–86, 1996.

CHIDAMBER S.R. and KEMERER C.F., A metrics suite for object-oriented design, IEEE Transactions on Software Engineering, 20(6), 476-498, 1994.

FENTON, Norman E.; BIEMAN, James. Software Metrics: a Rigorous and Practical Approach. 3. ed. Boca Raton: CRC Press, 2014.

FENTON, N. E.; WHITTY, R. W. *Axiomatic approach to software metrication through program decomposition*. Computer Journal, v. 29, n. 4, p. 329–339, 1986.

YOURDON, E.; CONSTANTINE, L. L. *Structured design*. Englewood Cliffs, New Jersey: Prentice-Hall, 1979.