Here are short notes, first from the authors who codified complexity classes and polynomial time reductions, and second from an earlier CS textbook which addressed polynomial time reductions.

"**Computers and Intractability" A Guide to the Theory of NP-Completeness**", Michael R. Garey and David S. Johnson, W. H. Freeman and Company, 1979 (twenty-seventh printing). [ISBN-10: 0-7167-1045-5]

Chapter 3 Proving NP-Completeness Results

To prove that $\prod \in$ NP-complete:

"Given a problem $\prod \in$ NP, all we need do is show that some already known NP-complete problem $\prod'$ can be transformed to $\prod$. Thus, from now own, the process of devising and NP-completeness proof for a decision problem $\prod$ will consist of the following four steps:

(1) Showing that $\prod$ is in NP,
(2) Selecting a known NP-complete problem $\prod'$,
(3) Constructing a transformation *f* from $\prod'$ to $\prod$, and
(4) Proving that *f* is a (polynomial) transformation." (page 45)

Notation: known NP-complete problem $\leq_P$ new problem

$\prod' \leq_P \prod$ (as expressed by Garey and Johnson)

$L' \leq_P L$ (as expressed in CLRS)

"**Fundamentals of Computer Algorithms**", Ellis Horowitz and Sartaj Sahni, Computer Science Press, 1978 [ISBN: 0-7167-8045-3]

Note: The Instructor has chosen to use the reduction symbol $\leq_P$ where Horowitz and Sahni used $\propto$ because Garey and Johnson defined the symbol $\leq_P$ for all modern usage.

Chapter 11 NP-Hard and NP-Complete Problems

"Definition: Let L_1 and L_2 be problems. L_1 *reduces to* L_2 (also written $L_1 \leq_P L_2$) if and only if there is a way to solve L_1 by a deterministic polynomial time algorithm using a deterministic algorithm that solves L_2 in polynomial time.

This definition implies that if we have a polynomial time algorithm for L_2 then we can solve L_1 in polynomial time. One may readily verify that $\leq_P$ is a transitive relation (i.e., if $L_1 \leq_P L_2$ and $L_2 \leq_P L_3$ then $L_1 \leq_P L_3$)." (page 511)