Domain specific languages (DSLs) have been proposed as a medium to capture the similarities and differences that occur in a family of software products. They are a mechanism for the representation, optimization, and analysis of such families.

Among the benefits to using domain specific languages is user-relevance. A significant advantage is gained by expressing application concepts in a language designed around those concepts. DSL designs are also user-modifiable. User-specified changes can be incorporated into fielded systems without compromising the integrity of the whole system. A DSL design of a component remains directly connected to the executing code. And last, DSLs provide a language with which design ideas can be communicated, and design principles taught.

A barrier to the widespread adoption of domain specific languages is the ability to systematically design, develop, and implement domain-specific languages. Good domain specific languages are mathematically-sound and are computer-readable. In 1998, Tim Sheard designed and taught a class about Domain Specific Languages to introduce students to the techniques used to design sound, and useable DSLs. It teaches the abstraction techniques used to make such designs easy to modify and evolve, and the implementation techniques used to embed such designs into an underlying system.

In winter quarter 2000, Tim redesigned the course and taught it again calling on new material from his own research and selected papers from the DSL workshop held in
conjunction with POPL, on January 18, 1997, the Usenix Conference on Domain
Specific Languages, held on October 15-17 1997, and the Usenix Domain Specific
Language Conference held on October 3-5 1999 in Austin Texas.

The class was structured around student led discussions of the literature, lectures by the
instructor, and projects where each student designed and implemented their own Domain
Specific Language.

A record of the course was captured by building and maintaining a web page with all the
notes, lectures, and student presentations. The page can be found at
http://www.cse.ogi.edu/~sheard/cse583W2000/index.html
Cover: The chart on the cover tracks the problem severity factor (PSF) of each of PacSoft's program transformation tools during the software development cycle. Low PSFs indicate that detected defects in the software have been fixed. This chart was constructed by Alexei Kotov, a graduate student for the Pacific Software Research Center.