A Case for Contemporary Literate Programming

Vreda Pieterse

Derrick Kourie

Andrew Boake

Department of Computer Science

Pretoria University

Agenda

- Introduction
- LP Essentials
- LP Environments
- Current Trends
- Conclusion

Introduction

- LP introduced by Knuth in 1984
  - Program: “explains solution” to computer (instructions)
  - LP: explains solution to human (document)
  - LP programs should be enjoyable to read
- Last 20 years:
  - Some recognition but not widely accepted
- Hypothesis: Now is the “kairos”
  - Technology is adequate
  - Need is strong
  - Time has arrived for LP

LP Essentials (1)

- Literate Quality
  - Programs as art / Pleasant for humans
- Psychological order
  - Ordered to enhance human understanding
  - Might be quite different from compiler needs
- Integrated documentation
  - Explanation to humans with “code comments” to be extracted for compilation

LP Essentials (2)

- TOC, index & cross references
  - Should be automatically generated
  - Should be easily searchable
  - Contemporary facility: hyperlinks
- Pretty printing
  - Colour-coded keywords / special fonts, etc.
  - Now commonplace
- Verisimilitude
  - Facilities to keep code and documentation in sync

LP Origins: Tex?

```
\section{The Algorithm}
The algorithm to be constructed assumes that
    the lattice $L$ has at least one element,
    $\top_{L}$, and that the element to be
    inserted into $L$ is a proper subset of
    $\top_{L}$. It is trivial to adapt the
    algorithm for cases where this does not
    apply.
The pre- and postconditions for the algorithm
    may therefore be stated as follows.
\begin{gcl}
\PROC insert(\ell)
\PRE \{isSICL() \wedge \ell \subset \top_{L}
    \wedge L^{'} = L\}
(L) : S
\POST \{isSICL() \wedge \ell \in L \wedge
    isMin(L^{'},\ell) \}
\end{gcl}
To elaborate statement $S$, two possible
    situations have to be considered. In the
    first case, the boundary case, $\top_{L}$
    may be the only element in the lattice
```

2 The Algorithm

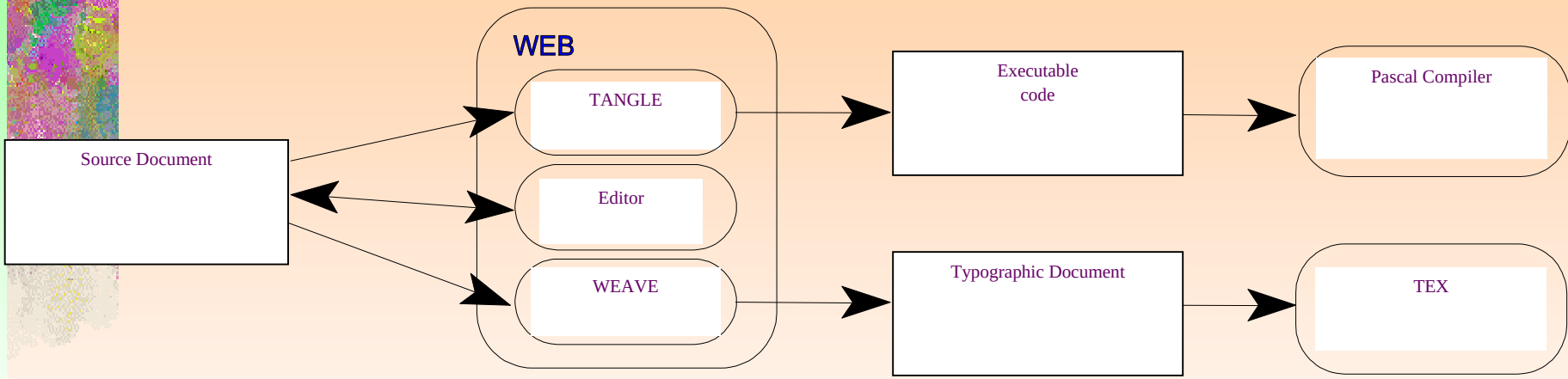
The algorithm to be constructed assumes that the lattice L has at least one element, $\top_L$, and that the element to be inserted into L is a proper subset of $\top_L$. It is trivial to adapt the algorithm for cases where this does not apply. The pre- and postconditions for the algorithm may therefore be stated as follows.

proc $insert(\ell)$
Pre : $\{isSICL() \wedge \ell \subset \top_L \wedge L' = L\}$
$(L) : S$
Post : $\{isSICL() \wedge \ell \in L \wedge isMin(L', \ell)\}$

To elaborate statement S, two possible situations have to be considered. In the first case, the boundary case, $\top_L$ may be the only element in the lattice

LP Environments (1)

- Pioneers: Language specific
  - Knuth's WEB in 1984



- Language Independency
  - LIPED (Bishop et al): Assembler, Clipper, Pascal
  - LEO (Ream): Java, C, C++, Pascal, Fortra

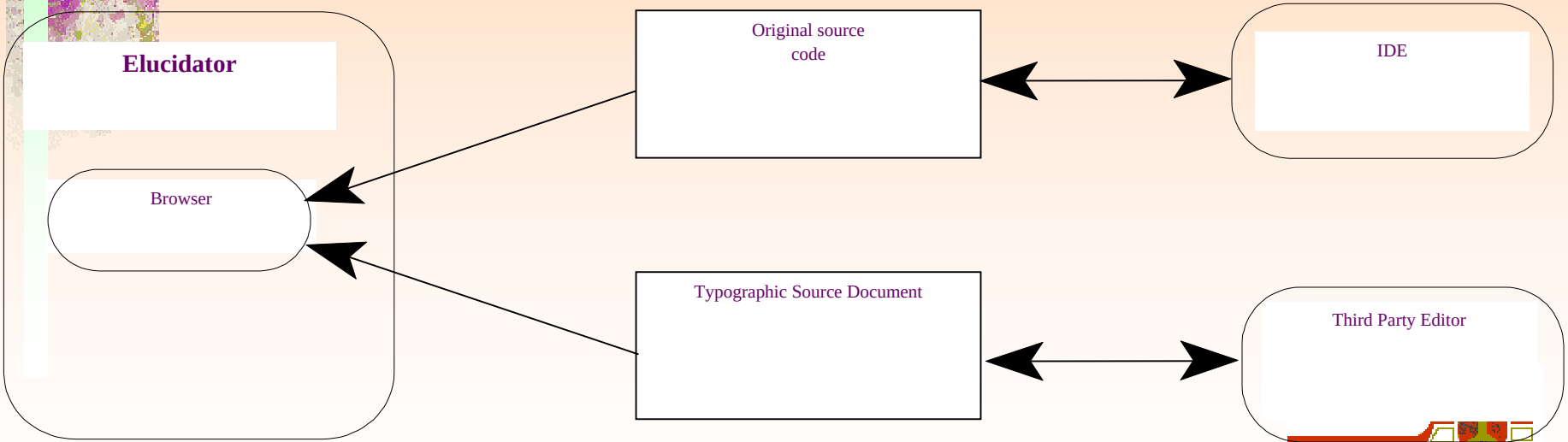
Universiteit van Pretoria

LP Environments (2)

- OO Programming
  - Yoyo problem: inheritance hierarchy
  - Browser for hypertext (ease navigation)
  - AOPS by Shum et al, 1993
- Wysiwyg
  - Main document in 3rd party editor
  - LPW by Lindberg, 1991
- Interactive coding / debugging
  - Address need to interact with code
  - Separates code and document
  - IDE eliminates need for code extractor
  - WARP by Thimbleby, 2003
    - Tool for journal publications

LP Environments (3)

- Elucidative Programming Environments
  - Separates code and doc
  - Supports program maintenance
  - Elucidator: Normark et al, 2000



Universiteit van Pretoria

LP Environments (4)

- Theme based LP
  - Use XML to create docs whose info can be presented in views to different audiences
  - CBDE by Kacofegitis et al, 2002
  - Atomic units: chucks
    - Code segment
    - Piece of documentation
    - Diagram
    - Unit test
    - Etc.

Trends (1)

- Documentation
  - Remains important
  - LP addresses mismatch problem
- Javadoc
  - Widely accepted
  - Half the battle won
- IDE Development
  - Pretty printing, navigation, verisimilitude between model & code, toc, etc
  - LPE needed to integrate narrative (eg to explain design rationale)

Trends (2)

- Event-driven programming
  - Behaviour in code difficult to document
  - Theme based LP: Define a theme for each event
  - Extract relevant info for online user help
  - Sync with actual implementation
- Design patterns
  - Standardization is an issue
  - LP can help in understandability and searching
- Portability
  - XML used for standards to interchange data among various tools
  - LPEs exist to implement XML technology to integrate tools

Trends (3)

- Agile methods
  - Scalability and outsourcing is a problem
  - LP can alleviate problems about team/project size
- Open source software
  - Concerns about code comprehensibility
  - LP can advance growth and success of OSS
- Product-line based SE
  - Involves large-scale reuse of artifacts / components
  - Difficult to determine component capability
  - LP-documented components would be beneficial
- Aspect oriented programming
  - Addresses functionality that cuts across system (power consumption, failure handling, security, etc.)
  - Highly reusable code
  - LP to support documentation

Conclusion

- Closing semantic gap between code / understanding
- Agile:
  - Self-documenting code
  - Pair programming / collective ownership
  - External doc – separately budgeted item
- However:
  - Inadequate for complex systems
  - Cartesian cleavage between code and doc is unprofessional
- Recognize resistance to documentation
  - Educational responsibility
- Future: Integrate LP features into IDE