Chapter 5

Software Tools

Introduction

- The demands of modern life require user-interface architects to build reliable, standard, safe, inexpensive, effective, and widely acceptable user interfaces on a predictable schedule
- User-interface architects must have simple and quick methods of sketching to give their clients a way to identify needs and preferences
- They need precise methods for working out the details with the clients, and for coordinating with colleagues
- Chapter outline:
  - Specification methods
  - Interface building tools
  - Evaluation tools

Specification Methods

- Design requires a good notation to record and discuss alternate possibilities:
  - The default language for specifications in any field is the designer's natural language, e.g., English
  - But specifications in a natural language
    - tend to be lengthy, vague, and ambiguous
    - difficult to prove correct, consistent, complete
- Formal languages have specified grammar to determine whether a string adheres to the language's grammar
- In graphical user interfaces, graphical specifications methods are more appealing

Grammars

- Grammars are useful to specify textual commands or expressions that a program should understand
- Still used in interactive systems that need powerful symbolic expressions, such as spreadsheet calculators

- *Backus-Naur Form* (BNF), also called *Backus Normal Form*, is often used to describe programming languages
  - High-level components are described as non-terminals
  - Specific strings are described as terminals

Grammars

- BNF Example

  <Telephone book entry>::= <Name><Telephone number>
  <Name> ::= <Last name>, <First name>
  <Last name> ::= <string>
  <First name> ::= <string>
  <string> ::= <character>|<character><string>
  <character> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
  <Telephone number>::= (<area code>) <exchange>-<local number>
  <area code>::= <digit><digit><digit>
  <exchange>::= <digit><digit><digit>
  <local number>::= <digit><digit><digit><digit>
  <digit>::= 0|1|2|3|4|5|6|7|8|9

- Examples of acceptable entries
  - WASHINGTON, GEORGE (301) 555-1234
  - BEEF, STU (726) 768-7878
  - A, Z (999) 111-1111

Grammars

- Attempts have been made to extend BNF with strategies for specifying sequences of actions, describing time delays, etc.
- **Multiparty grammars**: Non-terminals are labeled by the party that produce the string. e.g., in a login process:

  <Session> ::= <U: Opening> <C: Responding>

  <U: Opening> ::= LOGIN <U: Name>

  <U: Name> ::= <U: string>

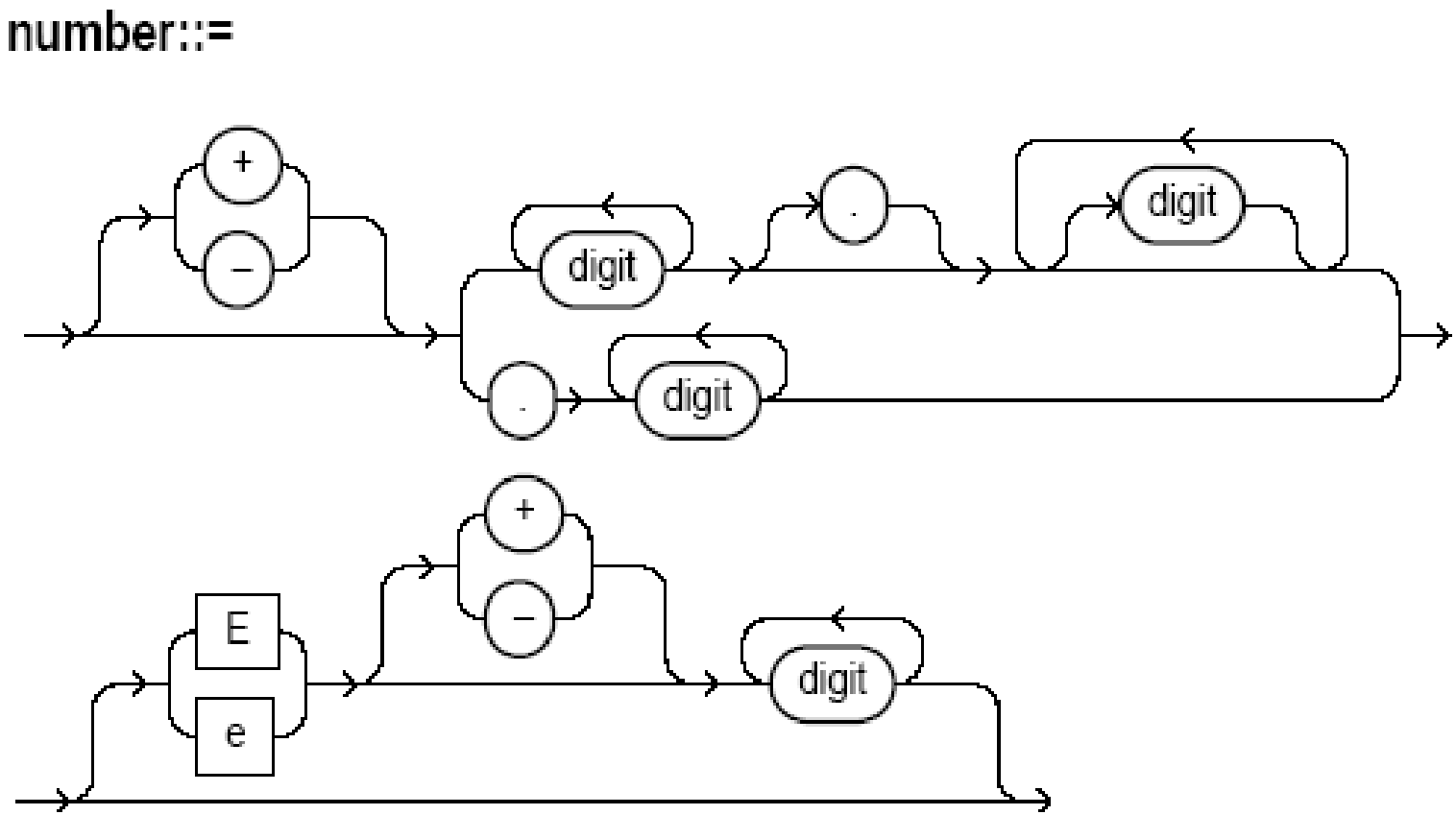
  <C: Responding> ::= HELLO [<U: Name]

  U: User C: Computer

- Multiparty grammars are effective for text oriented command sequences
- Describing form fill-in or direct manipulation and graphical layouts is difficult

Grammars

- Syntax diagrams



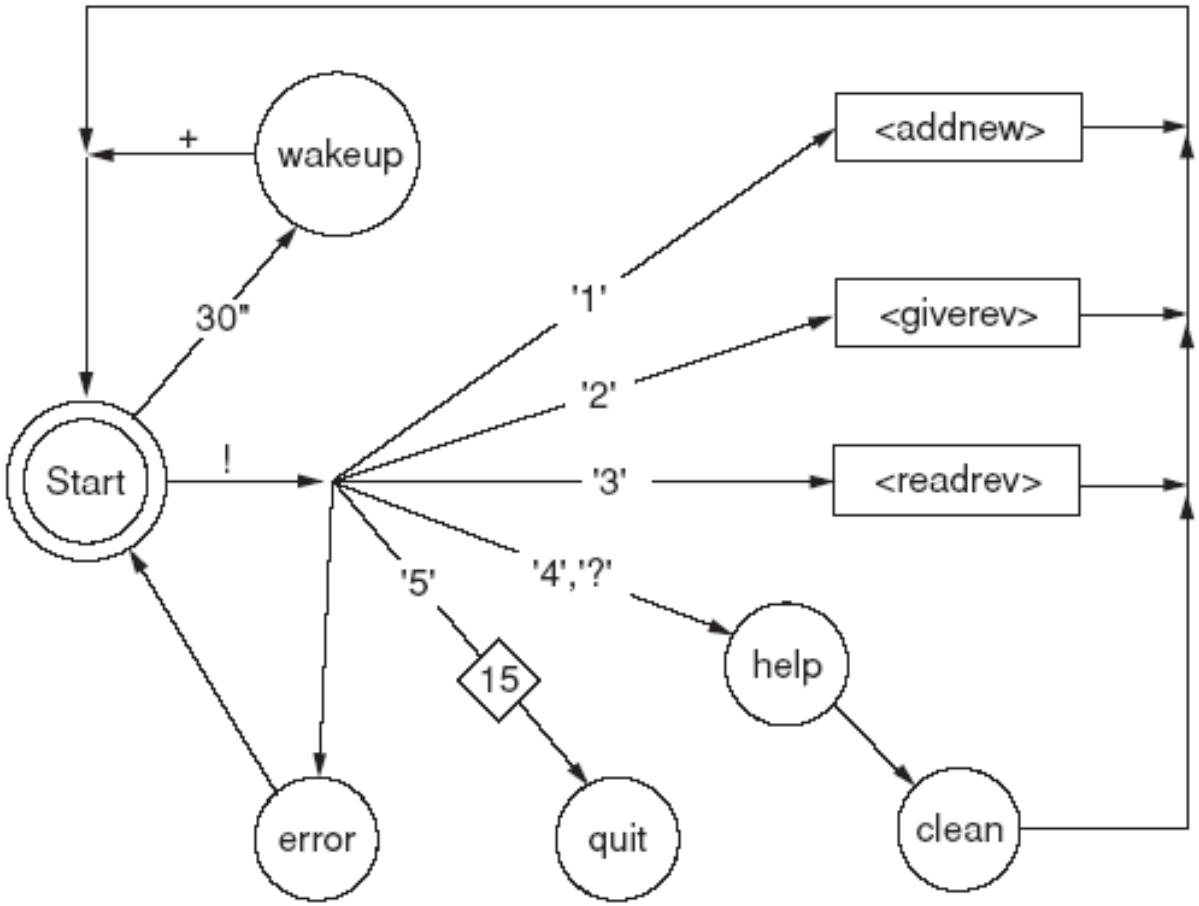
Not in the BOOK ☹

Menu-selection and Dialog-box Trees

- A *menu-selection tree* has a simple structure that guides designers and users alike.
- *Dialog-box trees* means printing the dialog-boxes and showing their relationships by mounting them on a wall or other surface.
- Both of the above are good specification tools to show the complete coverage of the system. They show high-level relationships.

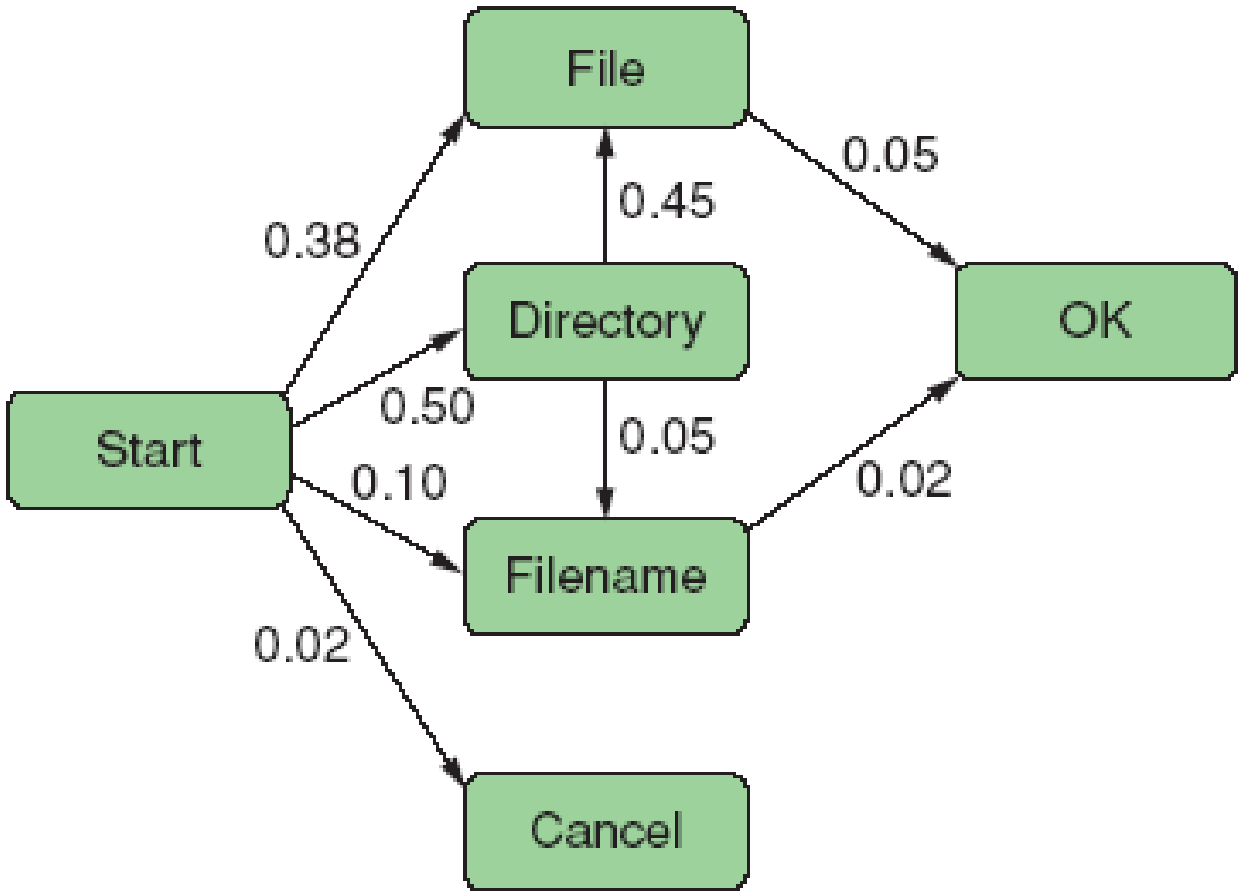
Transition Diagrams

- *Nodes* represent system states
- *Links* represent transitions.
- Links are usually labeled with user actions.

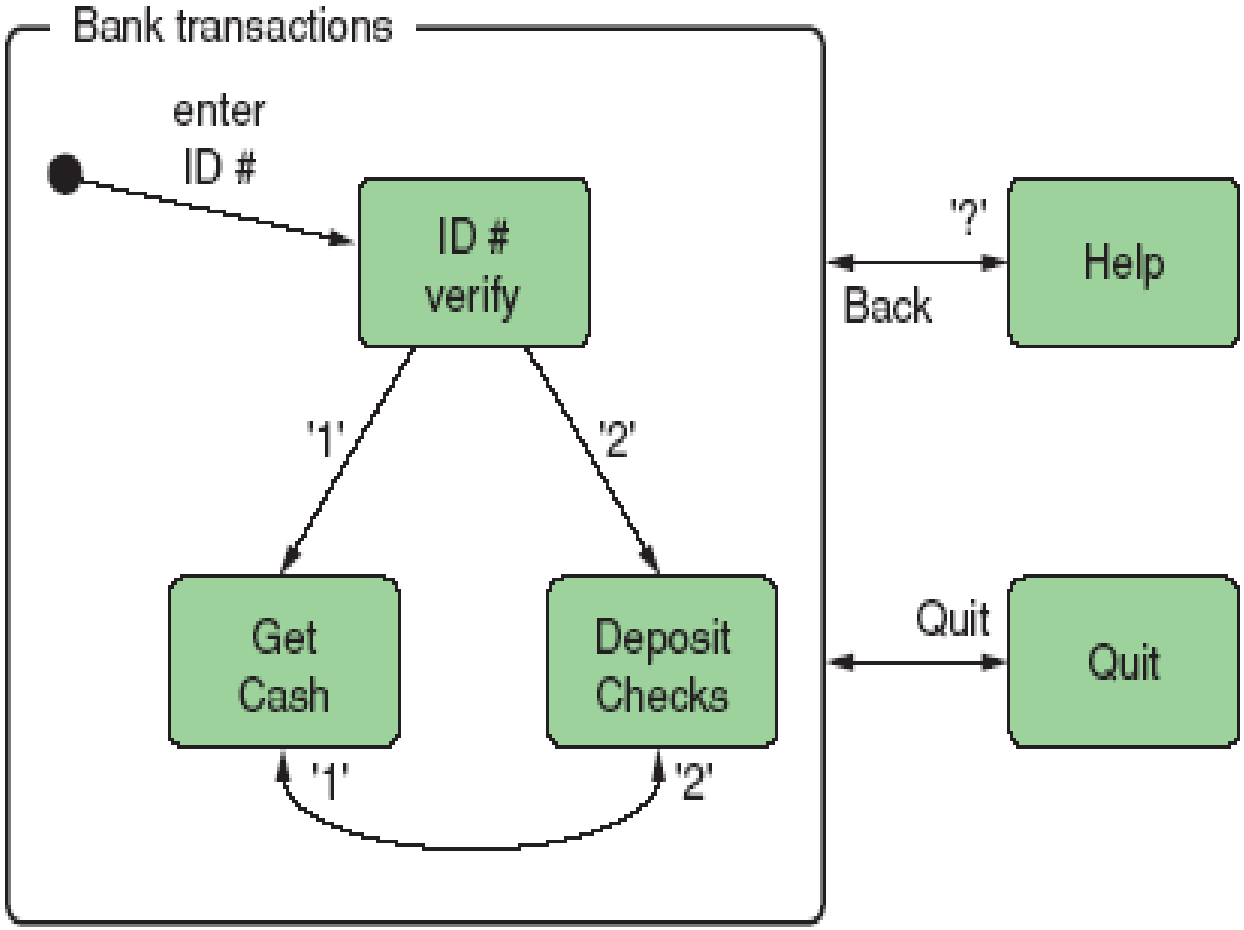


Transition Diagrams

- Transition diagram for file-manipulation actions. Link labels indicate how frequently each transition is made.



Statecharts



Note: Extended statecharts are skipped ☺

Interface-Building Tools

- Paper-based designs are a great way to start, but the detailed specification of complete user interfaces require software tools.
- Many software tools are available that emphasize convenient and rapid building of onscreen prototypes.
- These direct-manipulation tools enabled many task-domain experts who have only modest technical training to become user-interface designers.

Interface-Building Tools

- While choosing tools, important considerations are:
  - Do they support cross-platform development?
  - Do they allow the user interface to run under a web browser?
    - if yes, cross-platform goal is automatically met
- For non-desktop platforms (mobile, PDA, etc.) *up-to-date* high-level tools are usually not available. Designers have to use low-level tools

Interface-Building Tools

- The main advantage of the high-level software tools is that they support *user-interface independence* – the decoupling of user-interface design from the complexities of programming
  - This speeds up interface design, revision, expert review and usability testing
  - When the interface is stabilized, programming can be applied to complete the system
  - The prototypes can serve as specifications from which writers create user manuals and software engineers build the system

Interface Mockup Tools

- Examples
  - Paper and pencil
  - Word processors
  - Slide-show software
  - Macromedia Director, Flash mx, or Dreamweaver
  - MS Visio
- Visual Editing (Drag and Drop)
  - Microsoft Visual Studio
  - Borland JBuilder
  - Eclipse

Software Engineering Tools

- Experienced programmers build user interfaces with general-purpose programming languages such as, Java, C#, or C++, but using specialized tools for UI development becoming more popular
- Many tools are available. They belong to different layers.

Software Layers		Visual Tools	Examples
4	Application	Model-Based Building Tools	Microsoft Access, Sybase PowerDesigner
3	Application Framework/ Specialized Language	Conceptual Building Tools	Macromedia Director, Tcl/Tk, Microsoft MFC
2	GUI Toolkit	Interface Builder	Borland JBuilder Microsoft Visual Studio
1	Windowing System	Resources Editor	Windows Graphical User Interface Apple Quartz X11 Windowing System

Software Engineering Tools

- Finding the right tool is a tradeoff between six main criteria:
  1. Part of the application built using the tool.
  2. Learning time
  3. Building time
  4. Methodology imposed or advised
  5. Communication with other subsystems
  6. Extensibility and modularity

► Software Engineering Tools

Comparison between six features of design tools depending on the software layer

Layer	Part of the application built	Learning time	Building time	Methodology imposed or advised	Communication with other subsystems	Extensibility and modularity
4	All for a specific domain	Long	Short	Specification first, then visual, then programming (if required)	Very good for the specific domain of the tool	Very good
3	Presentation, interaction	Short (days)	Short	Visual first	Depends on the tool	Languages: Bad Frameworks: Good
2	Presentation	Long (weeks)	Long	Visual first with tools, none otherwise	Good	Medium/ good
1	All	Very long (months)	Very long	None	Very good	Very bad

Software Engineering Tools

- Many software architectures separate the user interface of an application from the internal functions. This makes it easier to make modifications to the interface without changing the internals.
- This is now a standard practice and it also facilitates cross-platform development
- However, one effect of this separation may be to postpone some of the usability issues to the end of development cycle.
- This may make the product less-usable because some of the usability problems cannot easily be solved late in the lifecycle; they need early planning.
  - E.g., “undo” facility, “canceling” actions, or progress bars

The Windowing System Layer

- Sometimes working at a low-level is required. e.g., new platform
-

```
main() {
   InitializeSystem();
   SetInitialState();
   DisplayInitialGraphics();
   while(true) {
        Event event = readNextEvent();
        switch(event.type) {
        case EVENT_REDISPLAY: redisplay(); break;
        case EVENT_PEN_DOWN: doPenDown(event.x, event.y); break;
        case EVENT_CHAR: doInputChar(event.detail); break;
        …
        default: doSystemDefault(event); break;
        }
   }
}
```

The GUI Toolkit Layer

- Most products provide user-interface program libraries called GUI toolkits that offer common widgets, such as windows, scroll bars, pull-down or pop-up menu, etc.
- Offer flexibility to experienced programmers
- Difficult to use without an interactive support
- Examples:
  - Java: JBuilder, Eclipse, NetBeans, etc.
  - .NET: .NET GUI Toolkit
  - Open Source Software: Gtk, Qt tookits

The Application Framework and Specialized Language Layer

- Application frameworks are based on object-oriented programming. It captures the common structure of UI and translates it to a set of classes and methods.
  - Can quickly build sophisticated interfaces
  - Require intensive learning
  - Examples: MFC, NextStep, and Cocoa
- Specialized language layers lighten the programming burden
  - Tcl (and its toolkit Tk)
  - Perl/Tk
  - Python/Tk
  - Visual Basic
  - Java Script

► The Application Framework and Specialized Language Layer

- Coupling of visual editor with scripting languages have become popular recently
- Examples:
  - Apple's HyperCard
  - Macromedia Director
  - Flash MX

Evaluation and Critiquing Tools

- Software tools are natural environments in which to add procedures to evaluate or critique user interfaces. e.g.,
  - Spell-checker, display and widgets count, etc.
- *Run-time logging software* are of great benefit to maintenance personnel and to revisers of initial design.

Evaluation and Critiquing Tools

- Tullis' Display Analysis Program (1988):
  - Takes alphanumeric screen designs and produces display-complexity metrics plus some advice:

Upper-case letters: 77% The percentage of upper-case letters is high.

Consider using more lower-case letters, since text printed in normal upper- and lower-case letters is read about 13% faster than text in all upper case. Reserve all upper-case for items that need to attract attention.

Evaluation and Critiquing Tools

- Maximum local density = 89.9% at row 9, column 8.
  Average local density = 67.0%
  - The area with the highest local density is identified...you can reduce local density by distributing the characters as evenly as feasible over the entire screen.
- Total layout complexity = 8.02 bits
  Layout complexity is high.
  - This means that the display items (labels and data) are not well aligned with each other...Horizontal complexity can be reduced by starting items in fewer different columns on the screen (that is, by aligning them vertically).

Evaluation and Critiquing Tools

- Web-page and web-site analyzers:
  - Bobby http://webxact.watchfire.com/
  - HTML Tidy http://sourceforge.net/projects/tidy

line 6 column 5 — Warning: <table> lacks “summary” attribute
line 10 column 22 — Warning: <img> lacks “alt” attribute
line 15 column 22 - Warning: <img> lacks “alt” attribute
line 86 column 17 - Warning: <table> lacks “summary” attribute
line 151 column 55 - Warning: unescaped & or unknown entity “&r” 5 warnings, 0 errors were found!

The table summary attribute should be used to describe the table structure. It is very helpful for people using non-visual browsers. The scope and headers attributes for table cells are useful for specifying which headers apply to each table cell, enabling non—visual browsers to provide a meaningful context for each cell.
The alt attribute should be used to give a short description of an image; longer descriptions should be given with the longdesc attribute which takes a URL linked to the description.

Evaluation and Critiquing Tools

- Doctor HTML - Web Page Analyzer:
  - http://imagiware.com/RxHTML

> Did not find the required open and close HEAD tag. You should open and close the HEAD tag in order to get consistent performance on all browsers.
> Found extra close STRONG tags in the document. Please remove them.