A Methodology for Analyzing the Temporal Evolution of Novice Programs Based on Semantic Components*

Chris Hundhausen, Jon Brown, Sean Farley, & Daniel Skarpas
Visualization and End User Programming Lab
School of Electrical Engineering and Computer Science
Washington State University
{hundhaus, sfarley, jbrown}@eecs.wsu.edu

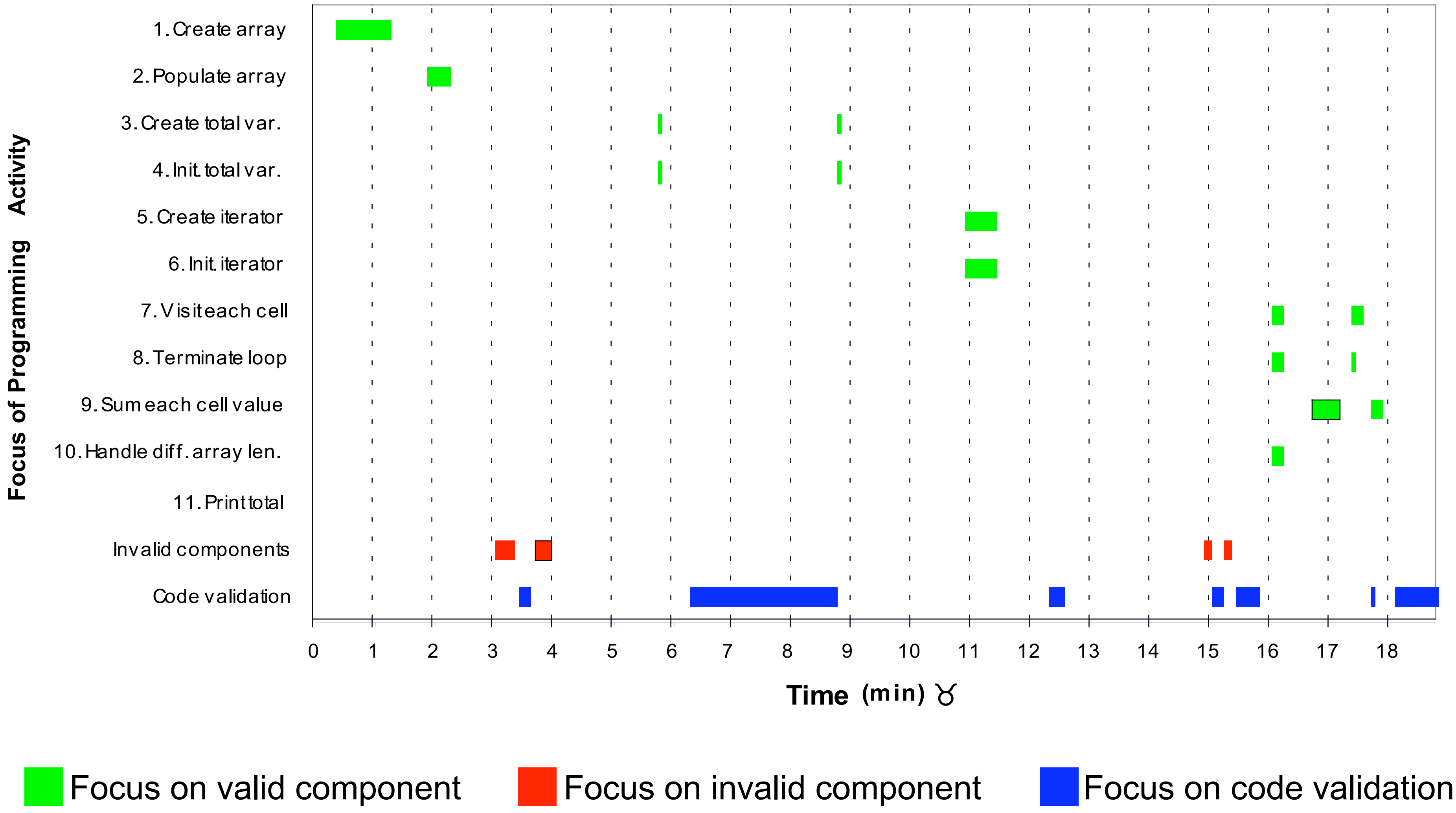
*This work is funded by the National Science Foundation under grant nos. 0406485 & 0530708

How Can We Build Better Novice Programming Environments?

- Plausible Answer: Understand (better) the ***programming processes*** promoted by those environments

- Key Research Questions That Relate to Programming Processes:
  - How do programmers spend their time within a given environment?
  - How does a novice program evolve over time within a given environment?
  - **How can a given programming environment assist a programmer in identifying, fixing, and avoiding syntactic and semantic programming errors?**

Illustration of the Kind of Analysis That Might Shed Light on Those Questions



We Present a Methodology for Gathering and Analyzing Video of Novice Programmers

- Why useful?
  - Basis for quantitative comparison of programming activities promoted by alternative novice programming environments
  - Basis for *timeline visualizations*, which provide qualitative feel for patterns of novice programming activities
- Remainder of Talk
  - Related Work
  - Overview of Methodology
  - Case Study
  - Summary and Future Work

Some Past Work Has Been Specifically Concerned with Methodological Issues

- Brooks, 1980
- Shneiderman, 1986
- Gilmore, 1990

But:

This work does not specifically address the issue of studying programming ***processes*** *for purposes of improving a programming environment*

Several Lines of Work Have Studied Programming Processes

- Goldenson & Wang, 1991 (Pascal Genie)
- Guzdial, 1993 (Emile)
- Jadud, ICER 2006 (BlueJ)

Our work differs from this work in two key respects:

- Human video analysis, as opposed to log files
- Characterization of programming processes based on breakdown of a code solution's semantic components

Our Methodology Builds on Three Established Methodologies

- Protocol Analysis (Ericcson & Simon, 1980)
  - Single participants verbalize their thought processes as they complete (programming) tasks
  - Participants' verbalizations are then analyzed in detail
- Sequential Analysis (Bakeman & Gottman, 1996)
  - Human behaviors or interactions are coded
  - Researcher looks for patterns in behavior
- Code Grading Based on a "Model Solution" Broken Into Semantic Components

Our Methodology Has Five Key Steps

- ✑✎Constructing model solutions
- ✒✎Making video recordings
- ✓✎Coding the recordings
- ✔✎Quantitatively analyzing the coding data in order to perform comparisons and to test hypotheses
- ✕✎Qualitatively analyzing the coding data by constructing and inspecting timeline visualizations

Step 1: Experts Construct Model Solution and Break into Semantic Components

Five Guiding Questions:

- ✑✎What *variable roles* must variables play in a correct solution?
- ✒✎To what values do variables need to be initialized?
- ✓✎Must the solution work for general input?
- ✔✎How must iteration proceed?
- ✗✎What are the lines of code in a model solution? (catch-all)

Step 3: Independent Analysts Code Video into Mutually-Exclusive Categories

We code activities directed toward…

- valid components of model solution (CS, CE, CI, and IVS codes)
- invalid components (IS, IE, ID codes)
- validating code correctness through explicit execution (VS and VE codes)

We are also interested in identifying points at which feedback aided…

- creation of valid component (FG code)
- removal of invalid component (FD code)
- creation of invalid component (FI code)

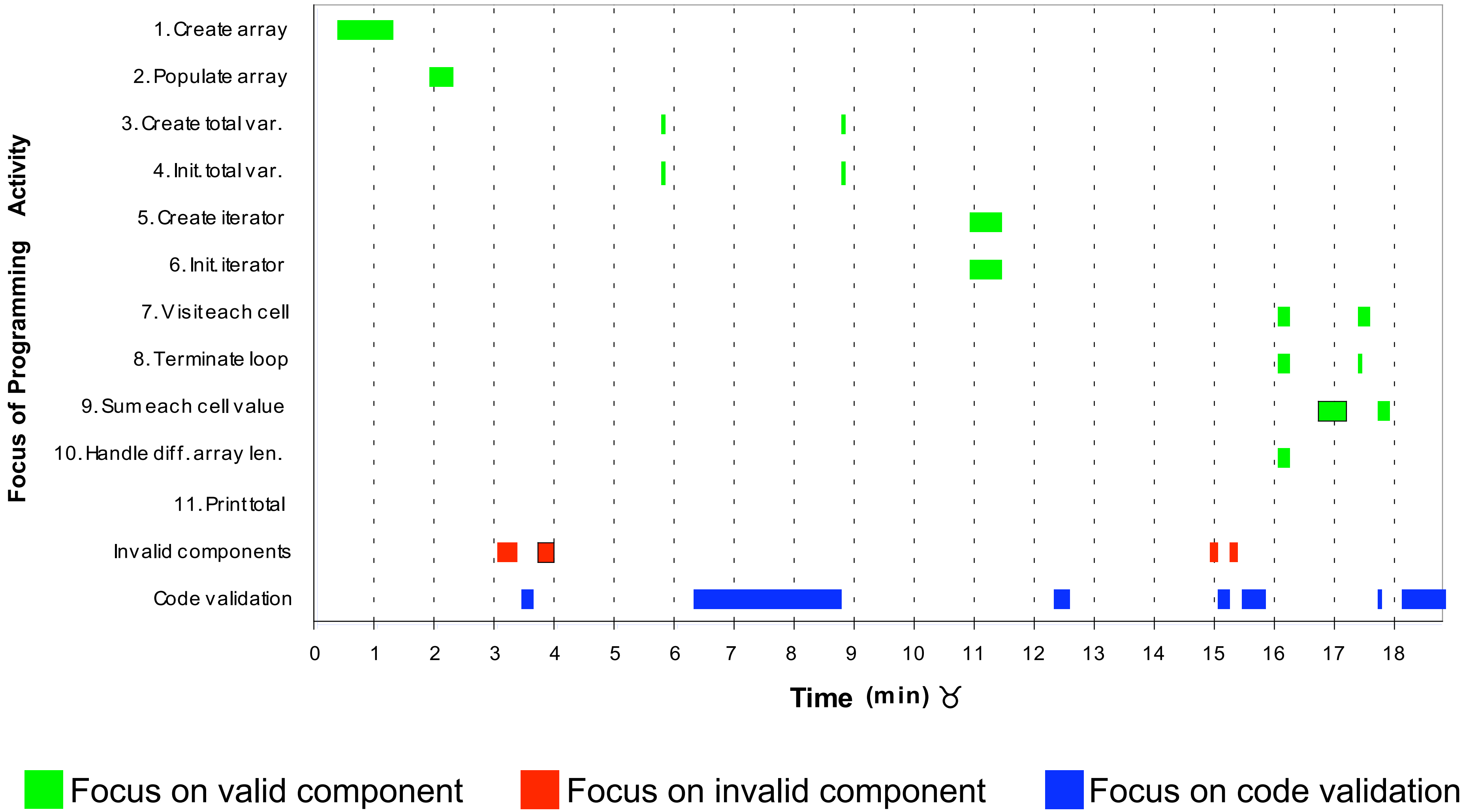
PC	SC	TC	Time	Comment
IS	F1		0:01:14	
ID	F1		0:01:53	set array array1 to index
CS	1		0:02:17	
CE	1		0:02:33	create array a1 with 6 cells
CS	2		0:02:48	
CE	2		0:03:11	populate a1 with random ints between 0 and 100
IS	F2		0:03:53	
IE	F2		0:04:09	create variable v1
IS	F3		0:04:25	
ID	F3		0:05:28	add a1[0] to v1
IS	F4		0:05:30	
IE	F4		0:06:09	while a1[
IS	F2		0:06:50	create variable v1
IE	F2		0:07:16	create variable sum
IS	F4		0:07:28	while a1[

Sample Coding Spreadsheet

Step 4: These Codes Generate Statistics That Help Answer Key Research Questions

Research Question	Supporting Statistics
Do participants spend their time focused on productive programming activities?	• *% dead time* • *% valid component editing* • *% invalid component editing time*
Are participants able to find and correct semantic errors in their code?	• *% invalid components deleted or fixed*
To what extent do participants explicitly validate their code's semantic correctness?	• *% validation time* • *Avg. # components validated per validation session* • *Average validation lag time*
To what extent does semantic feedback help or hinder coding progress?	• *% invalid components deleted or fixed via feedback* • *% valid and invalid components generated with the help of feedback*

Step 5: Coding Can Be Automatically Transformed into Timeline Visualizations



Case Study Illustrates Application of Methodology in Practice

- General Research Questions
  - Can semantic feedback benefit novice programmers?
  - If so, what form is best?
- We experimentally compared three alternatives
  - Automatic feedback
  - On-demand feedback
  - No feedback (control treatment)
- 35 novice programmers recruited out of CS 1 course at WSU
- Participants wrote SALSA solution to “Compute Sum” task in one of three experimental versions of ALVIS novice programming environment

"Compute Sum" Model Solution Included 11 Semantic Components

- Create array
- Populate array
- Create (role of) total
- Initialize (role of) total
- Create (role of) iterator
- Initialize (role of) iterator
- Looping visits each cell
- Looping terminates correctly
- Add cell value to total
- Iteration handles variable-length arrays
- Print total

We Collected and Coded 19.6 Hours of Video

- Three analysts independently coded a random 20% sample (1,602 observations)
  - Achieved 94.4% agreement, 0.936 kappa
- Once reliability was established, we divided the remaining video evenly across the three analysts
- Entire process required…

  …two weeks of training per analyst

  …2 – 4 hours to code each hour of video

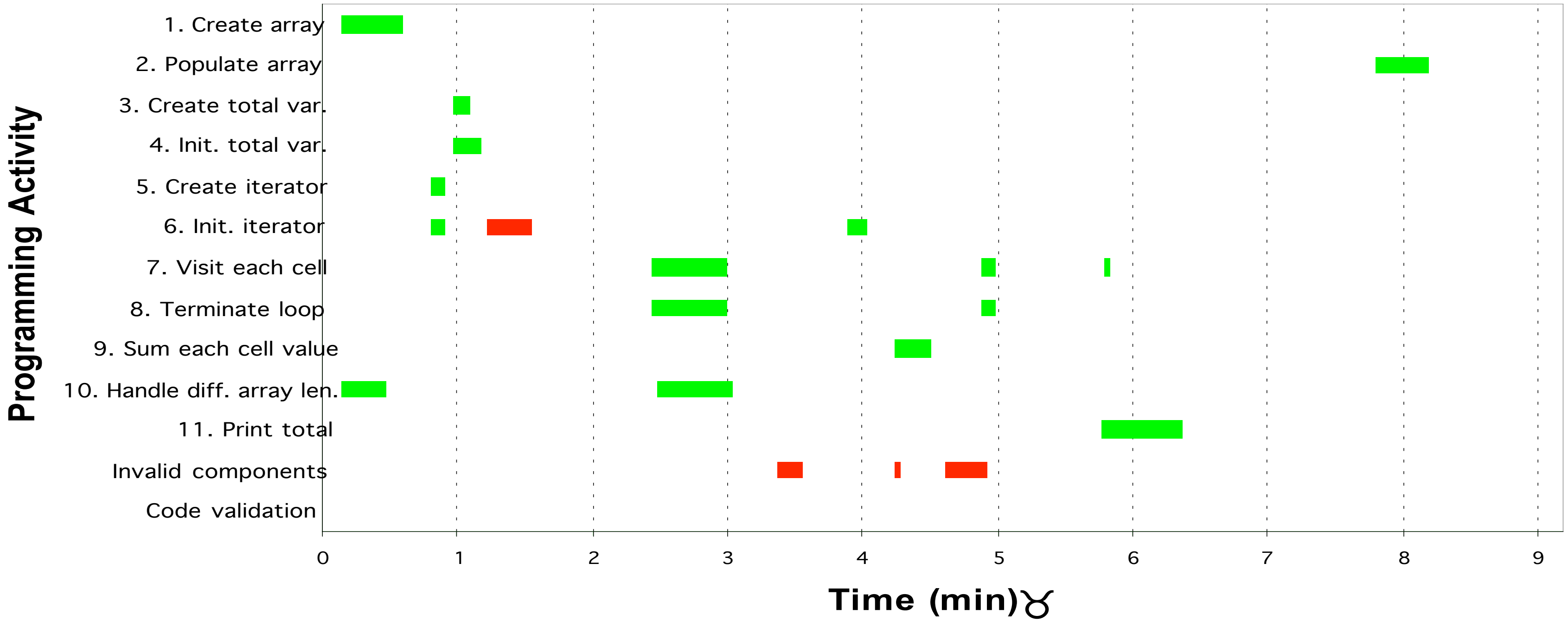
Feedback Conditions Achieved Higher Accuracy on Some Semantic Components…

Measure	Treatment	*Mean*	*Std. Dev.*	KW *p*-value
Total (out of 11 points)	Automatic	9.3	2.3	0.102
	On Request	7.9	3.4	
	No Feedback	5.5	4.0	
SC #5, 6 (Create/Initialize Role of Iterator)	Automatic	0.92	0.29	**0.019**
	On Request	0.82	0.40	
	No Feedback	0.42	0.51	
SC #7, 8 (Visit Each Cell, Terminate Loop Correctly)	Automatic	0.75	0.45	0.053
	On Request	0.55	0.52	
	No Feedback	0.25	0.45	
SC #9 (Add Each Cell to Total)	Automatic	0.83	0.39	**0.032**
	On Request	0.72	0.46	
	No Feedback	0.33	0.49	

...But Higher Accuracy Appears More Related to Persistence than Feedback

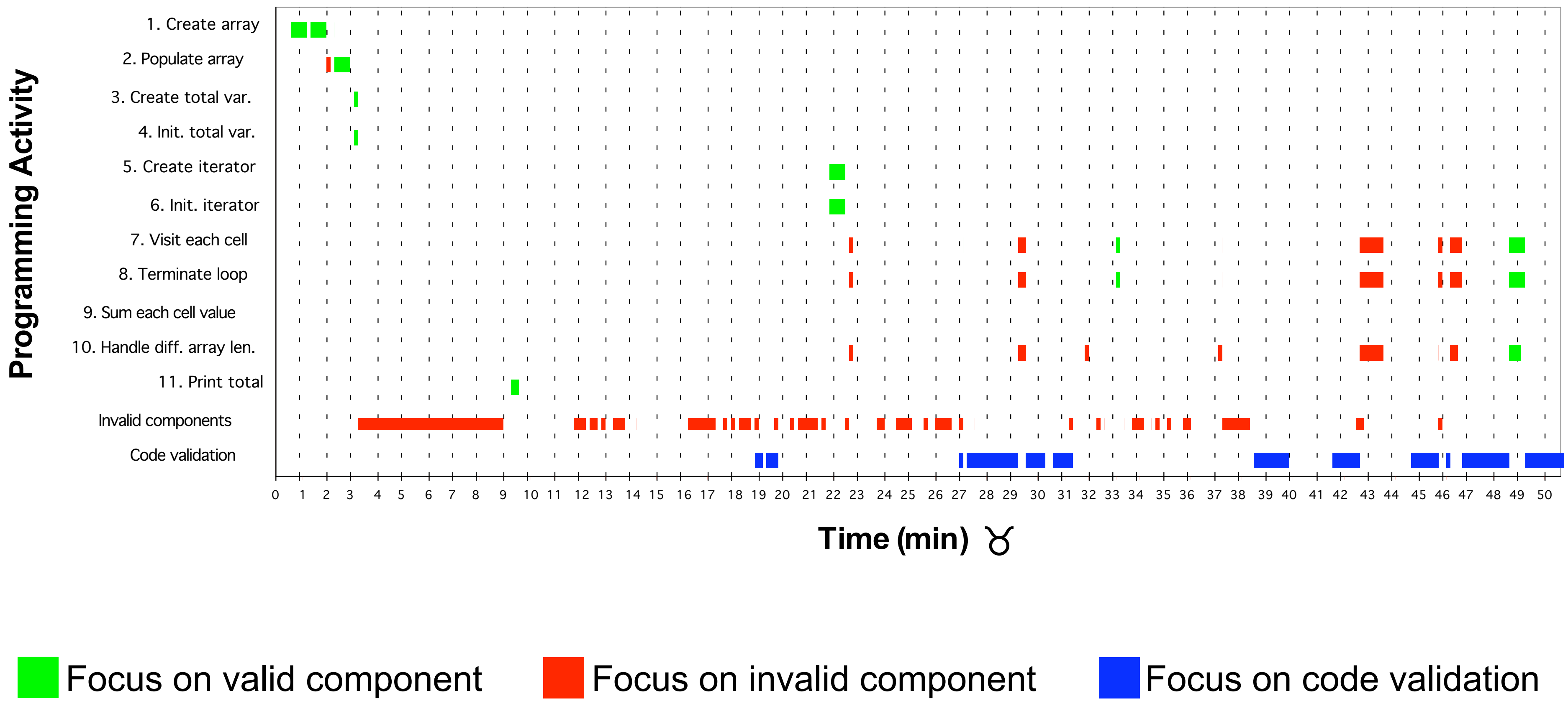
Measure	Treatment	*Mean*	*St. Dev.*	KW *p*-value
Time On Task (min.)	Automatic	45.6	40.1	0.057
	On Request	39.5	39.8	
	No Feedback	16.1	9.8	
% Valid Component Editing Time	Automatic	11.4	7.4	0.277
	On Request	11.8	11.3	
	No Feedback	16.5	9.5	
% Invalid Component Editing Time	Automatic	34.4	13.7	0.250
	On Request	21.7	18.8	
	No Feedback	28.0	21.1	
% Invalid Components Deleted/Fixed	Automatic	90.7	12.3	0.312
	On Request	78.5	30.5	
	No Feedback	61.5	42.3	
% Invalid Comp. Deleted/Fixed via Feedback	Automatic	9.7	19.4	0.312
	On Request	11.2	9.0	
	No Feedback	0	—	

“No Feedback” Participant Succeeds with Few Missteps

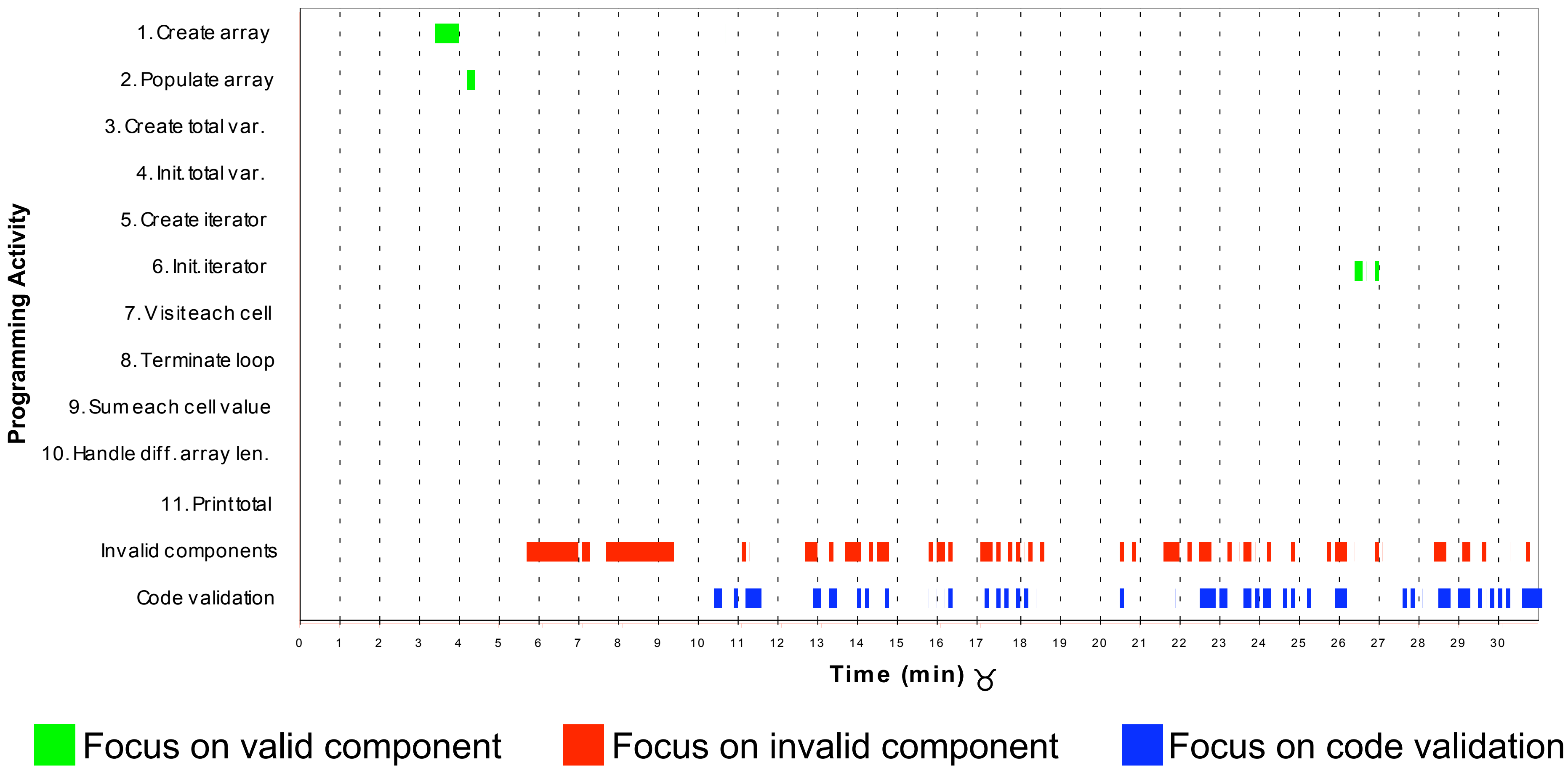


Focus on valid component   Focus on invalid component   Focus on code validation

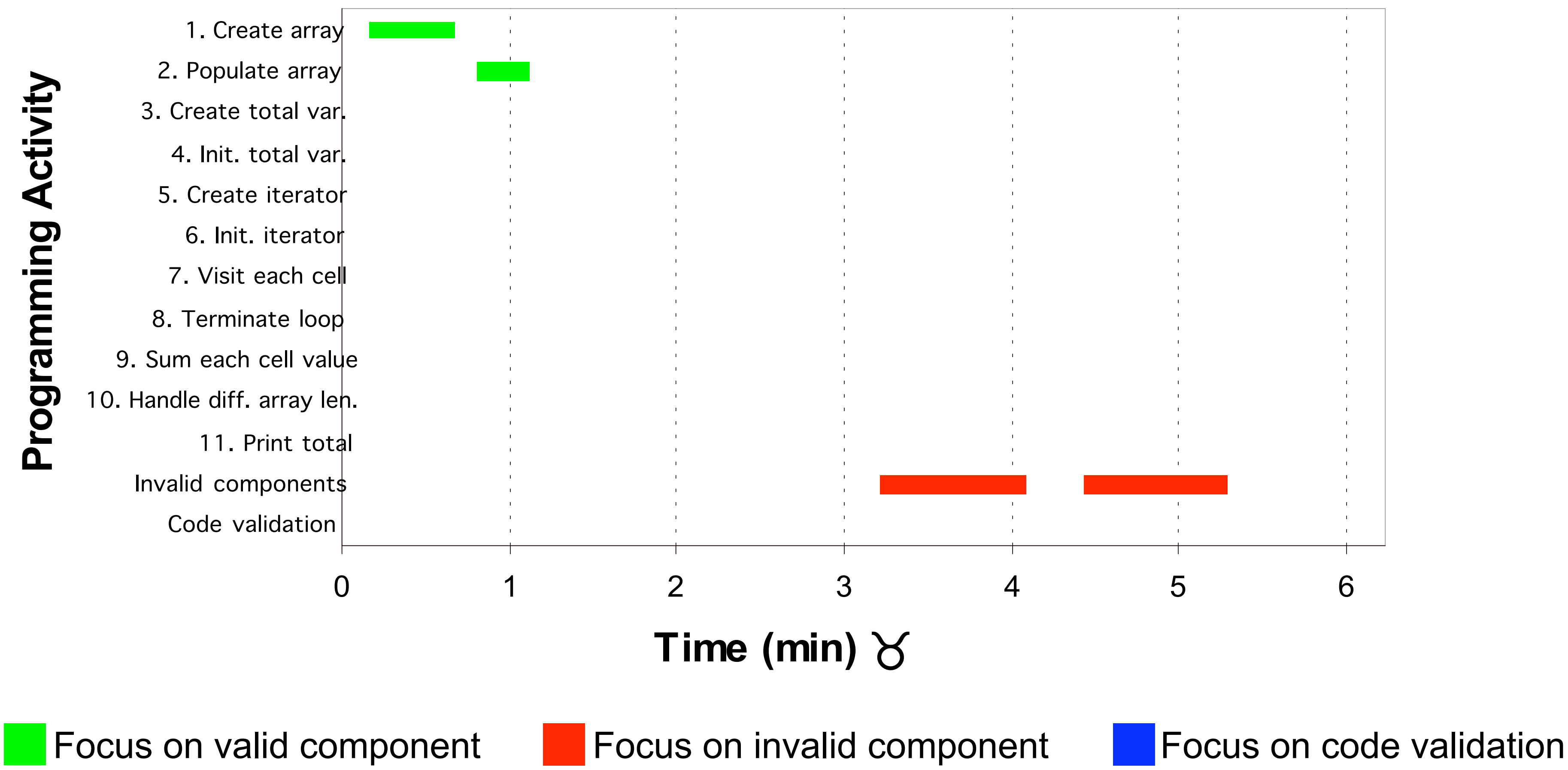
"Automatic" Participant Succeeds through Persistence



“On Request” Participant Cannot Get On Track Despite Honest Effort



"No Feedback" Participant Gives Up Quickly



We Have Presented a New Methodology for Analyzing Novice Programming Processes

- Novelty
  - Frames programming activity in terms of time-ordered sequence of editing episodes focused (or not) on semantic components of model solution
- Strengths
  - Shows contribution of each editing episode to final solution
  - Provides empirical basis for comparing novice programming environments
- Limitations
  - Development of model solution may be difficult for more complicated algorithms
  - Requires substantial investment of time and labor (but could be partially automated)
  - Says nothing about nature of invalid components (but could extend coding scheme to classify semantic errors based, e.g., on Spohrer and Soloway, 1986)

Questions?

For further information, and to download the ALVIS software, visit the Visualization and End User Programming Lab (VEUPL) website:

http://eecs.wsu.edu/~veupl

VEUPL