Schema Creation in Programming

Robert S. Rist
Cognitive Science, 13, 389-414 (1989)

Summary by Prof. Thomas LaToza
SWE 795, Spring 2017
Software Engineering Environments

GEORGE MASON UNIVERSITY

Schema Creation in Programming

- Developers use plans to reason about and create programs.

- In forward design, developers simply apply the appropriate plan for the task at hand.

- What happens when such a plan is not present?

Design methods

- Developers design programs front to back
- Hierarchically decompose problem into subproblems
- Experts use knowledge to sketch out high-level plan first (breadth first design)
- Novices that lack this knowledge must explore each sub piece before realizing next must be built (depth first design)

Protocol Showing the Creation of a Complex Plan from Retrieved Basic Plans:
All Basic Plans Show Top-down or Forward Plan Expression

Verbal Statements	Program Code		Interpretation	
(N2) "The first thing I want to do is to get the original word...You might want to use an array for the letters in the word, so you have...			goal	
	TYPE letters=array[1..20] of char;			
	VAR word: letters;		define word	
Now get the original word, so say WHILE...use a REPEAT loop...give a prompt first...	write ('Please enter a word');		extension	I_{read}
then REPEAT...	repeat		extension	I_{loop}
umm...you want to read it in...	read (...		focus	$C_{read,\ loop}$
you want a counter...			goal	
initialize a counter,	i:=0;	[before repeat]	extension	I_{count}
and then you want to repeat	read (word[i]);		use	O_{count}
...so you read it in, and then let i...	i:=...		focus	C_{count}
...initialize i to 1	i:=1;	[before repeat]		
...and then increment it by 1	i:=i+1;			
...uh-uh, until..."	until (word[i]=' ');		use	O_{read}

Schemas include separate slots

Basic Pascal Plan Schemas

Plan Creation: / Plan Retrieval:	Extension / Initialization	Focus / Calculation	Goal / Output
Prompt	write ('Enter . . . ');	read (*number*);	value of *number*
Label	write ('Output is . . . ');	write (*number*);	display
Running total	count: = 0;	count: = count + 1;	value of *count*
Found	found: = false;	if *condition* then found: = true;	value of *found*
Loop	for i: = 1 to 30 do	read (rainfall);	all rainfall values

Results

- Developers used bottom up process to initially create plans

- As developers gained experience in plan, retrieved plan top down rather than implementing plan bottom up

- Reusing plans in a novel context led participants back to bottom up plan design process

Questions for discussion

- Overall reaction to the paper

- How would these results apply to larger programming tasks?
- Or tasks by non-novices?