

# Using Sex Differences to Link Spatial Cognition and Program Comprehension

Maryanne Fisher  
Department of Psychology  
Saint Mary's University  
Halifax, Nova Scotia, Canada  
mlfisher@smu.ca

Anthony Cox  
Faculty of Computer Science  
Dalhousie University  
Halifax, Nova Scotia, Canada  
amcox@cs.dal.ca

Lin Zhao  
Faculty of Computer Science  
Dalhousie University  
Halifax, Nova Scotia, Canada  
lzhao@cs.dal.ca

## Abstract

*Spatial cognition and program development have both been examined using contrasting models. We suggest that sex-based differences in one's perception of risk is the key to relating these models. Specifically, the survey map approach to navigation and the top-down development/comprehension strategy use similar and related high risk cognitive skills that males show a preference towards. Conversely, the route-based approach to navigation and the bottom-up development/comprehension strategy use similar and related low risk cognitive skills that women show a preference towards. On the assumption that programmers are consistent in their risk-taking behaviours, we believe that they will, as much as possible, tend to use the same strategy when performing program development and comprehension. In an experimental setting, we compare programmer's performance on spatial cognition and program comprehension tasks. The correlations that we found suggest that programmers use equivalently risky strategies for program comprehension and spatial cognition. Thus, there is evidence that similar cognitive skills are used for spatial cognition and program comprehension/development, and that the similarities are a consequence of sex-based differences in risk-taking behaviour.*

## 1 Introduction

Program comprehension, reading a map, and driving an automobile are tasks that, at first, appear to have little in common. However, it is very unlikely that we have developed a highly specialised skill set for each of these tasks. Instead, as a result of evolutionary pressures, we have developed a small set of widely adaptable, general purpose cognitive skills and structures [5]. When given a new activity that requires a cognitively complex skill set, the needed skills are generated by adapting and combining existing skills [4]. Furthermore, mechanisms and structures that have proved

useful in one domain will be used in similar domains and will be changed only when driven by the need for performance improvement [4].

As the need for program comprehension skills has developed only recently, in evolutionary terms, these skills must result from the 'new' application of existing skills. We postulate that some of the skills used during program comprehension overlap significantly with those used for spatial cognition. In other words, we equate the navigation and exploration of source code, an abstract virtual space, with our navigation of the world in which we live, a three-dimensional physical space. A rationale for equating the application of real-world navigation skills with those used in virtual spaces has been suggested by Vinson [35].

In previous research [7], we used the term *codespace* to describe the abstract mental representation that programmers form for a software system. We now further refine this definition and define codespace as:

A programmer's mental model of source code with respect to the perceived spatial attributes of entities identified within the code.

We focus on source code, as opposed to other representations (e.g., flow graphs, class hierarchy diagrams), as these alternative representations can be considered as abstractions of the source code. We also consider other code-like artifacts (e.g., build scripts) as part of a system's source code.

Support for this definition is provided by Green [13], who postulates that "mental representations of programs seem to use spatial imagery where possible." Evidence also suggests that, when possible, programmers use spatial imagery as a coding for program concepts [14]. Green and Navarro found that program comprehension was facilitated when related program concepts occurred in close physical proximity [14]. Douce *et al.* [11] suggest that comprehension and maintenance substantially use programmer's spatial abilities and that the source file can be viewed as *program space*.

Movement by a programmer, within and between the

source artifacts of a system, during maintenance and development tasks has been considered as *software navigation*. Kersten and Murphy [18] observe that, “programmers tend to spend more time navigating [the] code than working with it.” To support software navigation, a variety of tools (e.g., Mylar [18], NavTracks [31]) and visualisations (e.g., terrain maps [9], fluid views [10]) have been developed. As well, spatial complexity has been examined as a metric of software complexity [11, 24].

Thus, there is a well-documented relationship between spatial cognition and a programmer’s mental modeling and navigation of a system’s source code. However, the skills that are used must be adapted for the specific environment as differences also exist. For example, one can not ‘teleport’ to a new location in the real world in the same way that ‘jumping’ to a search target can be performed in codespace. The existence of virtually instantaneous movement illustrates that there are differences between codespace and the real world. However, these differences do not prevent the same skills from being used in both environments. Instead, the differences suggest that a common subset of skills is used in each world and that this subset is augmented and modified to deal with the unique aspects of each. Cosmides and Tooby [6] support this view in their *swiss army knife hypothesis*.

Spatial cognition, program development, and program comprehension have been examined using contrasting approaches. Program development/comprehension is performed using top-down and bottom-up approaches while navigation, a key element of spatial cognition, uses route-based and survey map approaches. These approaches are examined in more detail in Sections 2 and 3. We suggest that a sex-based difference in one’s risk-taking behaviour, as examined in Section 4, is the key to relating these approaches. Specifically, the survey map approach to navigation and the top-down development/comprehension strategy use similar and related high risk cognitive skills that males show a preference towards. Conversely, the route-based approach to navigation and the bottom-up development/comprehension strategy use similar and related low risk cognitive skills that women show a preference towards. These relationships are examined in Section 5. On the assumption that programmers are consistent in their risk-taking behaviours, we believe that they will, as much as possible, tend to use the same strategy when performing program development and comprehension.

In Section 6, we describe an experiment that we performed to explore the relationship between spatial cognition and program comprehension. This experiment, by linking known but previously unrelated cognitive strategies, furthers our understanding of software maintenance and the sex differences that have been previously reported [33]. In Section 7, the results of this experiment are examined

and discussed, before we provide suggestions for future research in Section 8 and summarise our results in Section 9.

## 2 Program Comprehension

Program comprehension begins with source code and ends with a mental model describing the code’s purpose, operation, and abstractions. To construct this model, programmers must examine the text of the source code using some form of strategy.

Brooks [3] suggests that maintainers use a top-down hypothesis-driven model of comprehension. Beginning with a primary hypothesis, programmers examine source code to confirm and refine this hypothesis, and consequently generate new supplementary hypotheses. Soloway and Erlich [32] suggest that a top-down approach is used when maintainers are familiar with the source code. Using *rules of programming discourse*, programmers match code fragments to a suspected *program plan* that generalizes similar and familiar programs. In both models, programmers begin at a high level of abstraction and move towards a lower level.

Pennington [26] finds that maintainers use a bottom-up model of comprehension and begin by forming a control-flow *program model* based on the syntactic elements of the source code. A *situation model* is then constructed to describe the program in terms of semantic objects and concepts. In bottom-up comprehension, programmers begin at a low level of abstraction and move towards a higher level.

To unify the bottom-up and top-down comprehension approaches, von Mayrhauser and Vans [36] developed an integrated comprehension model. In their model, maintainers opportunistically shift between top-down and bottom-up approaches. Generally, programmers begin using a top-down approach when they have sufficient experience to generate high level hypotheses or they begin using a bottom-up approach when they lack this experience. A shift between these approaches occurs when either they lack the experience to form additional hypotheses (i.e., top-down to bottom-up) or they encounter a recognizable element such as a *beacon* [3] (i.e., bottom-up to top-down). Switches continue to occur as opportunities present themselves.

These models suggest that programmers use multiple strategies during program comprehension, depending upon a variety of contextual factors. Some of these factors, such as programmer experience [26], have been identified, while others remain unknown. However, independent of strategy, these models are based on the examination of source code. As documented by Mosemann and Wiedenbeck [25], programmers do not examine a file sequentially. Thus, during comprehension, programmers must develop a mental model of the code so that they can identify the location of the features under examination.

### 3 Spatial Cognition

Just as individuals vary in their use of a program comprehension strategy, they similarly vary in their performance of navigational tasks. As will be demonstrated in this section, the literature on spatial cognition indicates that individual, task and situational differences all affect the choice of skills used for spatial cognition.

Downs and Stea [12] define spatial cognition as:

“A process composed of a series of psychological transformations by which an individual acquires, stores, recalls, and decodes information about the relative locations and attributes of the phenomena in their everyday spatial environment.”

Spatial cognition is composed of several elements: wayfinding, object location memory and mental rotation. Each of these elements is now examined.

Wayfinding or navigation uses landmarks, route maps and survey maps. Landmarks are identifiable environmental markers associated with specific geographic locations [4]. Route maps are sequences of instructions, often involving landmarks, that describe at a personal level how to get from one location to another [2]. Survey maps are similar to topological maps and describe an environment's spatial layout independent of a specific navigational task [2].

It is well accepted in the spatial cognition literature that Piaget's model for children is correct and that we develop and learn to use these concepts in the order they are listed [27]. However, as adults, when placed in a new and unfamiliar environment, we do not learn about the environment in the same order.

Moeser found that after two years in the same building, most nurses had not developed survey maps of a hospital and used other strategies to navigate [23]. Appleyard [2] suggests that about 75% of the population use route maps and the remainder survey maps, but that this choice is mediated by a variety of factors. For example, one's situation is a factor; drivers are more likely to use a survey map while bus passengers are more likely to use a route map. It is also suggested that routes and landmarks are inseparable. A landmark is identified when needed to distinguish an important element of a route, such as a change of direction.

Aginsky *et al.* [1] show that the use of a route oriented or survey oriented strategy is an individual preference and that one can use the survey oriented strategy before developing a prerequisite route map. However, one is not restricted to using a sole strategy, as Kitchin [19] found that one person uses many strategies.

Kato and Takeuchi [17] discovered that, independent of wayfinding ability, individuals are able to use both survey maps and route maps and switch between the two strategies. However, individuals with a better sense of direction

made more effective use of each strategy and switched more appropriately. They also suggest that wayfinding is an opportunistic process and that individuals change strategies at different parts of a route. Also, one's choice of strategy may be dictated by the environment (e.g., when a lack of sufficient landmarks exists, one adopts a survey strategy).

It would be useless to navigate without any knowledge of places where one can go. Thus, our memory of objects (e.g., buildings) and their location with respect to other objects, is also an element of spatial cognition. Our ability to recall the set of objects that we know exist is referred to as *object memory* and our ability to recall the position of these objects is referred to as *location memory*. As with all elements of spatial cognition, our object and location memory can be used at various levels of abstraction. As well as remembering where we work and live, we must also remember the items that are in our home and on our desk.

Object and memory location are often measured using the tests developed by Silverman and Eals [30]. This test has been modified by James and Kimura [15] to explore our ability to remember an item's location when the item is moved (i.e., a location shift), or swapped with another item (i.e., a location exchange).

The final element of spatial cognition is that of mental rotation. Mental rotation is our ability to manipulate a three-dimensional object such that we can visualise the object from a different perspective. Mental rotation is often measured using a variant of the Vandenburg and Kuse mental rotation test [34]. This paper-based test is highly abstract and requires participants to match a two-dimensional drawing of a three-dimensional object with two of four possible solutions. The solutions are drawings that show either a rotated view of the object, or a rotated view of a similar, but slightly different object (e.g., a mirror image of the object).

Thus, spatial cognition is a complex task for which individual differences exist. When given a choice of cognitive skills such as using routes or survey maps, there is considerable variation depending upon the individual, the task and the current situation. As with program comprehension, an individual will often use more than one strategy or skill and will change opportunistically according to their current needs and situation.

### 4 Sex Differences in Behaviour

It has been well established among social psychologists that men are more risk tolerant, and take more risks, than women (see Daly and Wilson [8] for a comprehensive review). In contrast, women take fewer risks and accept risk only as a last resort, or when the benefits are maximised and the costs minimised. In general, women tend to choose high probability, low payoff strategies, whereas in similar

situations, men tend to choose low probably but high pay-off strategies [8].

When developing code, it is well known that the more code one writes, the greater the risk of an error being made. Thus we expect women, who prefer lower-risk strategies, will tend to write shorter code fragments and compile more frequently than will men. As well, since compilers only identify syntax errors, they should execute and unit test the code more frequently to detect other coding errors. If this is the case, women should tend towards using a bottom-up development strategy to permit more frequent testing of source code.

Bottom-up program development and comprehension are low risk strategies. One works from a known position and builds using only what one knows is correct. Top-down strategies are much higher risk, as they can not be confirmed as correct until the supporting lower-level code is available. Thus, we suggest that women, documented as more risk averse, will prefer to use bottom-up comprehension and development strategies while men are more risk prone and will use top-down strategies.

It is documented that women tend to program using a bottom-up approach while men tend towards using a top-down approach [33]. Thus, it is likely that this known difference is a result of differences in risk-taking behaviour. We expect a similar difference will exist for program comprehension task since they can also be performed using a top-down or bottom-up approach. As one's risk-taking preferences tend to remain consistent, the gender difference found for program development should also be evident in program comprehension. Of course, other factors such as programmer experience may limit one's choice of strategy. However, when a choice exists, we assume that, when feasible, women will tend to use a bottom-up comprehension strategy while men will prefer to use a top-down strategy.

It has also been found that women, more than men, tend to use less abstract, more concrete representations of the environment, such as landmarks. Conversely, men tend to use more abstract representations such as cardinal directions (e.g., north, south) [29]. When the level of abstraction is considered, it can be seen that bottom-up programming and landmark identification both use a low level of abstraction. Similarly, top-down programming and cardinal direction tend towards a higher level of abstraction.

Postma *et al.* [28] suggest that route-based navigation can be performed at both an abstract or a concrete level, but that the use of survey maps is predominately performed at an abstract level. If women tend towards the less abstract, then they should tend to use a route-based navigation strategy. Lawton [20, 21] confirms this suggestion and found that women were more likely than men to use route-based strategies whereas men were more likely than women to use survey navigation strategies.

Silverman and Eals [30] report that women tend to perform better on an object location task than do men. In related research, James and Kimura [15] reproduce Silverman and Eals' result when using object exchanges, but find no sex difference when using object shifts, suggesting that neither sex has an advantage when objects are moved to new positions. However, when asked to play the commercial game "Memory," McBurney *et al.* [22] found that women out-performed men by a large margin.

Spatial cognition demonstrates robust, frequently reproduced sex differences. Men typically perform better on abstract mental rotation tests, tend towards using survey maps when navigating, and do so because they are more accepting of the higher risk associated with these activities. Women typically perform better on more concrete object and location memory tests and tend towards using route-based navigation strategies, as these activities exhibit lower risk.

We believe it is important to note that the *tendencies* exhibited by a particular sex are simply activities that are more probabilistically likely, for that sex, and can not be construed as rules constraining one's activities. We also stress that it is incorrect, and inappropriate, to consider a specific sex-based behaviour as better or superior. Each sex has its tendencies and each sex is evolutionarily optimised to exploit these tendencies.

## 5 Comparing Spatial Cognition and Program Comprehension

From an evolutionary perspective, humankind has only recently needed skills for performing program comprehension. Hence, these skills should have developed from the adaptation of an existing skill used for some other task. Furthermore, as source code provides the notational representation for a complex set of domain and programming concepts, the skills needed to manipulate and understand code must be similarly complex and have arisen to manipulate and understand an equally complex environment.

When directly compared, program comprehension and spatial cognition do not exhibit much similarity. While there are gross, surface level correspondences, the two areas have significant differences when examined in detail. The key to achieving a successful comparison is found by using known sex-based differences in behaviour.

It is our hypothesis that since women tend to favour a lower risk and less abstract, bottom-up programming style and the less abstract and less-risky route-based navigation strategy, the two use similar cognitive skills. That is, route-based navigation is cognitively similar to bottom-up program comprehension. Men tend to prefer the higher risk and more abstract top-down programming style and the more abstract survey map navigation strategy, thus suggesting that the two approaches also use similar cognitive skills.

More specifically, survey map navigation is cognitively similar to top-down program comprehension.

Cartesian space is generally reduced to one-dimensional space within a specific source file. Programmers tend to assign a linear ordering to code, just as we assign a linear ordering to the words, sentences and paragraphs that make up this paper. The linear nature of code is made explicit in older language dialects, such as BASIC and Fortran, where all lines are monotonically numbered. This ordering simplifies the concept of direction, since one can only go forwards or backwards with respect to the numbered lines in a file. The treatment of codespace as unidimensional is further influenced by the fact that to improve readability, each line typically contains a single meaningful syntactic element.

To explore codespace, it is necessary to make some assumptions. First, we assume that for an object-oriented language such as Java, methods will be one of the primary codespace entities that programmers recognize and remember. Secondly, in a predominantly unidimensional environment, distances and sizes will be remembered with respect to this dimension. That is, a method's size is measured by the number of lines it covers and distance between two methods is measured by the number of lines between them. Finally, as one's maintenance ability is highly impacted by one's understanding of the code being maintained, we use measures of one's effectiveness at program maintenance as an indicator of one's program comprehension.

The experiment we describe in the next section compares participants' object and location memories, their mental rotation abilities, and their program maintenance abilities. Mental rotation and object location memory differ with respect to their level of abstraction and risk and as a consequence, exhibit robust sex differences. As top-down comprehension is risky, needs abstract skills, and is likely to be preferred by men, there should be correlation for men between mental rotation and their maintenance ability. Conversely, as bottom-up comprehension is low risk, concrete in nature and is likely to be preferred by women, there should be correlation for women between their object and location memory and their maintenance ability. We next describe our experiment before discussing the results.

## 6 Experimental Validation

To examine the use of spatial cognition in codespace, we performed a study to compare participants' knowledge about codespace objects with their spatial cognition skills. Participants were asked to perform a set of maintenance tasks and then given a post-task questionnaire. After completing the questionnaire, their mental rotation skills, object memory and location memory were tested.

### 6.1 Subjects

The 30 participants were students, both undergraduate (7) and graduate (23), at Dalhousie University in the faculties of Engineering (3), Science (2), and Computer Science (25). All participants indicated that they had sufficient knowledge of the Java programming language to perform a set of maintenance tasks on a Java program. One half the participants were male (mean age 27.67, SD = 5.59) and one half were female (mean age 25.80, SD = 5.28).

### 6.2 Stimuli

The program used in the study, `Calc.java` was 300 lines long and contained 10 methods in 1 class. The program extends the `Applet` class and implements a simple calculator providing the operations: addition, subtraction, multiplication, division, exponentiation, and square-root. Java was selected due to the results of an informal survey that indicated it was the language with which students were most familiar. Students were shown the program using the web-browser of their choice to avoid any effects introduced by differences in environment (i.e., editor). We consider the differences in features of web-browsers to be minimal when viewing a single file with no embedded hyperlinks.

While a 300 line program is certainly trivial with respect to actual production code, we were limited by experimental criteria from using a larger code-base. During pilot testing [7] we found that participants were unable to complete the experiment in under an hour when using a longer program. As it is known that participants are subject to fatigue effects in experiments lasting longer than about an hour, we chose to limit the size of the program we used to ensure that all data is of a high quality.

### 6.3 Procedure

Participants performed the study at their own work area, or at that of a researcher if they were undergraduates without an assigned work space. After ensuring that the participants could access the online stimuli, the session began by obtaining informed consent. Next, a brief demographic questionnaire was administered to identify the participants' age, sex, program, field of study, years of experience in Java and 'comfort' in Java programming (a score from 1 to 7).

To perform the maintenance tasks, participants were given 15 minutes, with a prompt when 10 minutes had elapsed. For the three tasks, each programmer had to identify by line number, the locations where changes were needed to modify the program's behaviour. Task 1 required the programmer to switch the positions of two buttons. In Task 2 the programmer had to change the type of a variable from the primitive type `double` to the class `Double`. For

Task 3 the applet background and button foreground colours were to be changed to blue and magenta, respectively.

During the tasks, the source code was viewable using a web-browser. For convenience, line numbers were displayed to the left of the source code. The maintenance tasks required changes to be made throughout the program with the first change on line 11 and the last on line 298. These tasks ensured that the programmers examined the entire source file. If the tasks were finished in less than 10 minutes, the programmer was encouraged to study the file until 10 minutes had elapsed in preparation for completing a post-maintenance survey. Once the tasks were completed, the browser window was closed to prevent further study of the file.

The maintenance tasks were scored to determine the participant's correctness and completeness with regard to the lines identified as needing changes. To simplify coding, the tasks were designed such that each could only be successfully accomplished in one way, thus allowing all results to be compared against this single correct solution.

When the maintenance tasks had been completed, each participant was handed a shuffled deck of 10 cards, with each card displaying the prototype for one of the program's methods. Participants were asked to sort the cards such that they matched the order in which the methods appeared in the program and with the top card identifying the first method. After sorting these cards, they were given a second, identical, deck of cards and asked to sort the cards with respect to the length of the methods and with the top card identifying the shortest method.

Next, participants were given a post-maintenance survey to complete. In the first part of the survey, participants had to indicate on a horizontal bar the position of a method in the program. The bar was divided into quarters, with the divisions being identified as lines 0, 75, 150, 225, and 300. An example was provided as part of the instructions. The location of 5 methods was asked using this technique. In the second part of the survey, 5 pairs of methods were given and, for each pair, participants were to identify the method that occurred first in the file. In the third part, 5 methods were listed and participants were asked to select the category (1-5 lines, 6-10 lines, 11-20 lines, over 50 lines) that best described the method's length. For the fourth and final part of the survey, participants were given 5 pairs of methods and asked to select the category (adjacent, 5-10 lines, 11-50 lines, 51-100 lines, over 100 lines) that best described the spacing between the two methods. No time restrictions were put on completing the survey.

Participants then completed the object memory test and object location test using the protocol described by Silverman and Eals [30]. Each test was timed and took two minutes to complete. To conclude the study, the Vandenburg and Kuse mental rotation test [34] was then administered.

Participants were given 10 minutes to complete the test and scored using *negative* scoring – one point for a correct solution, minus one for a wrong solution and zero for an unattempted item.

After completing this set of surveys, each participant was debriefed, thanked for their time and given \$10 in remuneration. On average, each participant took about 45 minutes to complete the session.

## 7 Discussion

Within this section, an  $\alpha$  value of .05 is used to determine significance. All  $t$ -tests are two-tailed and assume that samples are independent. Rather than following a more traditional style, we combine our presentation of results with our discussion due to the complexity of the experiment.

### 7.1 Sampling Differences

Males and females were compared to determine whether they differed in years of experience using Java, comfort in Java programming, years of university experience and age. No significant differences were found between these groups ( $p > .05$  in all cases). When compared on their software maintenance skills, no significant differences were found between men and women in terms of their mean correctness or completeness on these tasks.

The lack of a significant difference between men and women, with respect to their background, suggests that the two groups are equivalent and that any differences that are found are not a result of their experience. The lack of difference on their ability to perform software maintenance further confirms this equivalence.

However, on almost all measures, women exhibited much higher standard deviations than did men. We have identified three possible explanations for this finding. First, there are very few women in the Dalhousie computer science programs. Thus, they may represent a far broader population than does the small subset of men we surveyed. Second, it is well known that it is difficult for women to flourish in the male-dominated world of computer science. Thus, it is possible that those who succeed in this domain are not 'typical' and may not form a highly representative sample. Finally, while not as probable, we may have inadvertently surveyed a very homogeneous and non-representative sample of men.

### 7.2 Location and Object Memory

Surprisingly, there were no sex differences on object memory,  $t(28) = .65$ ,  $p > .05$ , or location memory,

$t(28) = .082, p > .05$ . There were some significant findings, however, in how these abilities relate to program comprehension.

For all participants, location memory significantly correlates with post-maintenance task one,  $r(30) = .383, p = .037$ , with post-maintenance task three,  $r(30) = .50, p = .005$ , and with the card sort by method position,  $r(30) = .473, p = .008$ . However, when examined by sex, it can be seen that performance on the post maintenance tasks did not correlate significantly with location memory for men. Thus, the correlations are a consequence of women's abilities.

Location memory significantly correlated, for women, with their total correctness for task two,  $r(15) = .61, p = .017$ , which asked participants to state which of two methods appeared first in the file. As well, women's location memory significantly correlated with task three,  $r(15) = .58, p = .023$ , which asked participants to select the corresponding length of a method from a list. Thus, women with a high location memory ability recalled more correct orderings of methods, and knew the length of these methods. Additionally, although not significant, task one (the identification of the line number where a method began) approached significance when correlated with location memory for women,  $r(15) = .46, p = .076$ . Interestingly, the last post maintenance task, which asked participants the spacing between methods, was not correlated with location memory for women.

Location memory was not correlated with either of the card sort tasks for men, yet did yield significant correlations for women. Card ordering by location was significantly correlated with location memory  $r(15) = .69, p = .005$ , such that women's location memory was associated with more correct ordering of methods. Similarly, women's location memory was also significantly correlated with card ordering by method length,  $r(15) = .63, p = .011$ . Moreover, for women but not men, the two card sorts were related,  $r(15) = .57, p = .025$ .

Location memory seems to be important for women's global program comprehension. It is how they remember the location of items, and their length. The knowledge they can recall is important because it suggests they are forming landmarks using codespace entities, but have not yet begun the next stage of relating these landmarks, using their spacing, to form routes.

Object memory did not significantly correlate with men's or women's post maintenance tasks or with location memory. However, it was significantly associated with women's, but not men's, average correctness on the three code-based maintenance tasks,  $r(15) = .52, p = .048$ .

The findings for both object and location memory suggest that, when performing program comprehension during maintenance, women are using a bottom-up strategy,

working with the code on a low level, and are thus learning detailed low level spatial information about the methods – their names, locations and lengths. We suggest that women are using a bottom-up approach due to its low level concrete nature and its relationship to a low level route-based navigation strategy in which landmarks (significant objects) are identified as a precursor to forming routes. The fact that men did not demonstrate the same correlations provides evidence of a sex difference and supports our model.

### 7.3 Mental Rotation

We did not find a sex difference on the mental rotation test,  $t(28) = 1.41, p > .05$ . Although we were initially surprised by this finding, there are several potential explanations. We review these explanations in Section 7.5.

Women's mental rotation did not significantly correlate with their performance on the post-maintenance tasks, card sorts, or object and location memory tests. However, men's mental rotation did correlate with their average correctness on the software maintenance task,  $r(15) = .63, p = .012$ .

The mental rotation test is highly abstract, as is top-down program comprehension. If men tend to use abstract skills when performing comprehension, their abilities on the two should correlate. Thus, this correlation provides evidence for men's use of a top-down, more abstract, program comprehension strategy.

We do not deny that men could be using some form of mental rotation to orient themselves when performing software navigation, program comprehension and software maintenance. However, the linear nature of source code does not suggest that mental rotation is as important, or will be used as much, as women's use of location memory.

### 7.4 Approaches to Software Maintenance

The performance of men of post-maintenance task three significantly correlates with their average maintenance score for correctness,  $r(15) = .69, p = .005$ . For women, their performance on the object memory test and post-maintenance task one significantly correlates with their average maintenance score for correctness,  $r(15) = .52, p = .048$ , and  $r(15) = .56, p = .031$ , respectively. There were no significant correlations for men's or women's average maintenance completeness scores.

These correlations with the average maintenance correctness score are revealing and important. For men, their maintenance score (ability) is correlated with maintenance task three (how long is a method?). Thus, men are aware of a method's size, but not its location. For women, their maintenance score is correlated with their object memory and their score on task one (where is this method located?). Thus, women are aware of a method's location.

When using a top-down program comprehension strategy, men will focus on the functionality of the program's methods. If one assumes that methods that do more computation will take up more space (i.e., more lines), the relationship between men's maintenance skills and their awareness of method length can be considered as evidence that they are using a top-down comprehension approach.

Women are more aware of a method's location. Thus, the correlation of their method location and maintenance skills is evidence of the use of a bottom-up comprehension strategy. Rather than focusing on what a method does, women are tracing control-flow and, in doing so, are more aware of where methods are located and their order within a file.

## 7.5 Sex Differences

Although we expected to find sex differences in mental rotation, object memory, and location memory, the lack of any differences is easily explained. Typically, these psychological measures are administered to participants who do not rely on these abilities as part of their professional development. To succeed as a computer programmer, men are forced to develop better than average object memory skills. It is impossible to work with large numbers of source code entities without being able to organize and recall them. Conversely, computer programming can be highly abstract, thus forcing women to become more adept at abstract skills such as mental rotation.

However, regardless of their equivalence in these skills and when given the choice, men and women will still use the skills they are more comfortable with and first developed. Women can operate abstractly and equivalently to men, but will prefer to form route maps using their object location skills and a bottom-up approach. Men can remember the locations of objects, but will prefer to operate abstractly and form survey maps using a top-down approach. These preferences are demonstrated in the sex differences that our correlations exhibit.

Women can be considered as focusing on low level details, such as method names and positions, as they are using a low level, concrete strategy to identify elements that can be used as landmarks for a route-based navigation strategy. The better they can find landmarks, the better their navigation will be, thus allowing them to learn more about the software and improving their ability to perform maintenance. All these skills are related and built on the concept of minimising risk by using a low level, bottom-up approach. Men focus on high level abstractions and build a functional model of source code. Their ability to perform maintenance is based on their ability to use an abstract, high risk, top-down approach. That is, women learn there is a `calculate` method while men learn there is a method that calculates.

## 8 Future Work

It might be considered that one of the most important elements of our work is its ability to suggest new directions for research. While we have potentially established relationships between program comprehension, software navigation, spatial cognition, risk-taking behaviour and preference for abstraction, these relationships expose a wealth of research opportunities. We now present a few of these opportunities to provide evidence of the model's ability to integrate different research areas.

### 8.1 Landmarks

When examining source code, *beacons* are used to identify features and structures (i.e., meaningful application domain objects) within the code [3]. Brooks suggests that programmers scan or search source code for beacons that confirm or suggest the existence of specific syntactic and semantic objects. This use of beacons suggests that they may have a relationship to landmarks. However, we know of no research that explores the use of beacons for navigation.

We suggest that beacons are distinct to source code landmarks. Beacons identify the existence of a specific feature. Just as a square-shaped mound indicates the existence of a probable archaeological site, three sequential assignments using three variables indicates a probable swap and consequently, a sorting algorithm. Beacons indicate the presence of another, usually (cognitively) larger, object while landmarks identify a significant positional location in space.

Although they are distinct from landmarks, beacons are not unrelated. We suggest that beacons are a component of a landmark. With a church, the spire can serve as a beacon to indicate that a church exists and the church can serve as a navigational landmark. In code, the name 'sort' in a subroutine definition acts as a beacon for the existence of a sorting subroutine that can be used as a source code landmark. As they are a component of landmarks, the known sex differences in the ability to recall landmarks should also be exhibited for beacons. It now falls on future research to explore the relationships between beacons and landmarks and their opposing roles in software navigation and program comprehension.

### 8.2 Visual Sub-Systems

Viewing the world is much like program comprehension. That is, we gather data using our visual system and then generate meaning for this data. During program comprehension, we read source code to gather data, and then map the data to application domain concepts to give it meaning.

Within the human visual system, it has been shown that there are two separate subsystems: the *contour* and the *lo-*

tion systems [16]. The contour system identifies objects while the location system determines an object's spatial location. These perspectives match the concepts of object memory and location memory. Analogously, we can expect to find parallel systems in place for navigating source code.

Research on program comprehension suggests that programmers generate meaning for a program by assigning code fragments to domain concepts. This task can be considered as using the contour subsystem. That is, programmers identify meaningful conceptual objects that can be associated with the code.

Current research on program comprehension pays little attention to the equivalent of the location system. There is little examination on the location of these objects, or of their manifestations in source code. Program comprehension models specify the cognitive structures used to generate meaning for code and ignore the spatial relationships between code fragments and domain concepts. It is likely that this omission is tied to the lack of attention that is given to the structural view of source code. By ignoring positional relationships, the current models of program comprehension can be viewed as incomplete. Addressing this incompleteness can lead to an improved knowledge of the comprehension process and in turn can motivate the production of better techniques, environments, and tools for software maintenance.

### 8.3 Risk-Taking During Comprehension

We have found initial evidence to support the idea that one's risk-taking behaviour is a key influence on one's choice of program comprehension and development strategy. Using a top-down strategy is highly risky, as it takes considerable time for results to be achieved and one is prone to making more mistakes over a longer period of time. In opposition, a bottom-up strategy is less risky as results, albeit much lower-level ones, are more readily obtainable.

We suggest that exploring the relationship between risk and code manipulation strategy will yield considerable insight into both program comprehension and development. Furthermore, this exploration has the potential to considerably improve, refine and add detail to our model. The known sex differences in risk-taking permit an individual's perception of risk to provide the focus for an experiment and thus serve as a surrogate for one's sex. In this way, experiments can be carried out on the predominantly male population of computer science and then extrapolated, via risk, to identify their effect on female populations. We consider this relationship to be an important one that could potentially address many significant sociological issues exhibited within computer science and related informatics disciplines.

## 9 Conclusion

In this article, we have created a model to link spatial cognition, source code navigation and program comprehension. The model uses well known and robust sex differences in behaviour to identify links between the various elements. The complexity of our model suggests that source code navigation is not a trivial activity that occurs while we develop a code-to-concept map as part of program comprehension. Instead, the model indicates that programmers employ their preferred spatial cognition skills and create a spatially-based view of a software system. We refer to this spatial perspective as 'codespace.'

We readily admit that our model is complicated and the experiment that we performed is equally so. However, human behaviour is the product of many factors, and models that consider these many factors are necessarily complex. We consider our contribution as a first step in exploring the relationships between program comprehension, program development and software navigation.

Furthermore, it is clear that our initial experiment is not conclusive. We have found strong supporting evidence for our hypothesis, but additional experimentation is needed to replicate and validate our results. It now falls to our future research, as well as that of other researchers, to verify, correct, refine and extend our initial model.

The rewards for understanding programmers' behaviour at the cognitive level are high. Tool developers will be able to leverage this knowledge to create more effective tools that are more likely to be adopted. Educators will be more able to teach program development and comprehension through an improved understanding of the underlying cognitive skills. Finally, and perhaps most importantly, by understanding how sex-based differences in cognition are applied, there is the potential to tailor computer science programs to better suit the needs of different populations and potentially address the under-representation of these populations, such as women, in the computing disciplines.

Program comprehension is an important element of software maintenance. We believe that our work identifies new and previously unexplored links between comprehension, abstraction, risk-taking, and spatial cognition. These links expose a vast realm of new research opportunities and serve to further our knowledge of program comprehension and the rapidly developing area of software navigation.

## References

- [1] V. Aginsky, C. Harris, R. Rensink, and J. Beusmans. Two strategies for learning a route in a driving simulator. *Journal of Environmental Psychology*, 17:317–331, 1997.
- [2] D. Appleyard. Why buildings are known. *Environment and Behavior*, 1:131–156, 1969.

- [3] R. Brooks. Towards a theory of the comprehension of computer programs. *International Journal of Man-Machine Studies*, 18(6):543–554, 1983.
- [4] E. Chown, S. Kaplan, and D. Kortenkamp. Prototypes, location, and associative networks (PLAN): Towards a unified theory of cognitive mapping. *Cognitive Science*, 19:1–51, 1995.
- [5] A. Clark. *Microcognition: Philosophy, cognitive science and parallel distributed processing*. MIT Press, Cambridge, MA, 1989.
- [6] L. Cosmides and J. Tooby. Cognitive adaptations for social exchange. In *The Adapted Mind*, chapter 3, pages 163–228. Oxford University Press, Oxford, UK, 1992.
- [7] A. Cox, M. Fisher, and P. O’Brien. Theoretical considerations on navigating codespace with spatial cognition. In *Annual Workshop of the Psychology of Programming Interest Group*, 2005.
- [8] M. Daly and M. Wilson. Risk-taking, intrasexual competition, and homicide. In *Nebraska Symposium Series, No. 47: Motivation*, pages 1–36, 2001.
- [9] R. DeLine. Staying oriented with software terrain maps. In *International Conference on Distributed Multimedia Systems*, Banff, Canada, September 2005.
- [10] M. Desmond, M.-A. Storey, and C. Exton. Fluid source code views for just in-time comprehension. In *Workshop on Software Engineering Properties of Languages and Aspect Technologies*, Bonn, Germany, March 2006.
- [11] C. Douce, P. Layzell, and J. Buckley. Spatial measures of software complexity. In *Annual Workshop of the Psychology of Programming Interest Group*, pages 36–45, Leeds, UK, January 1999.
- [12] R. Downs and D. Stea. *Image and Environment: Cognitive Mapping and Spatial Behaviour*. Aldine, Chicago, IL, 1973.
- [13] T. Green. Cognitive approaches to software comprehension: Results, gaps and limitations. In *Experimental Psychology in Software Comprehension Studies*, Limerick, Ireland, 1997. Extended abstract of talk.
- [14] T. Green and R. Navarro. Programming plans, imagery, and visual programming. In *Human-Computer Interaction: INTERACT-95*, pages 139–144, London, UK, 1995. Chapman and Hall.
- [15] T. James and D. Kimura. Sex differences in remembering the locations of objects in an array: Location-shifts versus location exchanges. *Evolution and Human Behavior*, 18:155–163, 1997.
- [16] S. Kaplan and R. Kaplan. *Cognition and Environment*. Praeger, New York, NY, 1982. Republished, 1989, Ulrich’s, Ann Arbor, MI.
- [17] Y. Kato and Y. Takeuchi. Individual differences in wayfinding strategies. *Journal of Environmental Psychology*, 23:171–188, 2003.
- [18] M. Kersten and G. Murphy. Mylar: A degree-of-interest model for IDEs. In *International Conference on Aspect-Oriented Software Design*, pages 159–158, Chicago, IL, March 2005.
- [19] R. Kitchin. Exploring spatial thought. *Environment and Behavior*, 29:123–156, 1997.
- [20] C. Lawton. Gender differences in way-finding strategies: Relationship to spatial ability and spatial anxiety. *Sex Roles*, 30(11/12):765–779, 1994.
- [21] C. Lawton. Strategies for indoor way-finding: The role of orientation. *Journal of Environmental Psychology*, 16:137–145, 1996.
- [22] D. McBurney, S. Gaulin, T. Devineni, and C. Adams. Superior spatial memory of women: Stronger evidence for the gathering hypothesis. *Evolution and Human Behavior*, 18:165–174, 1997.
- [23] S. Moeser. Cognitive mapping in a complex building. *Environment and Behavior*, 20:21–49, 1988.
- [24] A. Mohan, N. Gold, and P. Layzell. An initial approach to assessing program comprehensibility using spatial complexity, number of concepts and typographical style. In *Working Conference on Reverse Engineering*, pages 246–255, Delft, Netherlands, November 2004.
- [25] R. Mosemann and S. Wiedenbeck. Navigation and comprehension of programs by novice programmers. In *IEEE International Workshop on Program Comprehension*, pages 79–88, Toronto, Canada, April 2001.
- [26] N. Pennington. Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive Psychology*, 19:295–341, 1987.
- [27] J. Piaget and B. Inhelder. *The Child’s Conception of Space*. Norton, New York, NY, 1967.
- [28] A. Postma, G. Jager, R. Kessels, H. Koppeschaar, and J. van Honk. Sex differences for selective forms of spatial memory. *Brain and Cognition*, 54:24–34, 2003.
- [29] D. Saucier, M. Bowman, and L. Elias. Sex differences in the effect of articulatory or spatial dual-task interference during navigation. *Brain and Cognition*, 53:346–350, 2003.
- [30] I. Silverman and M. Eals. Sex differences in spatial abilities: Evolutionary theory and data. In *The Adapted Mind*, chapter 14, pages 533–549. Oxford University Press, Oxford, UK, 1992.
- [31] J. Singer, R. Elves, and M.-A. Storey. Navtracks: Supporting navigation in software maintenance. In *IEEE International Conference on Software Maintenance*, pages 325–334, Budapest, Hungary, September 2005.
- [32] E. Soloway and K. Erlich. Empirical studies of programming knowledge. *IEEE Transactions on Software Engineering*, SE-10(5):595–609, September 1984.
- [33] S. Turkle and S. Papert. Epistemological pluralism: Styles and voices within the computer culture. *Signs*, 16:128–157, 1990.
- [34] S. Vandenburg and A. Kuse. Mental rotations: A group test of three dimensional spatial visualization. *Perception and Motor Skills*, 47:599–604, 1978.
- [35] N. Vinson. Design guidelines for landmarks to support navigation in virtual environments. In *ACM CHI 99 Conference on Human Factors in Computing Systems*, pages 278–285, Pittsburgh, PA, May 1999.
- [36] A. von Mayrhauser and A. M. Vans. Comprehension processes during large scale software maintenance. In *International Conference on Software Engineering*, pages 39–48, Sorrento, Italy, May 1994.