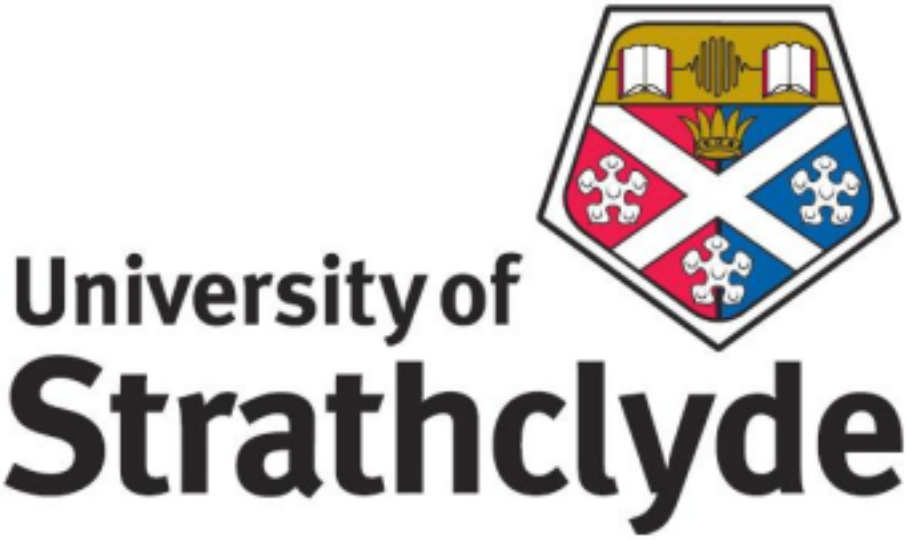
University of Strathclyde
Department of Computer & Information Sciences

Investigating and Improving Novice Programmers' Mental Models of Programming Concepts

by Linxiao Ma

A thesis presented in fulfilment of the requirements for the degree of Doctor of Philosophy

2007



ABSTRACT

Current programming education seems far from successful. A large proportion of programming students are failing programming courses all around the world. While lack of problem-solving ability is viewed as a possible cause of failure in programming learning, another potential cause is that students may hold non-viable mental models of key programming concepts which may lead to misconceptions and difficulties in solving programming problems. This thesis presents research that aims to investigate the viability of the mental models held by novice programmers, and to suggest and evaluate a constructivist-based teaching approach to improve these models.

This thesis first presents the theoretical knowledge and related work in the field of constructivism, mental models of novice programmers, visualization in programming education, and cognitive conflict strategy. A study that aimed to investigate the viability of novice programmers' mental models of fundamental programming concept (focusing on value assignment and reference assignment) is then presented. The results of this study revealed that many students were holding non-viable mental models of basic programming concepts such as variables, assignment and reference. In addition, this study found that the students with viable mental models performed significantly better in programming tasks than those with nonviable mental models. In order to improve novice programmers' mental models, a constructivist-based teaching model that integrates a cognitive conflict strategy and program visualization is proposed. A series of studies were conducted to evaluate the effectiveness of the proposed learning model. The results indicate that the proposed learning model is effective to enhance students' interests and engagement with the learning materials and help them to construct viable mental models. However, the visualization technique seems less effective for a relatively complex concept such as reference, when students lack necessary base knowledge to interpret the visualization.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude and appreciation to my supervisors, Dr. John D Ferguson, Dr. Marc Roper, and Dr. Murray Wood for their excellent supervision and invaluable encouragement of my research work. This dissertation would never have been completed without their remarkable enthusiasm, inspiration, and patience.

I would also like to thank many other people who contributed to this dissertation. First of all, sincere thanks to Dr. John Wilson for his enthusiastic support and advice for my Ph.D. study. Special thanks go to my officemate, Mr. David C Elsweiler, who was always there when I needed help. In particular, he gave me a lot of invaluable advice on how to be a Ph.D. student. I would also like to gratefully and sincerely thank Dr. Douglas Kirk, Mr. Konstantinos Liaskos, and Mr. Inah Omoronyia who helped me prepare experiment materials and gave me invaluable suggestions and feedback on my research.

I would like to give my deepest gratitude and love to my parents for their dedication and faith in me. I would also like to thank my girlfriend for the many years of love and support during my graduate study.

PUBLICATIONS

- Ma, L., Ferguson, J. D., Roper, M., Wilson, J., and Wood, M., "A Collaborative Approach to Learning Programming: a hybrid learning model", *6th Annual Higher Education Academy Subject Network for Information and Computer Science Conference*, York, UK, August 2005. pp. 75-80.

- Ma, L., Ferguson, J.D., Roper, M., and Wood, M., "Investigating the Viability of Mental Models Held by Novice Programmers". *38th ACM Technical Symposium on Computer Science Education*. Covington, Kentucky. 2007.

- Ma, L., Ferguson, J.D., Roper, M., and Wood, M., "Improving the Viability of Mental Models Held by Novice Programmers". *Eleven Workshop on Pedagogies and Tools for the Teaching and Learning of Object Oriented Concepts. ECOOP Workshops 2007*.

- Ma, L., Ferguson, J.D., Roper, M., Ross, I., and Wood, M., "Using Cognitive Conflict and Program Visualization to Improve Mental Models Held by Novice Programmers". *39th ACM Technical Symposium on Computer Science Education*. 2007.

CONTENTS

CHAPTER 1 – Introduction

Current programming education seems far from successful. A 2001 ITiCSE working group (the 'McCracken group') conducted a multi-national, multi-institutional study to assess the programming ability of first-year programming students and found that most students performed much more poorly than expected - the average score was only 22.89 out of 110 points on the general evaluation criteria (McCracken et al., 2001). This poor performance is undoubtedly a major contributor to the relatively high dropout rates, of around 30-50% (Denning & McGettrick, 2005), associated with Computer Science courses. Considering the rapidly increased influence of software technology that promotes the large demand for skilled programmers, significant concerns are being raised regarding programming education.

While lack of problem-solving ability is viewed as the main cause of failure in programming learning (e.g. Barnes et al., 1997), previous studies (e.g. Bayman & Mayer, 1983) found students often hold non-viable mental models of key programming concepts which may cause misconceptions and difficulties in solving programming problems. Researchers and instructors (e.g. Lui et al, 2004) highlight that the development of viable mental models of key concepts is critical in order to learn to program.

Object-oriented programming is currently the dominant programming paradigm used in industry. Many introductory programming courses are teaching programming starting with object-oriented techniques. However, several programming teachers and educators (e.g. Ben-Ari, 2001a) argue that it is impossible for students to properly understand and use object-oriented techniques without viable mental models of fundamental programming concepts such as variables and assignment.

To improve students' mental models it is proposed that an approach to teaching programming that emphasizes constructivism (Ben-Ari, 2001a) rather than

objectivism (Vrasidas, 2000) might be helpful. It is proposed that many students have deeply rooted, pre-existing ideas of how computing concepts such as assignment might operate. Constructivism theory argues that traditional approaches to teaching based on lectures and textbooks are too passive and do not do enough to challenge pre-existing ideas and to help students create viable mental models. Instead constructivism argues that students actively construct knowledge by combining the experiential world with existing cognitive structures (Ben-Ari, 2001a). One of the key teaching strategies in this approach is that of *cognitive conflict* which explicitly challenges existing ideas in order to encourage the learner to recognize errors in their understanding and to try to promote the improvement of non-viable mental models (Scott et al., 1992).

However, it should be noted that cognitive conflict alone is unlikely to be sufficient to achieve a change in non-viable models. Students must be supported to create new viable models, and concepts must be presented in an order and fashion that allows the correct construction of inter-dependent models. This is not an easy task, especially for programming students. Programming concepts are invisible and untouchable. Students cannot 'see' what is happening 'in' a computer when a program is executed. This increases the difficulty of constructing viable mental models of programming concepts. To address this, Ben-Ari has proposed that program visualization has the potential to create a suitable learning environment (Ben-Ari, 2001b). Visualization techniques have been used for over 20 years and have, arguably, not been as successful as hoped for. A possible reason for this is that they have been used from a traditional, objectivist perspective, ignoring a student's pre-existing models. It is therefore proposed that a potential way forward is to adopt an approach based on cognitive conflict to help students realize that there is a problem with their current understanding and to use a visualization-oriented learning environment to support them in correcting their non-viable models.

This research aims to investigate the viability of mental models held by novice programmers, and to suggest and evaluate a constructivist-based teaching model that integrates a cognitive conflict strategy with program visualization to improve novice

programmers' mental models. The following part of this chapter summarizes the research questions, research hypotheses and contributions of this Ph.D. project. The structure of this thesis is then introduced at the end of the chapter.

1.1 Research Questions

The following lists the research questions explored in this thesis:

- What is the range and viability of mental models held by novice programmers?
- Does the viability of the mental models held by novice programmers affect their performance in solving programming problems?
- Is a cognitive conflict strategy able to improve the effectiveness and pedagogical benefits of program visualization techniques?
- Is a constructivist-based learning model that integrates cognitive conflict and program visualization able to improve novice programmers' mental models of basic programming concepts?

1.2 Research Hypotheses

The following are the research hypotheses of this project:

- Novice programmers often hold non-viable mental models of basic programming concepts. Those who hold viable mental models will perform better in the programming tasks than those who hold non-viable mental models.
- A constructivist-based teaching model that integrates a cognitive conflict strategy and a program visualization technique would be more effective in establishing mental models of programming concepts than the traditional, objectivist-based teaching models.

1.3 Contributions

The work presented in this thesis makes the following contributions:

- Up until now, there were few empirical studies specifically conducted to investigate novice programmers' mental models of basic programming concepts. This thesis enriched the findings regarding the range and frequency of mental models held by novice programmers, focusing on the area of reference and value assignment.
- There was no previous work investigating the relations between the viability of novice programmers' mental models and their performance in solving programming problems. This research identified that students holding viable mental models performed better than those that held non-viable mental models.
- This research proposes and evaluates a constructivist-based learning approach that integrates a cognitive conflict strategy and program visualization technique to improve novice programmers' mental models.
- Visualization techniques have been used in a large number of domains. However, they are not as effective as expected. Many researchers (e.g. Naps et al., 2003) have been searching for the reason for this ineffectiveness. This project investigates whether or not visualization techniques would be more effective in a constructivist-based learning environment.
- An understanding of novice programmers' mental models is very important for instructors to design effective learning materials and activities. This thesis proposes a practical way to elicit novice programmers' mental models in an education context.
- This thesis proposes a questionnaire to investigate novice programmers' mental models of value and reference assignment concepts by collecting quantitative data and qualitative data.
- This project proposes a computer-supported learning environment to support the proposed learning model.

1.4 Outline of the Thesis

The theoretical knowledge related to constructivism, mental models and the cognitive conflict strategy are presented in **Chapter 2**. The general principles of

constructivism and its application in computer science education are first discussed, followed by an introduction of the definition, functions and characteristics of mental models, which is an important concept related to constructivism. Finally, a constructivist-based teaching strategy, cognitive conflict, is presented with the descriptions of the important theory and models underlying this teaching strategy, including *Piaget's Equilibrium Theory* (Piaget, 1977), Posner et al. (1982) and Hewson & Hewson (1984)'s *Conceptual Change Model*, and Lee & Kwon (2001)'s *Cognitive Conflict Process Model*.

Chapter 3 presents a review of previous studies related to the investigations of mental models held by novice programmers, the visualizations used in programming education and an assessment of the effectiveness of the cognitive conflict strategy. Firstly, a collection of previous studies that were specifically conducted to investigate novice programmers' mental model are analyzed, with special emphasis on the mental model elicitation methods used in these studies. This chapter then presents the state of the art of visualization when used in programming education, explaining the terminology currently associated with visualization systems, introducing a survey of visualization systems specifically developed for programming education purposes, and describing empirical studies of the effectiveness of visualization used in programming education. Finally, this chapter presents a review of empirical studies on the effectiveness of the cognitive conflict teaching strategy.

An investigation of the viability of the mental models held by novice programmers is described in **Chapter 4**. This study aimed to identify the range and frequency of the mental models held by novice programmers in the areas of value and reference assignment. In addition, the relationship between novice programmers' mental models and their performance in course tests and their final examination was also investigated. This chapter presents the research aims, methods, results and findings of this study.

The findings from the mental models investigation reveal that novice programmers often hold non-viable mental models, and those with non-viable mental models performed significantly worse than those with viable mental models. These findings highlight the importance of helping novice programmers construct viable mental models of key programming concept. **Chapter 5** proposes a constructivist-based learning model integrating cognitive conflict strategy and program visualization to improve novice programmers' mental models. In addition, chapter 5 also presents a computer-based learning environment developed by the author to support the proposed learning model.

Chapter 6 presents three studies that were conducted to evaluate the effectiveness of the proposed learning model for improving the mental models held by the students on an introductory programming course. The first study focused on a relatively simple programming concept, value assignment, while the second study investigated a relatively complex programming concept, reference assignment. The third study, conducted at end of the course, investigated students' mental models of both concepts again after a period of time. The results obtained from this study were compared with the results from the first mental model study (chapter 4) conducted in the previous year in order to compare and investigate whether or not the students who experienced the proposed teaching model in this year, performed better than those who did not experience this model in the previous year. In addition, this test is used to investigate the long-term effects of the proposed teaching model.

Finally, **Chapter 7** summarizes the achievements and limitations of this thesis, and suggests future work that could follow this research.

1.5 Timeline of the Empirical Studies

As Table 1-1 shows, a series of empirical studies were carried out to investigate novice programmers' mental models and to evaluate the effectiveness of the proposed learning model for improving novice programmers' mental models. At the

end of the academic year of 2005-2006, a mental model test was conducted to study first year programming students' mental models of assignment and reference concepts. In the fifth week of the academic year of 2006-2007, a study was carried out to investigate the effectiveness of the proposed learning model for the value assignment concept. In the twelfth week of this year, another study was carried out to investigate the effectiveness of the proposed learning model for the reference assignment concept. At end of this year, the first mental model test that took place in the academic year of 2005-2006 was repeated in order to form a comparison between the performance of the students who experienced the proposed learning model in the academic year 2006-2007 and those who did not experience the proposed learning model in the previous year.

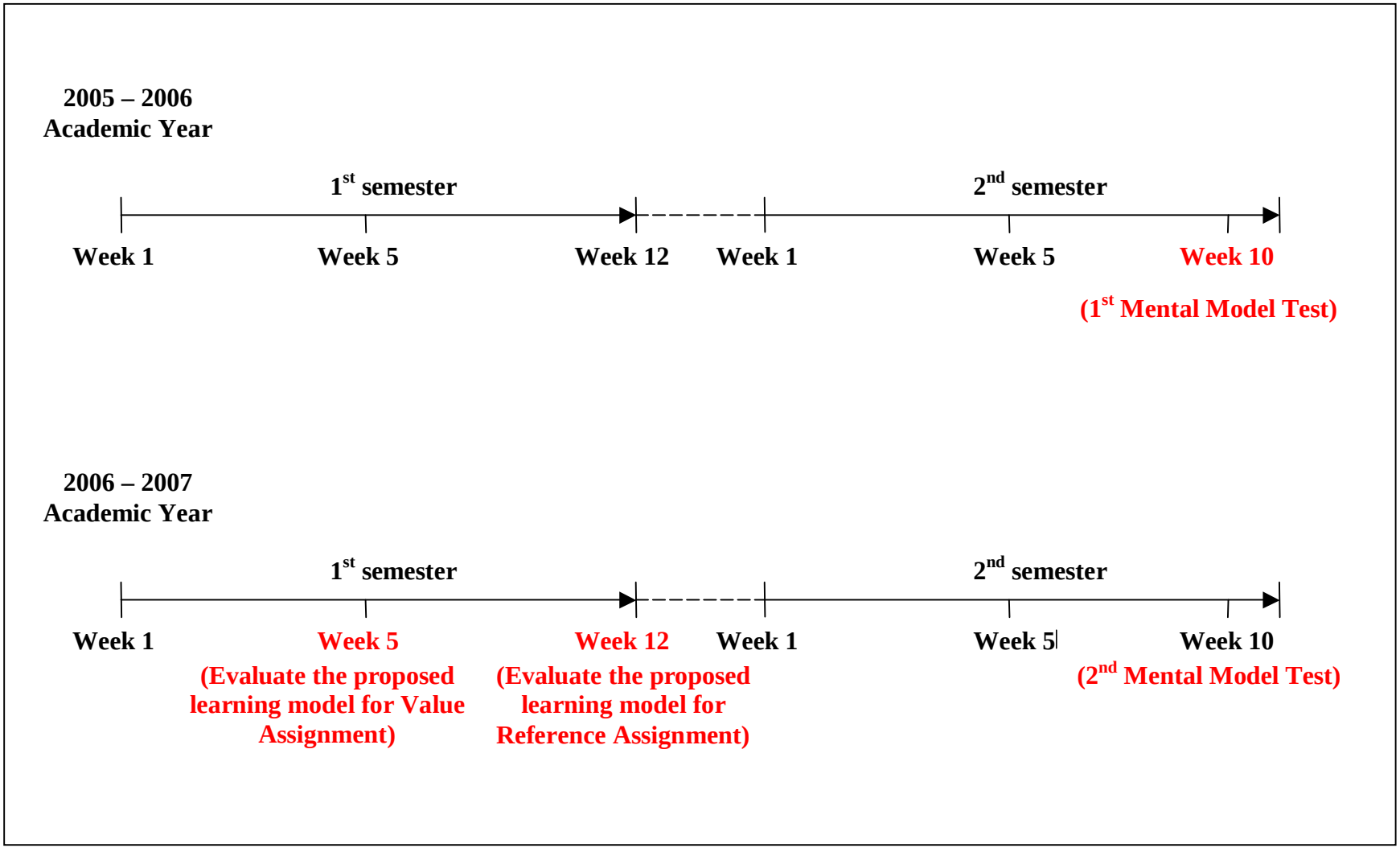


Table 1-1: The Timeline of the Empirical Studies

CHAPTER 2 - Theoretical Knowledge of Constructivism, Mental Models, and Cognitive Conflict

A number of educational theories are widely recognized as providing a model for educational practice. Objectivism describes a world that is made up of objects. Learning is seen as a process to create corresponding representations of these objects in the minds of learners (Lakoff, 1987). Cognitivism focuses on how human memory works to promote learning. It considers learning as "*involving the acquisition or reorganization of the cognitive structures through which humans process and store information*" (Good & Brophy, 1990). Constructivism emphasizes that students actively construct knowledge by combining the experiential world with existing cognitive structures (Ben-Ari, 2001a).

Over recent decades, constructivism has emerged as the key theory for relating prior experience to educational processes, particularly in the context of computer science education (Ben-Ari, 2001a). This chapter presents theoretical knowledge of constructivism along with an introduction of two important concepts, mental models and cognitive conflict, related to constructivism. Firstly, the general principles of constructivism and its application in computer science education are discussed in section 2.1, followed by a theoretical introduction of mental models in section 2.2, covering their definition, functions, and characteristics. Lastly, cognitive conflict and the main theory and models underlying this teaching strategy are discussed in section 2.3.

2.1 Constructivism

2.1.1 The Principles of Constructivism

Constructivism claims that knowledge does not exist independently of the learners. Students actively construct knowledge by combining the experiential world with

existing cognitive structures, rather than passively absorbing knowledge from lectures or textbooks (Ben-Ari, 2001a). Ben-Ari (2001a) points out that "*Teaching techniques derived from the theory of constructivism are supposed to be more successful than traditional techniques, because they explicitly address the inevitable process of knowledge construction*". Currently, constructivism has become the dominant theory of learning (Ben-Ari, 2001). The term 'constructivism' has become fashionable. Many 'constructivist' types of theories are currently being discussed in education domains (Proulx, 2006). As Ernest (1995) claimed, "*there are as many varieties of constructivism as there are researchers*". Therefore, Von Glasersfeld (1984) presents a version of constructivism that emphasizes its radical nature. He points out:

> "*A few years ago when the term constructivism became fashionable and was adopted by people who had no intention of changing their epistemological orientation, I introduced the term trivial constructivism. My intent was to distinguish this fashion from the 'radical' movement that broke with the tradition of cognitive representatio*n" (von Glasersfeld, 1992).

This section, which is structured based on the radical version of constructivism, discusses the basic principles of constructivism through a comparison with the basic principles of objectivism.

Inaccessible Ontological Reality

Objectivism, which is at the opposite end of the continuum from constructivism, acknowledges that there is a real world where humans are living. This real world consists of entities that are categorized and structured based on their properties and relations. Because this world is structured correctly and completely, it allows humans to model this real world in their minds (Lakoff, 1987). When humans interact with this world, they can achieve the information related to the properties and relations of the entities in this world, and can build a model in their minds that truly reflects those entities and their relations. Therefore, this world, as an ontological reality, is accessible to human reason.

As with objectivism, constructivism does not reject the existence of a real world. However, it views the world as inaccessible to human reason. Humans have no way to know what the objective, universal reality might be. Instead, they construct their own reality based on their own experience (Jonassen, 1991). In other words, they only have access to their own world of experience. Everything of the world that a person could know is based on their own personal experience and then would always be subjective (Proulx, 2006). As von Glasersfeld explained:

> "*It is argued that from the experiencer's point of view, ontological reality is like a 'black box,' in that he has no way of discovering what 'is' and how it might be structured. Here it should be stressed that radical constructivism does not deny the existence of a world, but it does deny the possibility of rationally describing such a 'real' world. The cognizing organism would have no way of determining or deciding whether or not its constructs in any sense reflect the structure of a 'real' world, even if it could come up with structures that are not dependent on its concepts of space and time.*" (von Glasersfeld, 1979)

According to constructivism, the world can never be known in a single way (Vrasidas, 2000). "*The physical world sets certain boundaries within which multiple perspectives can be negotiated and constructed.*" (Vrasidas, 2000). Humans build their own realities based on their own experience. It is impossible that two people have an identical 'reality'. As Jonassen (1991) mentioned, "*We all conceive of the external reality somewhat differently, based on our unique set of experiences with the world and our beliefs about them*". In other words, the reality is specific to individuals (Vrasidas, 2000).

In a nutshell, objectivism views the mind as a mirror of reality while constructivism believes that the world is a product of humans' minds (Jonassen, 1991).

No True Knowledge

Objectivism claims that knowledge exists externally and independently of the mind of individuals (Hannafin, 97). Knowledge has been viewed by objectivists as a kind of entity which can be transferred 'inside' humans' minds (Bednar et al., 1991). On the other hand, constructivism claims that there is no 'objective' knowledge existing externally in the world. Knowledge is personally constructed by individuals based on their own experience. In this case, knowledge is actually subjective and affected by individuals' own subjective view of things (Proulx, 2006).

Objectivists believe in the existence of 'true' and 'absolute' knowledge that is static and fixed (Proulx, 2006). According to objectivism, the 'true' knowledge has to match with the objective reality. The meaning of 'match' used here is that there is a direct 'mapping' relation between the object in the real world and the object in humans' minds. These two objects can be viewed as exactly the same as each other (Proulx, 2006). In other words, knowledge is actually an image of reality.

Constructivism rejects the existence of 'true' and 'absolute' knowledge. As von Glasersfeld (1989) claims, *"Knowledge cannot and need not be 'true' in the sense that it matches ontological reality; it only has to be 'viable' in the sense that it fits within the experimental constraints that limits the cognizing organism's possibilities of acting and thinking"*. The term 'Truth' in objectivism is replaced by 'Viability' in constructivism (Dougiamas, 1998), and the term 'Match' in objectivism is replaced by 'Fit' in constructivism. 'Viability' means that the knowledge is capable of supporting humans to accomplish a task or achieve a goal (von Glasersfeld, 1998), while 'Fit' means that the knowledge is compatible with the experiential world (Proulx, 2006).

Learning is an Active Process

According to objectivism, humans can study the external world by identifying its structure and entities along with their properties and relations, which can be represented through abstract symbols and theoretical models (Vrasidas, 2000). Objectivists believe that humans' mind are like a computer, which is able to process

abstract symbols in a computer-like style. These symbols are assigned with meaning when an external and independent reality is 'mapped' onto them through the interaction between the humans and the world (Bednar et al., 1991). In other words, "*Knowledge and learning are achieved when the abstract symbols that the learner came to know correspond to the one and only real world*" (Vrasidas, 2000). According to objectivism, the outcome of learning is one and only one correct understanding of a topic. The goal of instruction is to transfer the objective knowledge into the learners' head in an effective and efficient way (Bednar et al., 1991; Vrasidas, 2000). Therefore, learning is actually a passive process of accumulating knowledge, and a learner is actually a passive 'receiver' of objective knowledge.

On the other hand, constructivism believes that learning is an active, recursive and elaborative process (Proulx, 2006). Learners actively construct knowledge by combining the experiential word with existing cognitive structures, rather than passively, linearly acquiring and accumulating knowledge from lectures or textbooks (Ben-Ari, 2001a). According to constructivism, learners are not empty glasses that are waiting to be filled by teachers with the objective knowledge from an external world. Instead, they actually perceive and interpret their own reality based on their prior knowledge and experience. For constructivists, prior knowledge and experience play a central role in the learning process. Learning does not start from nothing. Instead, learners interpret and adapt new experience in relation to their previous understandings (Proulx, 2006). This reveals how important it is to take learners' prior knowledge into account when designing and implementing the teaching. (Ben-Ari, 2001a). Proulex (2006) claims that:

> "*In a sense, constructivism asserts that our previous experiences serve as lenses through which we read the world. So to speak, this means that everything we encounter is 'judged' in relation to what we already know.*"

2.1.2 Constructivism in Computer Science Education

Even though constructivism has been widely studied in science education, much less work has been done in computer science education (Ben-Ari, 2001a). Ben-Ari (2001a), one of the most influential researchers on constructivism in computer science education, reveals that "*a (beginning) computer science student has no **effective model** of a computer*".

Ben-Ari defined an ***effective model*** as a cognitive structure based on which learners can make viable constructions of knowledge by combining sensory experiences such as reading, listening to lectures and working with a computer. The computer world is a human-made world. "*Since computer science deals with artifacts—programming languages and software, the creator of the artifact employed a very detailed model and the learner must construct a similar, though not necessarily identical, model*" (Ben-Ari, 2001a). As Lui et al. (2004) have highlighted, "*Computer programming is all fabricated that finds few parallels in the physical world*". The experience gained in every day life may not help learners construct viable mental models of computer artifacts. Hence learners often misuse their prior knowledge or adopt intuitive models, which have been viewed as doomed to be non-viable by Ben-Ari (2001a), to solve computer-related problems.

Ben-Ari (2001a) emphasizes that the models of computer artifacts have to be explicitly taught and discussed. As mentioned above, students often go to a computer science course, such as a programming course, without the 'effective' mental models that are required for them to construct an appropriate understanding of learning materials. Instead, they often hold intuitive, non-viable mental models. Instructors cannot expect students to be capable of constructing viable mental models by themselves through listening to lectures or reading textbooks. If they do, students may construct inappropriate mental models based on the misuse of their prior knowledge and intuitive models. In this case, instructors have to explicitly help students build viable mental models at an early stage in order to avoid haphazard construction of mental models by students themselves. In addition, Ben-Ari (2001a) highlights the issue about how detailed a model should be. Obviously, it is

unreasonable to teach beginning students a computer model in terms of electronic properties of semiconductors. The extent and fidelity of a model should be designed to be suitable for the current level of the students.

In addition, Ben-Ari (2001a) also argues that computer science courses should not start with abstraction. This issue is obviously very important as current introductory programming courses often use an object-first teaching paradigm (Bruce, 2004), i.e. starting by introducing abstract concepts, such as class and object. Unlike professional programmers who use abstractions generally based on a good understanding of the underlying models (Ben-Ari, 2001a), beginning students do not have these underlying models such as memory models. Without these underlying models, beginning students are not able to construct appropriate understanding of abstract object oriented programming concepts. In this case, Ben-Ari (2001a) suggests that instructors have to design their courses carefully to ensure the underlying models are explained properly first.

2.2 Mental Models

Constructivism emphasizes the importance of mental models in computer science education (Ben-Ari, 2001a). Learning to program involves the construction of viable mental models of basic programming concepts (Bayman & Mayer, 1983). This section presents an introduction to mental models, covering their definition, functions, and characteristics.

2.2.1 What are Mental Models?

Craik (1943) first proposed the idea that humans represent the world they are interacting with through mental models, which can be constructed from perception, imagination, or the comprehension of discourse. He postulated that the mind constructs ‘small-scale models’ of reality, which can be used to anticipate events, to reason, and to underlie explanation.

The concept of mental models seemed not attract too much attention until 40 years later. In 1983, an important year in the history of mental model research, two influential books were published. Both of them are named as 'Mental Model' but hold very different meanings. Actually, these two books established the two key directions in mental model research.

The first 'Mental Model' book (Johnson-Laird, 1983) presents Johnson-Laird's theory of mental models, which has been viewed as one of the most influential theories formulated in cognitive psychology (Sasse, 1997). This theory views mental models as working memory constructs that support immediate logical reasoning (Gentner, 2002). Johnson-Laird (1983) argues that humans' reasoning activities are not only based on the logical inference rules but also involve constructing mental models to represent semantic content. A viable mental model must hold a similar structure to the phenomenon it models. The structural similarity is a necessary condition for the person who holds this model to produce appropriate inferences about the phenomenon. However, a mental model does not have to be as complex as the phenomena it represents, but rather it can be much simpler. Actually, the additional information beyond a certain level does not make the model more useful (Sasse, 1997).

The second 'Mental Model' book, edited by Gentner & Stevens (1983), includes a collection of papers that studies mental model in a different direction from Johnson-Laird's, which "…*seeks to characterizing the knowledge and processes that support understanding and reasoning in knowledge-rich domains*" (Gentner, 2002). Unlike Johnson-Laird who views a mental model as a working model to support human reasoning, researchers in this direction view a mental model as a model stored in the long-term memory and used to support humans to generate predications about what should happen in various situations (Collins & Gentner, 1987). For example, when a person holds a model of how a heavy object moves when it is thrown up into the sky, the existing model can help the person predict that a basketball will fall down from the highest point when it is thrown into the sky.

Although the two research directions of mental models mentioned above have very different focuses, it is valuable to bring them together (Gentner, 2002). Previous research (Schwartz and Blank, 1996) shows that the long-term mental model may influence the construction of the short-term working model.

While the Johnson-Laird's theory presented in the first book explains human thought in general, the papers collected in the second book focus on the mental models of a variety of specific domains, such as natural phenomena and devices. Sasse (1997) suggests that these papers present some evidence that mental models and the mechanisms by which they are constructed may differ in terms of task or problem domain. One of these papers, written by Norman (1983), focuses on the mental models of computer systems. It is also one of the earliest works exploring humans' mental models of computer systems.

Norman (1983) suggests that a user constructs the mental models of computer systems through interacting with the target systems, and constantly refines the models throughout the interactions. He argues that the following concepts have to be considered when investigating humans' mental models of computer systems:

- **Target System** – the system with which a user is interacting, such as hardware devices and software.
- **Conceptual Model –** invented by teachers, designers, scientists, and engineers to provide an accurate, consistent, and complete representation of the target system.
- **Mental Model** – a user's internal model of the target system, which is formulated as a result of interaction with the target system.
- **System Image –** the implementation of the conceptual model, which consists of the aspects of the target system with which the users can interact.
- **Scientist's Conceptualization of the Mental Model** – a researcher's understandings of the user's mental model.

A user is expected to construct a viable mental model of the target system. The conceptual models, placed between the target system and the user's mental model, are devised as tools for the understanding or teaching of the target system. In order to

ensure the mental model constructed by the user is viable, the conceptual models, implemented and presented to the user though the system image, have to be accurate and consistent with the target system.

2.2.2 The Functions of Mental Models

Norman (1983) proposes three functional factors of mental models:

- **Belief system -** mental models can reflect the holders' beliefs about the target system.
- **Observability –** there is a correspondence between the parameters/states of the mental model and the target system that the person can observe.
- **Predictive power** - the purpose of a mental model is to support people to understand and anticipate the behaviors of a target system.

In addition, Gentner (2002) claims that mental models can facilitate learning. She used Kieras & Bovair (1984) and Gentner & Schumacher (1986)'s findings to explain this function of mental models. Kieras & Bovair found that the subjects who held a mental model of a simulated device can operate this device more accurately and be able to diagnose malfunctions better than those who merely grasp how to operate it procedurally. Gentner & Schumacher (1986) suggest that the subjects who had a causal mental model of the operation of the first device were able to transfer an operating procedure from one device to another.

2.2.3 The Characteristics of Mental Models

Researchers (e.g. Norman (1983)) have identified some characteristics of mental models through observing humans' interaction with systems. The following briefly describes those characteristics:

1. Mental models are incomplete and simplified (Norman, 1983). Due to the limitations of background knowledge and expertise, the users cannot construct a complete mental model that covers all the details of the target system. Mental models are not expected to be as complex as the target system. An incomplete

and simplified mental model is appropriate if it can support its holder in accomplishing a task. That means mental models need not be technically accurate, but they have to be functional (Norman, 1983).

2. Mental models are unstable over time (Norman, 1983). Mental models are not static but rather keep evolving as their holders interact with the target system. On the other hand, the holders of the mental models can forget some details of the mental models over time, especially when they have not interacted with the target system for a period.
3. There are time delays involving in mental model changing (Doyle et al., 2001). When a mental model is changed old information is replaced by new information. However, the old information is not removed immediately from memory but rather persists in memory alongside the new information.
4. Mental models have vague boundaries (Norman, 1983). The mental models of target systems that have relations or similar properties can be mixed up. Doyle et al. (2001) explains this characteristic of mental models as being caused by the structure of human memory, in which information is interconnected in a complex network of association.
5. People are capable of holding two or more inconsistent models within the same domain, and the inconsistencies may never come to the users' attention (Gentner, 2002). People can use different models in different contexts. As Gentner (2002) mentioned, people, especially novices, "*often use locally coherent but globally inconsistent accounts, often quite closely tied to the details of the particular example*". The inconsistencies may be a result of the cue-dependent nature of human memory recall. The cue-dependent nature refers to the fact that people need an appropriate reminder or 'cue' to locate a particular piece of information in memory (Doyle et al., 2001). The choosing of mental models may depend on what external cues have occurred in the particular problem.
6. Mental modes are 'unscientific' and often contain 'superstitions' (Norman, 1983). People often maintain superstitious behaviour patterns, even when they know they are unneeded. For example, some users often first return to the home directory when they want to shut down or to log out a computer. These users explain that they know it is unnecessary but they feel more confident and

comfortable to behave in this way. Norman explains this as the kind of behaviours that cost little in physical effort and save mental effort.

7. Mental models are parsimonious (Norman, 1983). People will often rather do extra physical operations than mental planning that could help them save physical effort. This implies that people tend to avoid complexity in mental models, and prefer to use physical effort to replace mental effort.

It is interesting to investigate the characteristics of people's mental models. The uncovering of those characteristics could help researchers understand people's behaviours when they are interacting with the world.

2.3 Cognitive Conflict

One of the key teaching strategies based on constructivism is the *cognitive conflict* strategy. This was developed based on the assumption that students' prior knowledge and existing conceptions affects how they learn new knowledge and construct new conceptions. Cognitive conflict is a state in which the student perceives the discrepancy between his or her cognitive structure and external environments or between the components of his or her cognitive structure (Lee & Kwon, 2001).

Students' existing conceptions have been found often to conflict with the scientific ones, and these unscientific conceptions held by students would prevent them from accepting the scientific ones presented in classes (Hewson & Hewson, 1984). When a student is taught a scientific conception, he or she might retain and continue to use their existing, unscientific conception to interpret the new information presented by teachers. They are likely to give the new information meanings that differ from or conflict with the scientific one. It is possible that the student does not realize that their understanding is inconsistent with the meanings presented by teachers. On the teacher's side, they also lack awareness of the inappropriate understandings held by students (Nussbaum & Novick, 1982).

In this case, learning is not simply a case of appending knowledge to students' existing knowledge, but rather it involves changing the students' existing, unscienfic conceptions. Traditional instruction has been viewed as ineffective to accomplish that (Eryilmaz, 2002). Students use their existing conceptions to understand and function in their world. These existing conceptions are resistant to change. It is not easy for students to discard their ideas and adopt a new conception (Davis, 2001). Students often cannot even realize that their existing conception is inconsistent with the taught, scientific conception (Hewson & Hewson, 1984). Therefore, it is important that the teaching should first be able to help students to realize that their existing conceptions are unscientific, and then help them to construct scientific ones.

As Maier (2004) mentioned, "…*the way to resolve or prevent misconceptions is to have the learner confront the misconception directly with an experience that causes disequilibration followed by sound accommodation*". The *cognitive conflict* teaching strategy follows this way that explicitly challenges students' existing ideas in order to encourage the students to recognize problems in their understanding and to motivate them to construct appropriate understandings (Scott et al., 1992). Generally, the cognitive conflict teaching strategy involves: 1) investigating students' prior knowledge and existing conceptions; 2) challenging students with contradictory information; 3) evaluating the conceptual change between students' prior ideas or beliefs and current ones (Limón, 2001).

This section presents the main theory and models that provide theoretical support to the cognitive conflict teaching strategy, including *Piaget's Equilibrium Theory* (Piaget, 1977), Posner et al. (1982) and Hewson & Hewson's (1984) *Conceptual Change Model*, and Lee & Kwon's (2001) *Cognitive Conflict Process Model*.

2.3.1 Piaget's Equilibrium Theory

Piaget, one of the most influential theorists in human cognitive development, has been viewed as the main pioneer of constructivism (Proulx, J. 2006). Piagetian Theory has significantly influenced the development of constructivism. The

equilibrium theory is an important part of Piagetian Theory, which promoted the development of the cognitive conflict teaching strategy. This section presents the equilibrium theory.

Piagetian theory suggests that human intellectual development is driven by the two "most general" biological functions of *organization* and *adaptation* (Piaget, 1952). Organization is a function to form cognitive structure, termed a 'scheme' by Piaget (1977). It is an innate ability that allows humans to systematize the behaviours and thoughts into coherent structures. Adaptation is a function to adapt a person to the environments with which the person is interacting, which encompasses two "*inseparable, synergistic and entwined*" processes, *assimilation* and *accommodation* (Piaget, 1977).

Assimilation is a process that integrates new knowledge into a person's existing scheme. This process takes place when a person tries to use their existing schemes to make sense of the world. What the person is trying to do is to match the new experience with their existing schemes. The assimilation process tends to adapt the experience or information experienced by the person to the existing schemes, rather than making changes to the existing schemes. On the other hand, another process of adaptation, accommodation, tends to adapt the existing schemes or directly create a new scheme to fit the new experience or information. (Piaget, 1977).

Assimilation and accommodation are important for the *equilibration* process. Piaget claimed that intellectual advances need to pass through "multiple non-balance and reequilibrations" (Piaget, 1977). There is a biological drive to chase an optimal state of cognitive balance between a person's cognitive structures and the environment with which the person is interacting. An equilibration process covers the stages of disequilibrium and reequilibration. Firstly, cognitive equilibrium is at a lower developmental level; then cognitive disequilibrium is triggered by confronting puzzling, contradictory, and discrepant information; and then, cognitive reequilibration is achieved at a higher developmental level as the result of reconceptualization. When a person interacts with the environment, if their existing

cognitive structure works well to explain the new experience, the person is in the state of equilibration. When the existing cognitive structure fails to make sense of new experience and information, disequilibrium motives the person to chase reequilibration by accommodating the existing cognitive structure. The person will go up to a higher developmental level of state of equilibration after the accommodation process. There is no absolute end to the process of equilibration. As Piaget (1977) claimed, "no balanced structure can be said to remain in a final state even if it is found to conserve its special characteristics without modifications".

Although Piagetian Theory was built based on the observation on children, the Equilibrium theory actually reveals the general process of intelligence development of human beings, no matter whether children or adults. It would be interesting to investigate whether or not a learning model that is developed based on the equilibrium theory can benefit adults.

2.3.2 Conceptual Change Model (CCM)

Another one of the most widely accepted and influential theories underlying cognitive conflict teaching strategies is the Conceptual Change Model (CCM) proposed by Posner et al. (1982) and Hewson & Hewson (1984).

The conceptual change model suggests that a necessary condition of conceptual change is that the students have to be dissatisfied with the conceptions they are currently holding. It is less likely for a student who is satisfied with their current conception to accept a new conception that conflicts with the current one.

In addition, the conceptual change model also suggests that a new conception has to satisfy three conditions before the students can accept it.

Firstly, the new conception has to be *intelligible*. That means "*…the person considering it has to know what it means, has to be able to construct a coherent representation of it, and has to see that it is internally consistent, without necessarily*

believing it to be true" (Hewson & Hewson, 1984). An 'intelligible' conception does not have to be viewed as true by the students. In the traditional teaching model, teachers generally focus on how to make a new conception to be intelligible to students.

Secondly, the new conception has to be plausible. That means the person must "*…believe it to be potentially true, to be consistent with his or her world view*" (Hewson & Hewson, 1984). A 'plausible' conception has to be 'intelligible' first. The person has to know what the new conception is before he or she goes to believe it to be true. In the traditional teaching model, teachers generally assume that students could automatically view a conception as plausible when they are taught the meaning of the conception. However, the fact is that students are often not able to reconcile the conception with their ideas about the world in general.

Lastly, the new conception has to be fruitful. That means "*…there has to be some good reason before a person will incorporate a new conception, particularly if it is at the expense of an existing conception*" (Hewson & Hewson, 1984). A new conception might be viewed as fruitful if it can solve a previously unsolved problem or provide a more reasonable explanation or prediction of a phenomenon.

Apart from the conditions for conceptual change, the conceptual change model also emphasizes the notion of *conceptual ecology*. Conceptual ecology comprises all kinds of knowledge and beliefs that a student possesses, including the student's prior knowledge and existing conceptions, relationships among various concepts, new knowledge about alternative conceptions, and epistemological beliefs (Davis, 2001). According to Hewson & Hewson (1984), the most important constituents of a student's conceptual ecology are the epistemological commitments to generalizability, internal consistency and parsimony.

Hewson & Hewson (1984) provided an explanation of how cognitive conflicts promote learning based on the conceptual change model. Firstly, both the student's existing conception and the new conception have to be intelligible to the student.

There is no conflict if the student does not understand the conceptions. When both conceptions are intelligible, the student is able to compare them and achieve cognitive conflict. There are two ways to resolve the conflict. The first way is to limit the extent of internal consistency that results in the compartmentalization of their knowledge. In this case, both conceptions would be plausible within their own knowledge subsets. The second way is to accept the plausible conception and reject the other.

The conceptual change model had become one of the most important theoretical models of conceptual change since it was proposed. However, some criticisms were made of this model. The main criticism is that this model only focuses on the cognitive components of learning, but ignores the affective components such as motivation, values, and interests, and social components of learning (Davis, 2001).

2.3.3 Cognitive Conflict Process Model

Lee & Kwon (2001) proposed the Cognitive Conflict Process Model that can be used to anticipate how students might experience cognitive conflict. This model covers three stages (Figure 2-1): Preliminary Stage, Conflict Stage, and Resolution Stage.

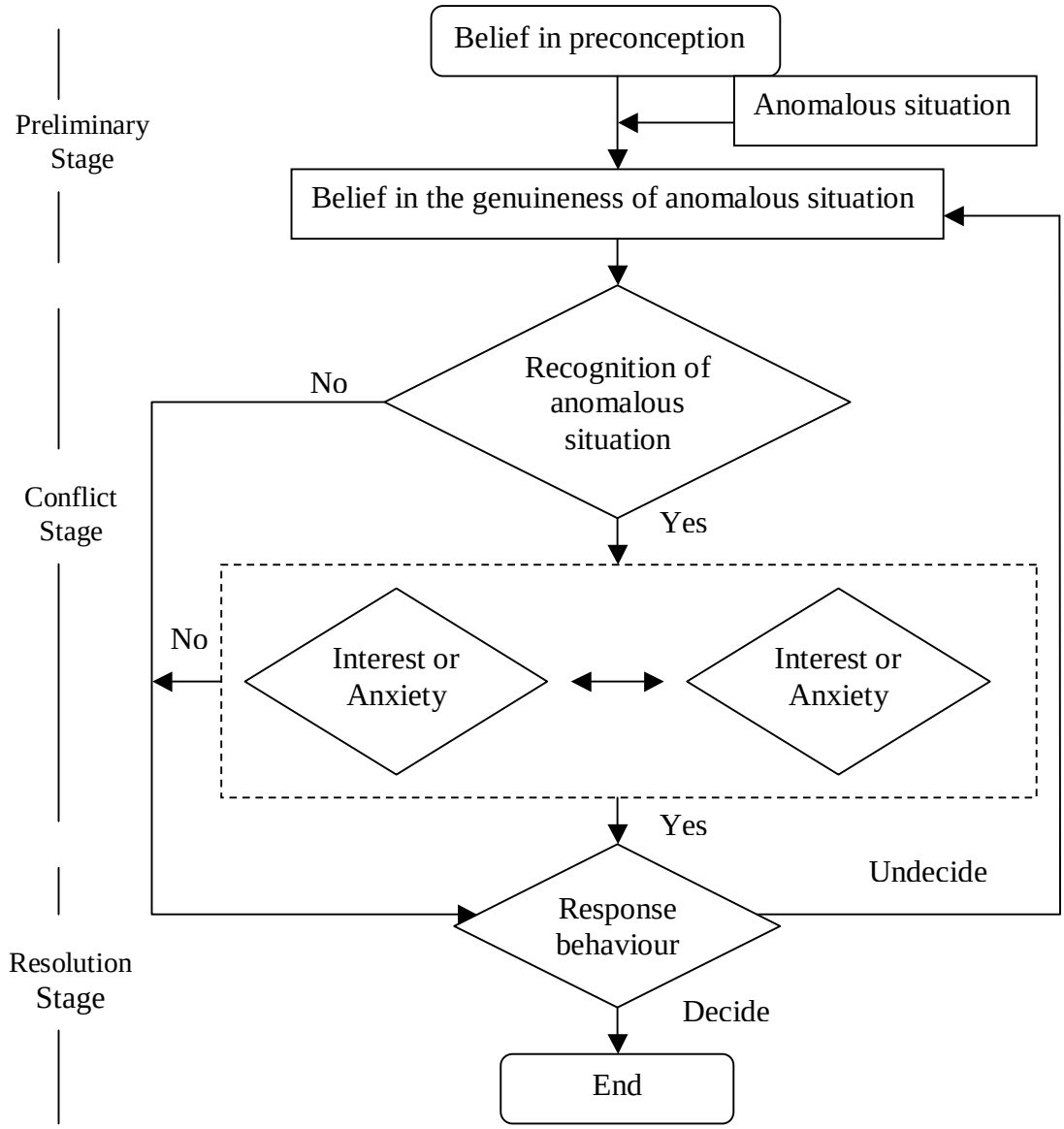


Figure 2-1: Cognitive Conflict Process Model (Lee & Kwon, 2001)

At the preliminary stage, the student who has belief in their existing conception accepts an anomalous situation (e.g. the experimental results) as genuine. The student would not be at the state of cognitive conflict as they do not consider the anomalous situation as a deception. The preliminary stage is the stage prior to the cognitive conflict process.

The cognitive conflict process occurs when a student experiences three activities:

- Recognizes an anomalous situation;

- Expresses interest or anxiety about resolving the cognitive conflict;
- Engages in cognitive reappraisal of the situation.

A student first *recognizes* that the anomalous situation is inconsistent with their conceptions, and then the student would be *interested* in or *anxious* about this situation. After this or simultaneous with this, the student *reappraises* the cognitive conflict situation and decides to resolve or dismiss it.

At the resolution stage, the cognitive conflict is resolved. It results in the production of a variety of external response behaviours. Lee & Kwon (2001) suggest that a external response behaviour would be one of those behaviours proposed by Chinn and Brewer (1998): ignoring, rejection, uncertainty, exclusion, abeyance, reinterpretation, peripheral theory change and theory change, and those proposed by Chann, Burtis and Bereiter (1997): sub-assimilation, direct assimilation, surface-constructive, implicit knowledge building and explicit knowledge building.

In the cognitive conflict process model, there are four psychological components of cognitive conflict: recognition of anomalous situation, interest, anxiety and cognitive reappraisal. These components lead to either constructive or destructive outcomes of cognitive conflict. If a student does recognize the anomalous situation, experiences strong interest and/or appropriates anxiety, and reappraises the cognitive conflict situation properly, the outcome of cognitive conflict is constructive. On the other hand, if a student does not recognize the anomalous situation, the response behaviour of this student is just ignoring the anomalous situation; if a student recognizes the anomalous situation but experiences inappropriate anxiety (e.g. being frustrated or being threatened), the outcome of cognitive conflict could be destructive.

2.4 Summary

This chapter presents the theoretical knowledge that provides the underlying support for this research. The general principles of constructivism have first been introduced and compared with the general principles of objectivism, followed by an introduction to constructivism specifically applied to computer science education. Two concepts related to constructivism, mental models and cognitive conflict are then described.

Firstly, the definition, functions, and characteristics of mental models are introduced, followed by a description of the cognitive conflict teaching strategy and then the main theory and models underlying this teaching strategy are discussed, covering Piaget's Equilibrium Theory, the Conceptual Change Model, and the Cognitive Conflict Process Model.

The next chapter presents the previous work conducted by other researchers that is related to this research.

CHAPTER 3 – Related Work on Mental Model Studies, Visualization, and Cognitive Conflict

Chapter 2 introduces theoretical knowledge of constructivism, mental models and cognitive conflict. This chapter presents previous studies related to the investigations of mental models held by novice programmers, visualizations used in programming education and the assessment of the effectiveness of the cognitive conflict strategy. Firstly, six previous studies that investigated the mental models held by novice programmers are discussed in section 3.1, with special emphasis on the mental model elicitation methods used in these studies. A survey of visualization systems that were specifically developed for programming education and previous empirical studies of the effectiveness of visualization used in programming education are then presented in section 3.2. Lastly, section 3.3 describes previous empirical studies on the effectiveness of the cognitive conflict strategy.

3.1 Investigating the Mental Models Held by Novice Programmers

It is important to study students' mental models when designing teaching material. Gentner (2002) explains the importance of mental models investigation as: 1) "*the errors that a learner makes can help reveal what the learning processes must be*"; 2) "*if typical incorrect models are understood, then instructors and designers can create materials that minimize the chances of triggering errors*".

Although a large number of studies have been conducted to investigate people's mental models of natural phenomena and interactive devices in cognitive science and Human-Computer Interaction domains, not many studies have been carried out to investigate novice programmers' mental models of programming languages and basic programming concepts. This section surveys previous studies that were specifically conducted to investigate the mental models held by novice programmers.

Six previous studies (Table 3-1) are presented in this section. For each study, the research *goal*, *method* employed, and *findings* achieved are described. In addition, there is a discussion on the strengths and weaknesses of the mental model elicitation method employed in the study, with special emphasis on the suitability of the method in the educational context.

One of goals of this thesis is to suggest an effective and practical 'method' or 'technique' for instructors to investigate the mental models of basic programming concepts held by their students. In order to integrate a mental model elicitation method into the 'real' teaching process, this method has to be easier to implement and less time-consuming than the methods employed in the research context. Generally, there are a large number of students in an introductory programming class. In practice, the instructors, especially those college lecturers who often need to do their own research work, cannot afford a time-consuming method that is suitable for studying a small population of students. This is one possible reason why only a few of the mental model studies were integrated into the 'real' teaching process even though many researchers have proposed that the understanding of students' pre-existing mental models are crucial for preparing suitable teaching materials (e.g. Gentner, 2002). In this case, 'time cost' is an important criterion to evaluate whether or not a mental model elicitation method is suitable to be integrated into a teaching process. The following stages are involved in a mental model study:

- *Preparation* Stage: prepare the materials and mental model elicitation activities;
- *Implementation* Stage: conduct the mental model elicitation activities;
- *Data Collection* Stage: collect the data from the mental model elicitation activities;
- *Data Analysis* Stage: analyze the acquired data.

The evaluation of the suitability of a mental model elicitation method needs to consider the time cost for each stage.

No.	Researchers	Studied Concept(s)	Programming Language	Elicitation Activity	Elicitation Techniques	Participants
1	Dehnadi & Bornat (2006)	Assignment; Sequence	Java	Close-ended Prediction	Questionnaire	61 participants took part in this study. Over 30 participants were from an adult education course; the others were from a first year programming course
2	Sasse (1997)	Object, Variables, Rule	Prolog	Open-ended Prediction + Verbal data	Interview	18 participants who had received 5 weeks' instruction and practice
3	Bayman & Mayer (1983)	Variable, Assignment, Input/Output, Control Flow	BASIC	Explanation	Questionnaire	30 college undergraduates who had no prior experience with computer, or programming
4	Kahney (1983)	Recursion	SOLO	Close-ended Prediction; Explanation	Questionnaire	30 novice programmers and 9 expert programmers
5	Kurland & Pea (1985)	Recursion	LOGO	Explanation	Interview	7 children, 11 years old or 12 years old, who had learnt Logo programming over 50 hours
6	Yehezkel et al. (2005)	Computer Archtecture	Assembly Language	Explanation	Interview	11 tenth-grade students who had received a one-year, 90-hour course in computer architecture and assembly language programming

Table 3-1: Related work on the investigation of novice programmers' mental models

3.1.1 Dehnadi and Bornat's (2006) Study on the Consistency of the Mental Models Held by Beginning Students

Goal of the study

This study aimed to investigate the relationship between the consistency of mental models and programming aptitude.

Methods

Dehnadi and Bornat (2006) devised a questionnaire (Appendix A-1) to investigate the mental models that students used when thinking about assignment statements. This questionnaire includes twelve questions. Each question presented a small sequence of assignment statements. Students were asked to predict the values held by the variables after the execution of the program. Figure 3-1 shows a typical question.

Read the following statements and tick the box next to the correct answer in the next column.

```
int  a = 10;
int  b = 20;
a = b;
```

Figure 3-1: A typical question in Dehnadi and Bornat' questionnaire

A collection of mental models (table 3-2) that a student might use to answer the questions was also proposed by Dehnadi and Bornat, using their teaching experience from introductory programming courses. For the question in figure 3-1, the model Mv2 is marked as the appropriate model while all the others are inappropriate. Apart from the mental models listed in table 1, Dehnadi and Bornat also proposed a *simultaneous* mental model that the statements are executed simultaneously rather than sequentially.

Model No.	Model Descriptions
Mv1	Value moves from right to left (a <- b; b <-0)
Mv2	Value copied from right to left (a <- b; b unchanged) '**Appropriate**' model
Mv3	Value moves from left to right (a ->b; a <-0)
Mv4	Value copied from left to right (a ->b; a unchanged)
Mv5	Right-hand value added to left (a <- a+b; b unchanged)
Mv6	Right-hand value extracted and added to left (a<- a+b; b <-0)
Mv7	Left-hand value added to right (a+b ->b; a unchanged)
Mv8	Left-hand value extracted and added to right (a+b->b; a<-0)
Mv9	Nothing happens (a, b unchanged)
Mv10	A test of equality
Mv11	Variables swap values

Table 3-2: Mental models for value assignment question (Dehnadi & Bornat, 2006)

Dehnadi and Bornat administered two tests: the first test was conducted before the participants had received any programming teaching (week 0); the second test was conducted to the same participants after the topics had been taught (week 3). 61 participants were involved in the tests. About 30 of them came from an adult education course. They had no particular pattern of age, gender or education background. Dehnadi & Bornat (2006) believed that these participants had no previous contact with programming. The rest come from the first year programming course.

Findings

The result of this test divided participants into three distinct groups:

- ***Consistent group*:** in which the participants used the same model for all, or almost all, of questions.
- ***Inconsistent group*:** in which the participants used different models for different questions.
- ***Blank group:*** in which the participants refused to answer all or almost all of the questions.

Most of the participants (21 out of 27) in the consistent group (based on the first administration of Dehnadi's test) scored a pass mark of 50 or above in the end-of-course exam, while most of those (26 out of 34) in the inconsistent group and blank group scored below 50. The analysis of the correlation between the first and second administration of Dehnadi's test showed that there were 13 participants who transferred from the inconsistent group to the consistent group, but no one moved in the opposite direction.

Dehnadi and Bornat speculated that the three groups were distinguished by their different attitude to *meaninglessness*. The consistent group was thought to accept the fact that "*the machine will blindly follow its meaningless rules and come to some meaningless conclusion*". On the other hand, the inconsistent group was thought as seeking meaning where it was not. The blank group was thought as realizing the meaninglessness, but refusing to deal with it.

Dehnadi and Bornat believed that they had found a test to "*predict success or failure even before students have had any contact with any programming language with very high accuracy"*. They thought it was extremely difficult to teach programming to the inconsistent and blank groups, while it was much easier to teach programming to the consistent group. They made the controversial statement that "*programming teaching is useless for those who are bound to fail* (i.e. the inconsistent group) *and pointless for those who are certain to succeed* (i.e. the consistent group)".

Discussion of the Mental Model Elicitation Method used

This study investigated the participants' mental models of assignment and assignment sequence concepts by asking them to predict the result of program execution. There is a collection of pre-defined mental models that the participants might use to make the prediction. This method to elicit students' mental models of programming concepts is easy to implement and places minimal time constraints on the data collection and analysis. At the implementation stage of this study, the mental

models elicitation activity, i.e. asking participants to predict the program execution, was guided by the structured questionnaire. Instructors did not need to be heavily involved in this activity. Compared to other techniques such as interviews, the use of questionnaires are also able to simplify the process of data collection. This study used quantitative data. There was no verbal data involved in the data analysis processes. In this case, the work load of researchers during the data analysis stage is relatively low, especially when the data collection and analysis processes can be automated using technologies such as online questionnaires. In addition, the use of quantitative data allows instructors to analyze the distributions of mental models held by a larger number of students. It may also help instructors design more effective teaching materials if they know what inappropriate mental models are popular in their classes.

Moreover, Dehnadi and Bornat's questionnaire, which uses multiple questions for a single programming concept, allows instructors to investigate the consistency of students' mental models. The consistency of mental models is an important criterion for the viability of mental models. According to Gentner (2002), people often hold inconsistent mental models. They may apply different mental models when confronted with the same problem. Obviously, a mental model is non-viable if the holder of it cannot use it consistently. In this thesis, a **viable** model is defined as meeting two conditions: 1) It has to match with the model of how a programming concept actually works (appropriate); 2) it '*always*' has to match with the actual model (consistent). In addition, the use of multiple questions can reduce the possibility of achieving inaccurate results. It is possible that students make an incorrect answer to the question due to carelessness. The use of multiple questions can help solve this problem because it is less likely that students are repeatedly careless in the same pattern.

On the other hand, the limitation of the mental model elicitation method used in this study is also obvious. This method required instructors to have a collection of pre-defined mental models, covering the mental models held by most of the students. In this case, instructors have to identify those models by surveying previous studies, or must define them by themselves based on their teaching experience. In addition, the

teaching experience of the instructors is often not sufficient to enable them to identify all (or most) of the mental models. In this case, additional methods are required to ensure the coverage of all of the mental models, or at least the typical mental models held by the students in the studies.

In addition, the mental model elicitation method used in this study may produce inaccurate results when an answer option could map to two or more mental models. It is difficult to identify which mental model the participant is holding when they choose this answer option. Collecting additional verbal data may solve this problem and obtain more accurate results. Besides, as mentioned above, the inaccurate results may be caused by participants' carelessness when choosing an answer option. In this study, multiple questions were used to tackle this problem.

3.1.2 Sasse's (1997) Study of the Mental Models of the Prolog Programming Language

Goal of the study

This study aimed to elicit students' mental models of the Prolog programming language.

Methods

18 participants were involved in this study. They had received 5 weeks' instruction and practice with the Prolog programming language, and had also written a small application using Prolog.

During this study, the participants were presented with a short Prolog program that consists of a knowledge base and one rule. Afterwards, they were asked to complete three sets of tasks (Table 3-3):

- **Predict the result of program execution (tasks 1 – 6) –** the participants were first asked to type in a number of Prolog queries, and then predict what the results of the queries were. If the result of the actual program execution was

different from the participants' prediction, they were asked to explain the possible reason.

- **Construct Prolog queries (tasks 7 – 10) –** the participants were asked to construct queries to solve the given problems. If the queries did not return the expected results, the participants needed to explain the possible reason.
- **Write new rules (task 11- 12) –** the participants were asked to write two new rules to meet the given requirements. The first rule (task 11) was analogous to an existing one while the second one was new and therefore a learning transfer task.

Predictions	Could you please predict, after looking at the program listing, which answer the system will return when you type in the following queries:
1 p(o,o)	person(annie, female).
2 p(V,o)	person(X,female).
3 p(o,V)	parent(jenny, X).
4 p(o,o)	parent(sophie, rupert).
5 p(o,o)	parent(carol, rosie)
6 p(o,V)	parent (rosie, Y).
7 Conjunction (TT)	What query would you have to enter to get a list of all daughters in the database?
8 Test rule	Please examine the father rule. Which query, using that rule, would give you a list of all fathers and their children?
9 Place holder	What query would you have used to get the system to list for you only the names of the father, without returning the names of the children?
10 Repetition	Why does the system return some names more than once in response to your query?
11 Analogous rule	Based on the father-rule, write a rule that defines mother.
12 New rule (TT)	Write a rule that defines either brother or sister.

Table 3-3: Sasse's (1997) Prolog Tasks

Findings

The results show that most of participants could make the correct prediction to the first set of tasks, while half of the participants could construct a correct query to test a given rule. Regarding the third set of tasks that require participants to construct new rules, most of participants were able to construct an analogous rule, but none of them could complete the learning transfer task successfully. The verbal data collected from participants showed that most participants lacked an appropriate mental model of objects, variables, and how instantiation works. They could not properly explain what objects and variables were and what they could be used for. Many participants associated the names of objects and variables with their semantic meaning. They believed that the computer program was able to extract the semantic information from the names of objects and variables. Sasse also found that some participants who lacked appropriate mental models could still perform well in the programming tasks. She explains this phenomena as that most of the participants wrote their programs by *trial and error*, with the help of immediate corrections from the investigator. This allowed the participants to pass these tasks even when their mental models were problematic.

Discussion of the Mental Model Elicitation Method Used

In this study, Sasse studied the participants' mental models by asking them to predict the behaviours of the program (the first set of tasks) and testing their performance when solving programming problems (the second and third sets of tasks). Unlike Dehnadi and Bornat's test that allowed participants to predict program execution based on a collection of pre-defined mental models, this study did not pre-define any potential mental models that the participants might have used. In Dehnadi and Bornat's test, a participant's mental model was identified based on their choice of an answer that was mapped to one of the pre-defined mental models. In this study, there was no pre-defined mental model. Participants' mental models were determined by the verbal data provided by them. The prediction tasks and problem solving tasks in this study were actually used to lead participants to provide verbal data. When a participant made an incorrect prediction, they were asked to explain their thoughts. Sasse explained that the verbal data related to errors, e.g. why the error happened and

what the participant expected to happen, and contained rich information about the mental models used by the participant.

Because the mental model elicitation method used in this study did not require the researchers to have a collection of pre-defined mental models, and the researchers did not have to design experiment materials based on those pre-defined mental models either, the time cost of the preparation stage in this study was less than that in Dehandi and Bornat's test. However, this study also employed the interview technique. The researchers had to be highly involved to guide the interview and collect the data. In addition, the use of verbal data in this study increased the workload of the researchers on data analysis.

In Dehnadi & Bornat's study, the close-ended questions were used to elicit students' mental models. As mentioned above, the results produced may not be accurate when an answer option could map to two or more mental models or when the participants were careless. This study identified participants' inappropriate mental models by collecting and analyzing their verbal data. The use of verbal data could avoid those problems. However, the accuracy of the results depends on the quality of the verbal data provided by participants. People often lack the ability to describe their thoughts. They may provide too little or irrelevant information. In this case, the accuracy of the results would be relatively low.

3.1.3 Bayman and Mayer's (1983) Study of the Mental Models of BASIC Programs

Goal of the study

This study aimed to elicit students' mental models of the BASIC programming language.

Methods

30 college undergraduates who had no prior experience with computers or programming participated in the study. This study was conducted after the participants had successfully completed a self-instruction, self paced course of BASIC programming language.

In this study, nine statements (Table 3-4) were presented to the participants who were asked to explain the execution of each of the statements (i.e. the steps that the computer would carry out for each statement) in plain English.

No.	Statement
1	INPUT A
2	30 READ A
3	IF A<B GOTO 99
4	LET A = B+1
5	20 DATA 80, 90, 99
6	60 GOTO 30
7	PRINT C
8	LET D=0
9	PRINT "C"

Table 3-4: The BASIC statements used in Bayman and Mayer's (1983) study

Findings

The verbal data collected from the participants shows that they often held inappropriate mental models of variables, data storage, and assignment. When the participants were asked to explain the INPUT statement, READ DATA statement, and LET statement, they did not know where the data came from, and how the data was stored in memory. Some participants lacked an appropriate model of variables. They viewed the statement of *INPUT A* as that the letter 'A' was input and stored in memory or they could not understand the difference between the statement *PRINT C* and *PRINT "C"*. Some participants used inappropriate mental models of assignment. They treated the equals sign '=' as equality rather than assignment. For example, they explained the statement *LET A = B+1* or *LET D = 0* as that the equation is stored in memory.

Discussion of the used Mental Model Elicitation Method

In this study participants were asked to describe their thoughts of the program execution by using plain English. The mental model of a participant was identified from their verbal data. This method is actually based on the assumption that students have to mentally simulate the program execution process using a mental model when they try to explain the program execution. It assumes that an understanding of their mental model can be extracted from the participants' verbal description. The use of verbal data increases the time cost at the data analysis stage. However, the use of the questionnaire technique reduced the workload of experimenters at the implementation and data collection stages. In addition, compared to Dehnadi and Bornat's test, the experimenters in this study did not have to identify a collection of mental models that could be held by the participants.

3.1.4 Kahney's (1983) Study of the Mental Models of Recursion

Goal of the study

This study aimed to elicit and compare novice and expert programmers' mental models of the recursion concept.

Methods

Recursion is defined as "*a process that is capable of triggering new instantiations of itself, with control passing forward to successive instantiations and back from terminated ones*" (Kahney, 1983). The mental model corresponding to the definition of recursion is called the 'Copies' model, i.e. the appropriate model of recursion. On the other hand, students are often identified as holding another model of recursion – the 'Looping' model. Kahney (1983) defined this model as that the recursive procedure is viewed "*as a single object instead of series of new instantiations, having the following features: 1) an 'entry point', the constituents of which are the procedure's name and a parameter slot; 2) an 'action part', which is designed to add*

information to the database; 3) a 'propagation-mechanism' for generating successive database nodes back to the 'front part', or 'entry point' of the procedure".

In Kahney's study, a questionnaire was designed to investigate which model, the 'Copies' model or the 'Looping' model, the participants were holding. This questionnaire is based on the problem shown in Figure 3-2.

> There is a database containing a collection of names of people who kiss each other, and the problem needs a computer program to infer: "*if somebody 'x' has 'flu' then whoever 'x' kisses also has 'flu', and whoever is infected spreads the infection to the person he or she kisses, and so on*".

Figure 3-2: The problem need to be solved in Kahney's (1983) study

The questionnaire also gives two solutions to the problem written using SOLO – a LOGO-like programming language, called Solution-1, and Solution-2 (Figure 3-3). The participants were asked to predict which program could solve the problem. In addition, they were also required to explain why the program could (or could not) work.

Solution-1	**Solution-2**
TO INFECT /X/ 1 NOTE /X/ HAS FLU 2 CHECK /X/ KISSES? 2A If Present: INFECT *; EXIT 2B If Absent: EXIT DONE	TO INFECT /X/ 1 CHECK /X/ KISSES? 1A If Present: INFECT *; EXIT 1B If Absent: EXIT 2 NOTE /X/ HAS FLU DONE

Figure 3-3: The programs used in Kahney's (1983) study

Both Solution-1 and Solution-2 can solve the task but by different procedures. The Solution-1 is an example of tail recursion with the *active* flow of control, i.e. control is passed forward to new instantiations (Gotschi et al., 2003), while the Solution-2 is an example of embedded recursion with the *passive* flow of control, i.e. control flows back from terminated instantiations (Gotschi et al., 2003). The 'Looping' model is

actually an appropriate model of tail recursion (Kurland & Pea, 1983), but it is not an appropriate model of embedded recursion. A viable mental model of recursion has to be adequate for both tail recursion and embedded recursion. In this case, the 'Looping' model is not a viable mental model of recursion. Because the Solution-1 is an example of tail recursion and the Solution-2 is an example of embedded recursion, the participants who held the 'Looping' model would think that the Solution-1 is correct but the Solution-2 is incorrect. The 'Copies' model is the viable model of recursion. The participants who held the 'Copies' model would think that both Solution-1 and Solution-2 are correct.

39 participants completed the questionnaire, including 30 novice programmers and 9 expert programmers.

Findings

The results show that a much higher percentage of expert participants held a viable mental model of recursion. 8 out of 9 expert participants selected both Solution-1 and Solution-2. The comments from them indicated that they had a 'Copies' model of recursion. The figure for novice participants who held viable mental models was very low. Only 3 out of 30 novice participants selected both Solution-1 and Solution-2. However, 2 of these claimed that they doubted their understanding of both Solution-1 and Solution-2. In addition, 4 novice participants selected Solution-1 and rejected Solution-2. The evidence is strong that these participants held the 'Looping' model. Apart from the 'Copies' model and 'Looping' model, the verbal data revealed that some participants held other mental models of recursion. For example, some participants held an *Odd* Model that viewed 'flow of control' statements, such as EXIT and Continue, as the stopping rules for recursion.

Discussion of the Used Mental Model Elicitation Method

This study asked participants to predict the behaviours of two given computer programs. These two programs were designed based on two known mental models of recursion. The researchers could identify which one of these known mental models was held by the participant based on their judgement of whether or nor the given computer programs could solve the given problem. This mental model elicitation approach therefore would only produce quantitative data. It could save researchers time on data analysis. However, in this case the quantitative data seems insufficient to elicit all the participants' mental models. As mentioned above, this quantitative data is only related to the two known mental models, 'Copies' model and 'Looping' model. It could not identify other mental models that might be held by many participants. In addition, it is possible that there may be some unknown mental models that map to the same answer option as the 'Copies' model or 'Looping' model and this increases the possibility of producing inaccurate results. Furthermore, this quantitative data could not reveal the detailed information of the participant's mental models. In this case, as was done in this study, additional qualitative data was collected and used to cover the range of mental models held by the participants and detailed information on them.

3.1.5 Kurland & Pea's (1985) Study of the Mental Models of Recursion

Goal of the study

This study aimed to elicit children's mental models of the recursion concept.

Methods

7 children, 11 years old or 12 years old, participated in this study. They were highly motivated, and had learned Logo programming over 50 hours, including the topics of iteration and recursion.

During this study, the participants were provided with a collection of short Logo programs that were used to move a 'turtle'. These programs are at four distinct levels of complexity: 1) there are only direct commands to move the turtle; 2) the iterative REPEAT command was involved; 3) tail recursive procedures were used; 4) embedded recursion procedures were used. The programs at the first two levels were relatively simpler because there was no recursion involved in them. The programs at the last two levels used recursive procedures that were relatively complex. Figures 3-4 and figure 3-5 show the examples of programs at the last two levels.

```
TO SHAPEB:SIDE
  IF:SIDE=20 STOP
  REPEAT 4 [FORWARD: SIDE RIGHT
90]
  RIGHT 90 FORWARD: SIDE LEFT 90
  SHAPEB: SIDE/2
END
```

Figure 3-4: The tail recursion program used in Kurland & Pea's (1985) study

```
TO SHAPEB:SIDE
  IF:SIDE=10 STOP
  SHAPEC: SIDE/2
  REPEAT 4 [FORWARD: SIDE RIGHT
90]
  RIGHT 90 FORWARD: SIDE LEFT 90
END
```

Figure 3-5: The embeded recursion program used in Kurland & Pea's (1985) study

For each level of programs, the participants were asked to explain how the programs would work. In addition, they were also asked to hand simulate the execution of the programs line by line using a graphic turtle 'pen' on paper. Afterwards, they would be shown the actual results of program execution, i.e. how the 'turtle' should move on paper. If the actual result of program execution was not consistent with the way explained by the participant, they were asked to explain the reason.

Findings

The results show that all the participants could successfully complete the first two levels of programs in which there were no recursive procedures. All of them made an

accurate explanation of program execution. Most of the participants could successfully complete the third level of programs that contained tail recursion, except two participants who had an inappropriate understanding of the IF statement. In contrast, none of the participants could properly explain the behaviours of the fourth level of programs that contain embedded recursion. Kurland & Pea (1985) claimed that there were several sources of the participants' difficulties to understand embedded recursion. Firstly, there were general errors in the participants' understandings of programming concepts. Some participants tended to understand each line of code individually, and ignored the context built by the lines that had previously been executed. This problem could also be explained as that these participants held inappropriate understanding of sequential program execution. In addition, some participants viewed programs as conversation-like. They did not think that a task was completed by the execution of a sequence of statements, but rather believed that the programmers could 'tell' the program to do a task by giving it a statement. Moreover, some participants often used their prior knowledge or experience to understand programming concepts. For example, they used natural language semantics to understand the statement END and STOP. Furthermore, some participants held the 'Looping' model of embedded recursion. As mentioned above, the 'Looping' model could only help the participants understand tail recursion (the third level of programs) accurately, but not embedded recursion.

Discussion of the Used Mental Model Elicitation Method

This study investigated participants' mental models of recursion by asking them to predict and explain the execution of the programs that contain recursive procedures. Compared to the Kahney (1983) study in section 3.1.4, the prediction of program execution in this study was open-ended and was not based on a collection of pre-defined mental models. The participants' mental models were identified based on the verbal data collected from them. In addition, this study employed structured interviews rather than questionnaires to collect data. It allowed the in-depth conversation between researchers and participants that could not be obtained by using questionnaires. Researchers could further investigate a participant's thoughts if

this participant gives interesting information. In this case, researchers are able to identify more detailed information about participants' mental models. However, a weakness of structured interviews is that researchers have to be highly involved in the implementation and data collection stage. The researchers, especially those who are teaching a big class, would not be able to afford the associated time cost.

3.1.6 Yehezkel et al.'s (2005) Study of the Mental Models of Computer Architecture

Goal of the study

This study aimed to study the effectiveness of visualization to improve novice students' mental models of computer architecture.

Methods

11 tenth-grade students who had received a one-year, 90-hour course in computer architecture and assembly language programming participated in this study. These participants experienced two learning phases. The first learning phase was merely based on theoretical learning, covering the topics of data representation, computer organization, basic symbolic representation and program execution. The learning materials presented in this phase used a static description of the computer architecture and dynamic information transfer between computer units. The second learning phase employed a visualization environment, EasyCPU, to provide students with a dynamic representation of information transfer. The data flow between the units of the computer, such as CPU register, memory segments, and I/O, are animated when each instruction is executed.

Two interviews were conducted to investigate participants' mental models of computer architecture: the first interview was carried out after the first learning phase when the participants only used static learning materials; the second interview was conducted after they had used the EasyCPU Environment. During both interviews,

the participants were asked to describe the topology of the interconnections between the units (the static viewpoint) and the data transfer between computer units when a specific instruction was executed (the dynamic viewpoint). Six scenarios were designed to test participants' understanding of dynamic data transfer between computer units (Table 3-5).

Type of scenario	First Interview	Second Interview
Internal to CPU	AL <- BL	MOV CH, BL
Internal to CPU with data	AL <- 03h	MOV BH, 02h
Reading memory	AL <- [02h]	MOV BL, [03h] MOV AL, [BX]
Writing to memory	[01h] <- CL	MOV [01h], BH MOV [BX], CL
Reading Input Port	AL <- Input	IN AL, 02h
Writing to Output port	Output <- AL	OUT 03h, AL

Table 3-5: The six scenarios in the interviews of Yehezkel et al.'s (2005) study

There was a small difference between the first interview and the second interview. As table 3-5 shows, symbolic form such as AL <- [02h] was used to present instructions in the first interview. This form was taught to participants in the first learning phase. The assembly language that had been learned by the participants in the second learning phase was used to present instructions in the second interview.

Seven schemas (Figure 3-6) were provided to the participants in the interview to help them to describe their mental models of the static system model and the six scenarios of dynamic models. The schema represents the four main units in a computer (CPU, Memory, Input port, and Output port) with the connections between them omitted.

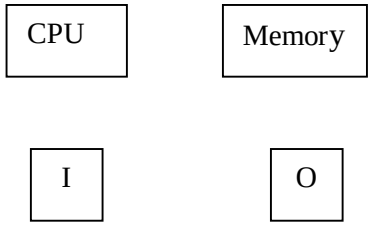


Figure 3-6: The schema used in Yehezkel et al.'s (2005) study

Findings

The results show that there were four mental models held by the participants. They could be categorized into four topologies (Figure 3-7).

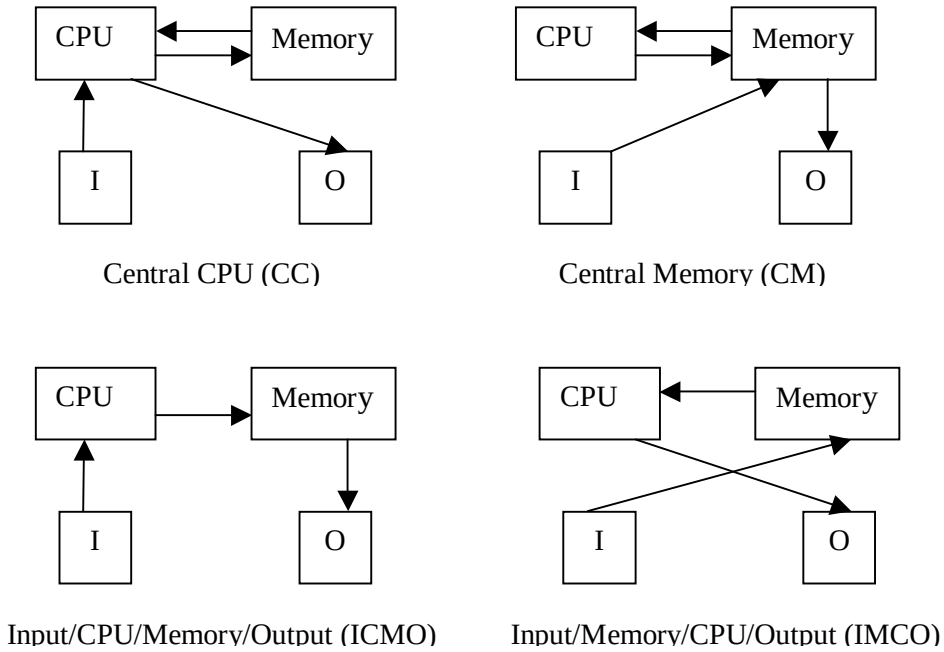


Figure 3-7: The four topologies identified in Yehezkel et al.'s (2005) study

- **CC model** - It is the appropriate model of computer architecture. In this model, the CPU is at the central position and all memory accesses and input-output tasks are carried out via the CPU.
- **CM model –** In this model, memory is at the central position. All the I/O operations are carried out directly to memory.
- **ICMO model –** In this model, the data is linearly transferred through the four units in the computer: Input -> CPU -> Memory -> Output.
- **IMCO model –** In this model, the data is linearly transferred through the four units in the computer: Input -> Memory -> CPU -> Output.

The ICMO model and IMCO model are two variations of the ICO (Input-Central processing- Output) model that views CPU and Memory as a whole: the data flows

from Input to the 'main part' of the computer for processing and then flows out via Output.

In the first interview that happened when the participants had only studied via static learning materials, 5 out of 11 participants demonstrated the ICO model and 2 other participants used the CM model. Only 4 participants were found to be holding the appropriate mental model, i.e. the CC model. In contrast, all the participants demonstrated the CC model in the second interview that happened when the participants had used the visualization environment. This provides evidence that visualization may help students construct viable mental models when these students are exposed to the dynamic processes inside the computer during instruction execution.

Discussion of the Used Mental Model Elicitation Method

This study investigated the participants' mental models based on their explanation of data transfer between the computer units. A structured interview technique was employed to collect the data. This could increase the researchers' time spent at the implementation and data collection stages, although it allows in-depth conversation between the researchers and the participants. In addition, a big difference between this study and the studies mentioned above is that this study used a graphical representation to assist participants to describe their mental models. It is suggested that mental models are picture-like (Sasse, 1997). People might find that it is easier to represent their mental models using graphical representation than using textual description.

3.1.7 Summary of the Mental Model Elicitation Methods

This section summarizes and discusses the mental models elicitation methods used in the studies presented above from two perspectives: the elicitation activities, and the elicitation techniques.

3.1.7.1 The *close-ended predicting* activities versus the *open-ended predicting* activities

Young (1983) suggests that mental models can be elicited when the holders of the mental models are carrying out four activities, including: *using* the target system; *explaining* the target system; *predicting* the behaviours of the target system; *learning* the target system. The review of previous mental model studies introduced above shows that all these studies elicited participants' mental models by asking them to predict the behaviours of a program[1]. Predicting activities seem suitable for studying programmers' mental models of program concepts. While programmers are predicting the behaviors of a program, they have to mentally simulate the program execution based on their mental models. Exposing the programmers' mental simulation process may produce information on the mental models held by those programmers.

Predicting activities could be *close-ended*, i.e. the prediction is based on a collection of pre-defined answers and each answer is mapped to a possible mental model. Two of the studies reviewed above employed the close-ended predicting activities. Dehnadi and Bornat (2006) investigated the consistency of novice programmers' mental models of assignment and sequence by asking participants to predict the results of the execution of small program fragments from a collection of pre-defined answers. Kahney (1983) devised two program fragments of recursion based on two pre-known mental models, i.e. the 'Copies' model and 'Looping' model, and asked participants to predict whether or not the program fragments could solve a given problem. The prediction from participants could reveal whether a participant held the 'Copies' model or 'Looping' model of recursion.

Predicting activities could also be *open-ended*, i.e. participants would be not provided with answer options that are mapped to the potential mental models. Five of the studies reviewed above used the open-ended predicting activities. Sasse (1997) asked participants to predict the result of a number of Prolog queries and explained

[1] In Yehezkel et al.'s study, the participants were also asked to explain the architecture of a computer system.

their thoughts when the actual result of program execution was inconsistent with the participants' prediction. Bayman and Mayer (1983) studied the novice programmers' mental models of BASIC language constructs by asking them to describe what steps the computer would carry out when a BASIC statement was executed. Kurland & Pea (1985) asked children to mentally simulate the execution of the LOGO programs containing recursive procedures to elicit their mental models of recursion. Yehezkel et al. (2005) asked participants to predict the dynamic data transfer between units when a specific instruction was executed in order to study the participants' mental models of computer architecture and assembly programming language. Actually, Yehezkel et al.'s study also employed the *explaining* activity that asked participants to explain the static topology of the interconnections between the units inside a computer system. Unlike the close-ended predicting activities, the open-ended predicting activities are not carried out based on a collection of answer options that are mapped to the potential mental models. Instead, researchers generally identify participants' mental models based on their verbal data that describes their thoughts of program execution.

Two kinds of representations could be used by participants to describe their thoughts, the textual representation (e.g. those were used in all the studies reviewed above) and graphical representation (e.g. that was used in Yehezkel et al.'s study). Because mental models are picture-like (Sasse, 1997), graphical representation seems a more natural way for people to depict their mental models. In addition, people often lack the ability to explain their thoughts (Gentner, 2002). It seems easier for people to describe their mental models by using graphical representation. On the other hand, merely using graphical representation is often not able to provide enough information. Different people may interpret the same graphic in different ways. It is possible that researchers and participants may have different interpretations of a graphic. In this case, additional textual explanation is required.

The use of close-ended predicting activities allows researchers to identify participants' mental models without collecting verbal data from them. As mentioned above, some participants may lack the ability to describe their thoughts. The close-

ended predicting activities seem a more practical way to elicit these participants' mental models. Because there is no verbal data involved, the close-ended predicting activities are easier to implement with little time spent on the data collection and analysis. It especially benefits researchers who aim to investigate the mental models held by a large class of students, especially when the data collection and analysis process are automated using computer technologies, such as online questionnaires and automatic assessment. In addition, when multiple questions are designed for a single concept, the close-ended predicting activities are capable of investigating the consistency of mental models.

On the other hand, there are several weaknesses of the close-ended predicting activities. Firstly, the close-ended predicting activities require researchers to achieve a collection of pre-defined mental models that need to cover the models held by most of participants. Although the potential mental models could be forecasted by researchers based on their teaching experiences or through review of previous work, these forecasted mental models may not be able to cover all the models held by the current participants. In addition, the use of close-ended predicting activities increases the risk that a participant chooses an unwanted answer by accident, e.g. carelessness. However, the use of multiple questions for a single concept may solve this problem, because it is less likely that participants are repeatedly careless in the same way.

A collection of pre-defined mental models are not necessary if the predicting activities are open-ended. The mental models could be elicited by analyzing the verbal data collected from participants. Researchers do not have to spend time on forecasting potential mental models based on their teaching experience or literature review. As mentioned above, the mental models forecasted by researchers may not cover all the models held by current students. In this case, the close-ended predicting activities may miss the mental models that are held by current students but not on the pre-defined mental models list. In contrast, this problem does not exist in open-ended predicting activities. The other strength of the open-ended predicting activities compared to close-ended predicting activities is that it encourages students to expose more information on the details of their mental models.

When open-ended predicting activities are used, verbal data is required to identify participants' mental models. The potential risk of the open-ended predicting activities is that some participants may not be able to provide valid and accurate information about their mental models. In addition, it is time-consuming to analyze the verbal data collected from open-ended predicting activities compared to the quantitative data collected from close-ended predicting activities. Also, the use of qualitative data increases the difficulty of automating the data collection and analysis process

According to Jonassen (1995), mental models are relatively intangible, so multiple data sources are required to assess them. Gentner (2002) proposed that mental models could be elicited from people's verbal descriptions or inferred from the patterns of people's behaviors. The combination of open-ended predicting activities that could collect subjects' verbal descriptions and the close-ended predicting activities that could investigate the patterns of their behaviors should be able to achieve more valid information about mental models. As the comparison between the close-ended predicting activities and the open-ended predicting activities presented above shows, both kinds of predicting activities have strengths and weaknesses when used to investigate mental models, and fortunately they could each help address the weakness of the others. The open-ended predicting activities are able to identify the mental models that are not forecasted and are also capable of gathering the detailed information of mental models. However, some subjects may lack the ability to provide sufficient information about their thoughts. The close-ended predicting activities are still able to identify mental models when the holders of them do not provide sufficient verbal data. However, this kind of predicting activity will not expose the detailed information of a mental model and will only identify known mental models.

2.1.7.2 The *questionnaire* versus the *structured interview*

Two model elicitation techniques, *questionnaire* and *structured interview*, were used in the studies reviewed above. While the questionnaire technique was employed in Dehnaid & Bornat's (2006) study, Bayman & Mayer's (1983) study, and Kahney's (1983) study to collect information about participants' mental models, the structured interview technique was used in all the remaining studies.

The structured interview technique has many advantages over the questionnaire technique. Firstly, the structured interview gives instructors and researchers a chance to discuss a topic in-depth with interviewees and search more deeply for the information that they are interested in. For example, when an interviewee provides an interesting answer to the interviewer's question, the interviewer can ask further questions in order to explore the interviewee's in-depth thoughts on this topic. In contrast, questionnaires cannot support the in-depth conversations between the researcher and the respondent, even though some questionnaires simply ask participants to provide explanations of their answers. Secondly, structured interviews allow researchers to seek more accurate and clearer explanations from interviewees. As (Gentner, 2002) mentions, people often provide inaccurate and unclear explanation of their thoughts. In an interview, the researcher could ask the interviewees to further explain their thoughts until they are satisfied with the information provided.

However, a vital weakness of the structured interview technique causes it to be inpractical as the main mental model elicitation technique that could be used in a 'real' education context. As discussed before, instructors need to spend too much time to conduct a structure interview. They are often unable to afford the time required to interview students, especially when there is a big class. The questionnaire technique seems more practical to use in a 'real' education context than the interview techniques. Compared to the interview technique, the use of questionnaires would save instructors considerable time at the implementation and data collection stages of the mental models studies. Although the interview technique is not practical as the main mental model elicitation technique in an education context, it can still be used

as a additional support when it is really required, e.g. when the instructor wants to achieve an in-depth understanding of a mental model.

Apart from the questionnaire and interview techniques, there are actually many other mental model elicitation techniques such as stimulated recall[2] (Du, 2004) that have been used in the cognitive science and HCI (Human-Computer Interaction) domains Though few of these have been employed to elicit novice programmers' mental models of programming language and basic programming concepts. These mental model elicitation techniques require researchers to be highly involved and spend a large amount of time on the interaction with participants. These techniques are often used in a study that targets a small population of participants. They allow researchers to explore the details of individuals' mental models. However, instructors cannot afford to use these techniques to investigate the distribution of mental models in large classes.

3.2 Visualization Used in Programming Education

Visualization technology has been employed to aid programming learning for over 30 years, with numerous visualization systems developed specifically for this role. However, a key question still remains, namely, how does visualization improve learning? Many researchers (e.g. Ben-Ari, 2001) suggest that visualization works because of its ability to help people construct mental models of abstract phenomena, such as programming concepts and algorithms.

This section presents the current 'state of the art' of the visualization systems used within the programming education domain. Firstly, sub-section 3.2.1 describes the terminology associated with visualization systems. A survey of visualization systems specifically developed for programming education is then presented in sub-section 3.2.2. Finally, sub-section 3.2.3 describes empirical studies of the effectiveness of visualization when used within programming education.

[2] Stimulated recall is a technique that records participants' activities by notes or a video device and asks participants to recall and explain their behaviors afterwards (Du, 2004).

3.2.1 Terminology of Software Visualization Systems

There is a long history of research in the use of visualization to support software development and programming education. However, the terminology associated with the use of visualization in this domain was obscure in the early years (Hyrskykari, 1993). The terms *visual programming* and *program visualization* were employed in almost any situation where graphical elements appeared together with programs (Hyrskykari, 1993). Myers (1990) made the first attempt to give precise definitions to these terms.

Myers (1990) defined ***Visual Programming*** as "*any system that allows the user to specify a program in a two (or more) dimensional fashion*". Unlike conventional programs that consist of a one dimensional text-based stream, the programs in visual programming are built up with two (or more) dimensional, graphic-based elements. Visual programming allows programmers to construct a program without writing any code. It should be noted that the conventional programming languages that are used to define picture, or drawing, packages, e.g. AWT and SWING, do not belong to Visual Programming (Myers, 1990).

With ***Program Visualization***, the programs are still represented in a conventional, textual manner, but "*graphics are used to illustrate some aspect of the program or its run-time execution*" (Myers, 1990). Unlike visual programming, where the graphics are used to create programs, the graphics in Program Visualization are just employed to illustrate programs after they have been created in the conventional textual manner (Myers, 1990).

Myers (1990) proposed that Program Visualization systems can be categorized along two axes: whether they illustrate the ***code***, ***data*** or ***algorithm*** of the program, and whether they are ***dynamic*** or ***static*** (Figure 3-8). Depending on the first axis, program visualization systems can be divided into three categories, including *code* visualization, *data* visualization, and *algorithm* visualization. Code visualization

systems add graphical marks to textual program code, or covert textual code into a graphical form, e.g. a flowchart or UML diagrams. Data visualization systems employ graphical elements to represent the actual data used within a program. Algorithm visualization systems use graphics, often with animations, to illustrate computer algorithms. Unlike code visualization and data visualization, algorithm visualization functions at a higher abstract level to visualize how a program operates. As Myers (1990) explains, the graphical elements in algorithm visualization are not directly mapped to the data in a program and the evolvement of the pictures might not correspond to the execution of a program statement.

Based on the second axis, program visualization systems can be divided into two categories, including *static* visualization and *dynamic* visualization (Myers, 1990). A static visualization system can only show snapshots of a program at different points, whereas a dynamic visualization system can represent the changes from one state of the program to another, as the program executes. Animations have been employed in some dynamic visualization systems. However, the definition of animation was also obscure. According to Stasko & Patterson (1992), the term 'animation' was often misused in the past. Many systems claimed to have animation capabilities although they just simply highlighted lines of code, or changing colour of graphics, while statements were executed. Stasko & Patterson (1992) give a definitive explanation of animation as "*consists of the rapid sequential display of pictures or images, with the pictures changing gradually over time*". They explain that the conditions for an animation are: the image's changes from frame to frame have to be small enough; and the speed of displaying has to be fast enough (Stasko & Patterson, 1992).

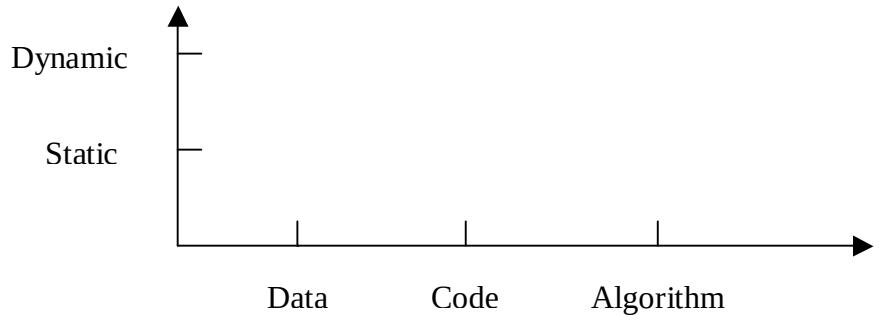


Figure 3-8: The taxonomy of program visualization system presented by Myers

3.2.2 Visualization Systems Developed for Programming Education

While a huge number of visualization systems were proposed to support professional programmers to develop and maintain software products in industry, the potential of visualization to support novices when learning programming has also been widely acknowledged. Over the last twenty years, many visual programming and program visualization systems have been developed to support programming education. This section presents a survey of these systems.

Based on Myers (1990)'s taxonomy, program visualization systems can be classified into six categories: *Static-Code*, *Dynamic-Code*, *Static-Algorithm*, *Dynamic-Algorithm*, *Static-Data*, and *Dynamic-Data*. A survey of program visualization systems that were developed specifically for education purposes shows that these systems fall into three categories: *Static-Code*, *Dynamic-Algorithm*, and *Dynamic-Data*. Along with visual programming systems, there are four categories of visualization systems available for supporting novice programmers (Table 3-6)

Visual Programming Systems		
RAPTOR (Carlisle et al., 2005); SFC, (Watts, 2004); BACCII++ (Calloni, 1997); FLINT (Henriksen & Kolling, 2004) SICAS (Marcelino et al., 2004)		
Program Visualization Systems		
Static-Code	***Dynamic-Algorithm***	***Dynamic-Data***

BlueJ (Kölling et al., 2003); JGRASP (Cross et al., 2002)	CABTO (Barnes & Kind 1987); DSV (Galles, 2006); DSN (Dittrich et al., 2001); IDSV (Jarc, 2005); LVJ (Hamer, 2004); VISA (Giannotti, 1987)	Bradman (Smith & Webb, 1999) Jeliot (Moreno et al., 2004) jGRASP (Cross et al., 2002) OOP-Anim (Esteves & Mendes, 2004) PlanAni (Sajaniemi & Kuittinen, 2003). Memview (Gries, 2005); ITEM/IP (Brusilovsky, 1993) VisMod (Jimenez-Peris, 1999) VIP (Virtanen, 2005), SDE (Romero et al., 2004)

Table 3-6: The visualization systems developed for educational purpose

3.2.2.1 Visual Programming Systems

A number of visual programming systems have been developed, aimed at improving a novice programmer's algorithmic problem solving ability and avoiding syntactic baggage while learning programming. This kind of system is often a flowchart-based development environment. One example of this type of system is RAPTOR (Figure 3-9), the Rapid Algorithmic Prototyping Tool for Ordered Reason (Carlisle et al., 2005). RAPTOR allows users to create a program visually by adding graphical symbols corresponding to loops, selections, procedure calls, assignments, inputs and outputs. The programs built by RAPTOR are forced to be structured. For example, selections and loops have to be appropriately nested, and each loop can only have a single exit. RAPTOR also provides a large number of pre-defined functions and procedures to simplify program development. When a program has been edited, it can then be compiled and executed by the RAPTOR environment. The user can run the program in a step-by-step mode or in a continuous play mode. When the program is executed, the location of the current executing symbol and the values of all variables are shown to the user.

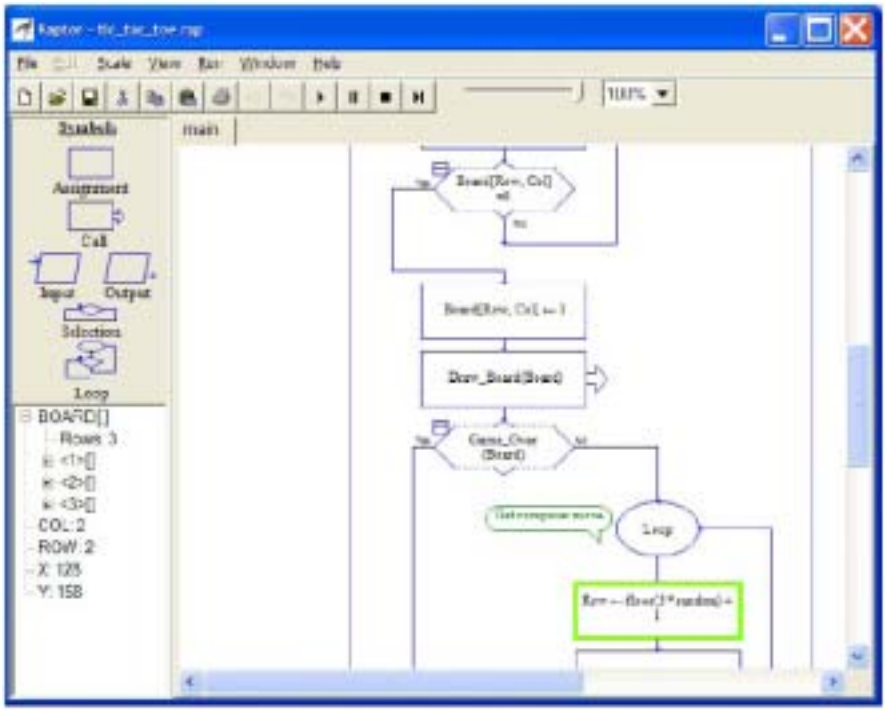


Figure 3-9: The RAPTOR development environment (Carlisle et al., 2005)

Apart from RAPTOR, there are several similar systems currently available to support novice programmers. The SFC, Structured Flow Chart editor (Watts, 2004), supports programmers when building flowcharts for structured programs, and simultaneously generates pseudo-code in a generic or C++ format. The BACCII++ (Calloni, 1997) system also supports the function of automatic code generation. Once a program is completed, the system can generate code for one of five languages, including Pascal and C++. The FLINT (Henriksen & Kolling, 2004) system allows programmers to first make a top-down decomposition of the program and then develop the sub-modules in flowchart mode. The SICAS (Marcelino et al., 2004) system further integrates an additional 'teacher mode' that enables teachers to set problems for their students to solve.

3.2.2.2 Program Visualization Systems

Static-Code Visualization Systems

An example of a pedagogical system that provides static-code visualization is BlueJ (Kölling et al., 2003). BlueJ is an integrated development environment, specifically developed for introductory programming education, focusing on teaching students

object-oriented programming concepts through the Java language. BlueJ has been accepted by many educators as a successful pedagogic environment and is being widely used in introductory programming courses all around the world.

BlueJ supports static visualization of a program structure as UML diagrams (Figure 3-10). In the diagrams, a box with a class name represents a class. The relations between classes are represented as lines: the 'use' relation is represented as a broken line and the 'inheritance' relation is represented as a solid line. The static visualization provided by BlueJ clearly shows the structure of an object-oriented program. It encourages novice programmers to view an object-oriented program at a higher level. However, Ragonis & Ben-Ari (2005) criticize BlueJ in that it fails to help students get the overall picture of program execution. They suggest that BlueJ has to be enhanced by a function showing dynamic visualisation of program execution. JGRASP (Cross et al., 2002) is similar to BlueJ in that it also supports UML class dependency diagrams to represent the overall topology of an object-oriented program.

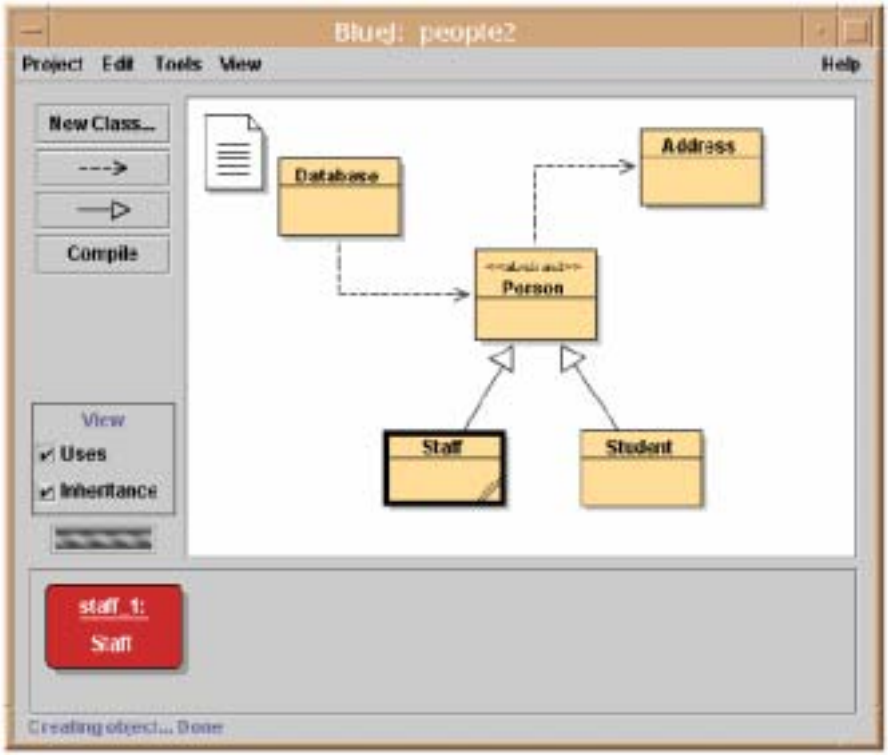


Figure 3-10: The BlueJ development environment

Dynamic-Algorithm Visualization Systems

Algorithm visualization systems can help programming students to construct and interact with visual representations of an algorithm. Undoubtedly, algorithms and data structures are crucial for program development. The classical equation 'Programs = Algorithms + Data Structures' reveals the importance for programmers to master algorithms and data structures. The in-depth understanding of algorithms is not only beneficial to the learning of programming, but also crucial for the understanding of other areas, such as database, graphics, and artificial intelligence (Stasko, & Hundhausen, 2004). However, it is not an easy task to learn algorithms. Most algorithms are complex, requiring many steps of operations, such as conditional, iterative and recursive, based on the manipulation of large and complicated collections of data (Stasko, & Hundhausen, 2004).

In order to help programming students understand algorithms, a large number of algorithm visualization systems have been developed, e.g. CABTO (Barnes & Kind 1987), DSV (Galles, 2006), DSN (Dittrich et al., 2001), LVJ (Hamer, 2004), IDSV (Jarc, 2005), and VISA (Giannotti, 1987). Shaffer et al. (2007) have developed a wiki with links to 350 currently available algorithm visualizations.

The current algorithm visualization systems cover a wide range of algorithms and data structures, covering the most important topics in undergraduate courses. Shaffer et al. (2007) investigated the distribution of algorithms and data structures that have been visualized (Figure 3-11 and Figure 3-12). The results show that most visualizations are focused on sorting algorithms, searching structures, linear structures, and graph algorithms. Actually, these topics are often the main contents in an undergraduate algorithm course.

Some simple observations reveal that most of the currently available visualization systems visualize algorithms using similar formats: data structures are represented using graphical elements with a specific layout and algorithms are visualized by dynamic illustrations of the changes of data over the life of the algorithm's execution. According to Stasko & Hundhausen (2004), algorithms are fundamentally sequences of operations upon data structures. The dynamic nature of algorithms decides the

dynamic nature of visualization. This *static-algorithm* visualization system appears to be incapable of representing an algorithm.

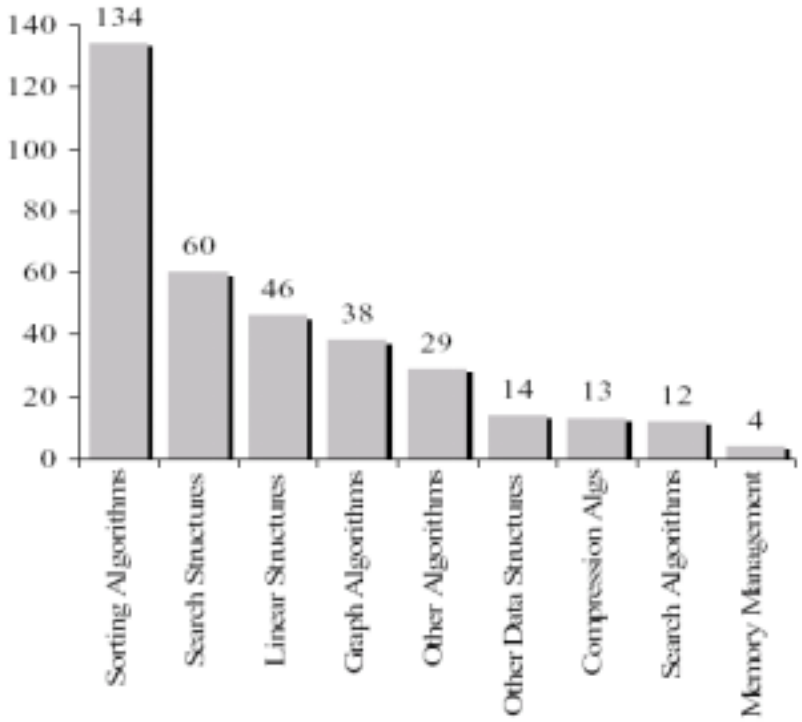


Figure 3-11: The distribution of algorithms and data structures that have been visualized (Shaffer et al., 2007)

Linear Structures	**46**
Lists	13
Stacks & Queues	32
Search Structures	**60**
Binary Search Trees	13
AVL Trees	7
Splay Trees	9
Red-Black Trees	8
B-Trees and variants	10
Skiplist	3
Other Data Structures	**14**
Heap/Priority Queue	8
Search Algorithms	**12**
Linear/Binary Search	2
Hashing	10
Sorting Algorithms	**134**
Sorting Overviews	6
Insertion Sort	18
Selection Sort	18
Quicksort	23
Mergesort	28
Heapsort	9
Radix Sort	10
Graph Algorithms	**38**
Traversals	10
Shortest Paths	11
Spanning Trees	9
Network Flow	3
Compression Algorithms	**13**
Huffman Coding	10
Memory Management	**4**
Other Algorithms	**28**
Computational Geometry	6
$\mathcal{NP}$-complete Problems	3
Algorithmic Techniques	7
String Matching	8
Mathematical Algorithms	4

Figure 3-12: Major categories for collected algorithm visualizations (Shaffer et al., 2007).

Dynamic-Data Visualization Systems

Dynamic-data visualization is capable of showing the changes of a program from one state to another, over the duration of the program's execution. Ben-Ari et al. (2002) argue that the details of program execution must be explicitly shown to students, or else students will try to construct a mental model of program execution by themselves, which may be inappropriate. A dynamic-data visualization system is one of the tools that is able to expose the details of program execution.

Name	Supported Programming Language	Support the visualization of OOP concepts	Animation Capability
Jeliot	Java	Yes	Yes
Memview	Java	Yes	No
jGRASP	Java	Yes	No
Bradman	C	No	No
OOP-Anim	Java	Yes	Yes
VIP	C++	No	No
PlanAni	Pascal	No	Yes
SDE	Java	Yes	No
ITEM/IP	Turingal	No	No
VisMod	Pascal	No	No

Table 3-7: The Dynamic-Data visualization systems

A large number of dynamic-data visualization systems (e.g. those shown in Table 3-7) have been developed specifically for supporting novice programmers. These systems have similar functions and generally work in a similar manner. Typically, in these systems (e.g. the Jeliot system shown in Figure 3-13), there is one or more 'graphical' windows showing the current state of the program. Generally, the classical 'box and arrow' diagrams are used in the 'graphical' windows, e.g. a box represents a variable and an arrow represents a pointer. The source code is often shown along with the 'graphical' windows. The statement currently being executed is highlighted and the visualization, represented in the 'graphical' window, evolves along with the execution of the current statement. It is argued that this approach, visually mapping a program statement with the behaviour of the statement, is able to help students construct a dynamic model of how the statement works. In addition, these systems typically support a step-by-step running model and some systems also support 'backward' execution.

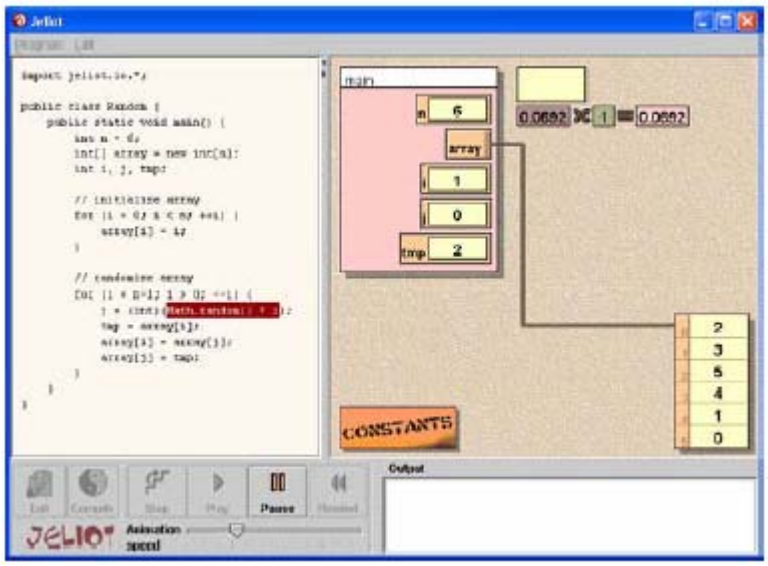


Figure 3-13: The Jeliot System (Moreno et al., 2004)

Some dynamic-data visualization systems, such as OOP-Anim (Esteves & Mendes, 2004), SDE (Romero et al., 2004), and Jeliot (Moreno et al., 2004), allow students to visualize the dynamic program execution of object-oriented programs that are organized in a very different way from procedural programs. Unlike the static-code visualization systems such as BlueJ that only provides static representation of the class dependency diagrams, those dynamic-data visualization systems visualize the dynamic processes of how an object is created, changed, disposed of and interacts with other objects. For example, the OOP-Anim system (Figure 3-14) not only provides class dependency diagrams, but also visualizes the states of object evolvement. As figure 3-14 shows, an object is represented as a box that contains two areas: the up area shows current values of the fields in the objects; and the low area shows the methods in the objects (the method will be highlighted when it is invocated). When the execution of a statement changes the state of an object, the corresponding object diagram will change simultaneously.

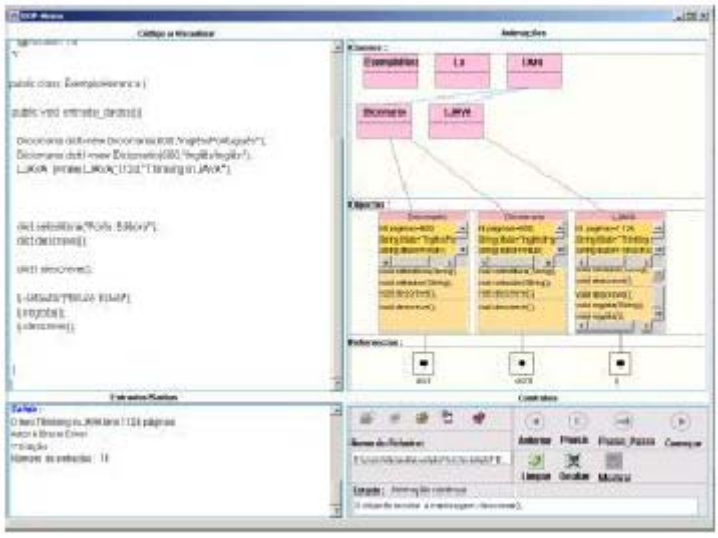


Figure 3-14: The OOP-Anim visualization system (Esteves & Mendes, 2004)

In addition, some of these systems, e.g. PlanAni (Sajaniemi & Kuittinen, 2003), OOP-Anim (Esteves & Mendes, 2004), and Jeliot (Moreno et al., 2004), use animations to simulate operations such as variable declaration, assignment, and object creation. For example, in the Jeliot system (Figure 3-13), the assignment process is represented as an animation where the symbol of a value (e.g. a number or a character) flies from a box which represents the 'source' variable to another box which represents the 'target' variable. In the OOP-Anim system, the object creation process is animated by highlighting the 'class' symbol with a red border and moving the square formed by the red border to the object's area and then changing it to a 'object' symbol.

Animation has been viewed as an important tool to help construct viable mental models of dynamic phenomena (e.g. Ben-Ari, 2001; Byrne et al., 1999; Moody et al., 1996). Some programming concepts such as assignment and object creation have dynamic attributes. Arguably, the use of static representation alone cannot represent a dynamic process and animation is a better choice. Animation is able to supply a vivid, explicit representation of how those dynamic concepts operate. It not only graphically represents the connections between the components of the model, but also depicts their dynamic transition from one state to another.

3.2.3 Empirical Studies on the Effectiveness of Visualization Systems in Programming Education

As described in section 3.1.7, Yehezkel et al. (2005) investigated 11 tenth-grade students' mental models of computer architecture before and after they used the EasyCPU visualization system. The results showed that many students failed to construct an appropriate mental model before using the visualization system, even though they had covered the theoretical knowledge with the traditional teaching materials. However, they successfully built an appropriate model after using the visualization system. This implied that the visualization played a critical role helping towards the construction of appropriate mental models.

George (2000) conducted a study to investigate the importance of graphical representations for supporting novice programmers' mental models of recursion. Kahney's Model Test (introduced in section 3.1.4) was used in this study as a tool to investigate the mental models held by participants. Kahney's Model Test is able to reveal whether the participant held a 'Copies' model (the appropriate model) or a 'Looping' model. The results showed that many participants who had previously demonstrated an understanding of the 'Copies' model (at that time they were facilitated by explicitly diagrammatic traces) failed to do so without diagrammatic traces. George (2000) claimed that mental models are unstable, and graphical representations can assist novice programmers to retrieve their mental models.

Moody et al. (1996) performed an experiment to study the importance of animation for the creation of mental models. In this experiment, the participants learned about and used a simplified computer operating environment, namely SPEAkS (Simplified Program Editing And Submission system), which supported users to create, save, and retrieve files (i.e. programs) and also allowed users to run programs, to view the output and to print program files and resulting output. There were two groups of participants: the first group received animated materials that explained the information communication between the various components in the system and the other group received non-animated materials. The results showed that the

participants with the animation interacted more effectively with the target system and demonstrated a better understanding of the system than those without animation.

Kasmarik, K. and Thurbon, J. (2003) carried out an empirical study of the value of diagrammatic representation as an aid to improve program comprehensibility at novice level. The results revealed that the use of diagrammatic representation could significantly improve the novice programmers' understanding of program code. Statistical analysis identified that a diagrammatic aid increased the correctness by 18.2% without affecting efficiency.

Ben-Bassat Levy et al. (2000) performed a long-term (a full year course) evaluation of the Jeliot 2000 visualization system. There were two groups: one group took the course that was facilitated by the Jeliot system; the other groups took another course that was receiving exactly the same teaching materials, but without the aid of the Jeliot system. The results showed that animation, in long term use, did not improve the performance of all the students. However, the animation group was found to use a better vocabulary of terms in their explanations and predictions than the group without Jeliot.

Although many empirical results show that visualization technology is capable of aiding the construction of mental models and of improving a student's understanding of programming concepts and algorithms, there were many empirical studies that did not identify any pedagogical benefits from visualization.

Stasko et al. (1993) conducted a study to investigate the effectiveness of visualization when teaching the 'pairing heap' algorithm to graduate students in Computer Science. The participants were divided into two groups: one group only used textual materials and the other group used the same textual materials supplemented by interactive animation. The results did not identify a significant difference between the two groups. The group with animation did not perform any better than the group with only textual materials.

Humdhausen et al. (2002) conducted a meta-analysis of twenty-four experimental evaluations of visualization technology. The results showed that the pedagogical benefits of visualization technology were not as marked as expected. Figure 3-15 shows, 10 out of 24 studies found no significant difference between the group using the visualization materials and the group using traditional materials. 2 studies found that the group using visualization materials performed better, but they could not determine whether or not the difference was from the use of visualization. 1 study achieved a 'negative' result in which the participants with visualization technology performed significantly worse than those with text-based tools. Only 11 studies (less than 50% of all studies) found that the use of visualization technology improved a student's performance.

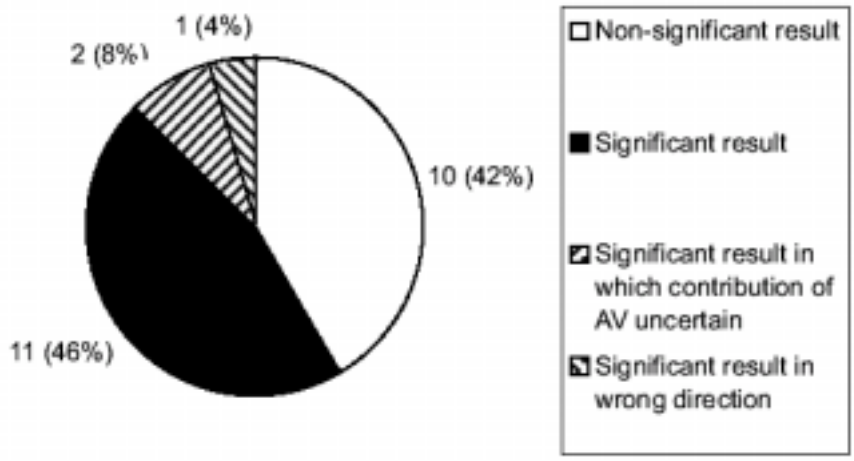


Figure 3-15: The results of the 24 experiment analyzed by Humdhausen et al. (2002)

Naps et al. (2003) attributed the poor pedagogical value of visualization technology to the low engagement of learners with the visualization. As they claimed, "*visualization technology, no matter how well it is designed, is of little educational value unless it engages learners in an active learning activity*". They further proposed that the more active engagement activities such as constructing a visualization by students themselves or presenting a visualization to an audience for feedback and discussion would help students achieve more pedagogical benefits than those passive activities such as only viewing the visualization.

In addition, Kohoe et al. (1999) found that it was more likely that students would achieve pedagogical benefits from animation in an open homework-style learning scenario, rather than in a closed exam-style learning scenario. This implies that students should not only be able to access the animation systems in school, but also outside school. They further speculated that animation is more pedagogically effective when it is used in coordination with other learning materials, or accompanying the instructors' explanation of the animation. In addition, Stasko et al. (1993) also proposed two conditions in which animation might be more beneficial: 1) animations need be accompanied by comprehensive, motivational instructions; 2) animation systems need to have 'rewind' or 'replay' functions.

3.3 Empirical Studies on the Effectiveness of the Cognitive Conflict Teaching Strategy

The role of cognitive conflict has been emphasised by many researchers (e.g. Posner et al, 1982) as a central condition for conceptual change and a large number of empirical studies have been conducted to investigate the pedagogical effectiveness of the cognitive conflict teaching strategy.

Many empirical studies have achieved positive results. The following presents several examples. Baser (2006) conducted a study to compare the effectiveness of cognitive conflict based physics instruction over traditional physics instruction for improving a student's understanding of heat and temperature concepts. 82 second grade pre-service teachers, 27 males and 55 females, took part in this study. They were divided into two classes: one class included 42 participants who received cognitive conflict based instruction (this class was named as the CCI group); and the other class included 40 participants who received traditional physics instruction (this class was named as the TPI group). Before the instruction, all the participants' prior conceptual understanding of heat and temperature concepts was assessed using a Thermal Concepts Evaluation (TCE) test that was specifically designed to investigate any misconceptions held by a student of heat and temperature. Participants then took a three-week course. In the class that received cognitive conflict based instruction,

the instructors demonstrated an anomalous situation to trigger cognitive conflict, or in some situations where an experiment was possible, participants conducted the experiment and observed the result to achieve cognitive conflict. Afterwards, the participants were asked to compare and discuss their prior ideas and what they just saw from instructor's demonstration or experiment results. Finally, the instructor explained the scientific ideas to the participants in detail. The results of the pre-test showed that there was no significant difference of the understanding of heat and temperature concepts between the CCI group and the TPI group. The results of post-test showed that the cognitive conflict based instruction was indeed more effective in improving the participants' understanding. Mean scores of the post-TCE test of the CCI group were significantly higher than that of the TCI group.

Demircioglu et al. (2005) carried out a study to investigate the effects of the cognitive conflict based learning approach within chemistry, on the topic of acids and bases. 88 participant tenth grade students, aged from 16 to 17 years old, took part in this study. They were divided into an experimental group and a control group. Both groups took a chemistry course consisting of five 45-minute sessions per week, including three lectures and two laboratories. However, the experimental group experienced the cognitive conflict based teaching approach while the control group experienced the traditional teaching approach. In the experimental group, students conducted chemical experiments that were designed to trigger conflicts between the students' existing misconceptions of acids and bases concepts and their observed results of the experiments. After the experiment the instructor encouraged and guided the students to compare and discuss their misconception and experiment results. Two instruments, the 'Concept Achievement Test' and the 'Chemistry Attitude Scale' were used before and after the study to investigate the students' understanding of acids and bases concepts and their attitudes towards chemistry. The results of the post-test showed that the students who experienced the cognitive conflict based teaching approach exhibited a significantly greater achievement than those who experienced the traditional teaching approach. In addition, there was a significant difference in the attitude of the students towards chemistry between these two groups.

The students who experienced the cognitive conflict based teaching approach showed a more positive attitude towards chemistry.

An earlier study was conducted by Nussbaum and Novick (1982) who proposed a cognitive conflict based teaching model that consisted of three stages: 1) create an 'exposing event' to drive students to invoke their existing conception and interpret it; 2) create a 'discrepant event' to trigger a conflict between a student's existing conception and the observed phenomenon or experiment results; 3) support the student to look for a solution to the conflict and encourage accommodation. This teaching model was used in a teaching unit to improve a student's understanding of the particle model of gases. Participants in this unit were school students in sixth to eighth grades. The data observed from the first two lessons revealed that this model helped students become aware of the problems in their own existing model, engaged them in a meaningful discussion of competing hypothetical models, and enhanced cognitive accommodation.

While a large number of empirical studies achieved positive results with a cognitive conflict teaching strategy, there were still many studies that did not support the effectiveness of this teaching strategy. For example, Hand & Treagust (1988) employed the cognitive conflict teaching strategy in order to enhance students' learning of the concepts associated with acids and bases. This study did not find any significant improvement. A student's preconceptions of acids and bases were collected through interviews held three months before the study commenced. Based on these preconceptions, a collection of experiments involving acids and bases were designed. Students then carried out these experiments and noted whether or not the experimental results were consistent with their preconception. The results obtained showed that the cognitive conflict teaching strategy was not successful in promoting conceptual change for all the misconceptions appearing in this study, although some students showed partial understanding of the concepts.

Researchers (e.g. Chinn and Brewer, 1998) found that students often failed to achieve meaningful conflict and hence did not become dissatisfied with their prior

concepts. Even when students experienced a conflict event when confronted with contradictory information, they often did not recognize the conflict between the contradictory information and their prior ideas. Students often present different responses to the conflict event and anomalous data. Chinn and Brewer (1998) conducted a study to investigate students' responses to anomalous data in science. They summarized eight possible responses from students:

- **Ignoring –** the student simply ignores the anomalous data.
- **Rejection –** the student rejects the data and does not believe that the data is valid. They might think the data is produced due to methodological flaws or other errors. Even when they deny the data, they still explain why the data is invalid.
- **Uncertainty-** the student is not sure of whether or not the data is valid.
- **Exclusion –** the student excludes the data from the domain of the current theory. The student may believe the data, or may not, but they think that it is not necessary to explain the data because it is irrelevant to the current theory.
- **Abeyance** – the student believes the data is valid, but they also believe that their theory should be able to explain the data.
- **Reinterpretation-** the student does not change their current theory to a new one, but rather reinterpret the data by using the current theory.
- **Peripheral Theory Change –** the student believes the anomalous data is true and agrees to change the current theory. However, this student just makes minor changes to the peripheral components of the theory, but keeps the core components in the theory.
- **Theory Change –** the student abandons their current belief and constructs a new theory instead.

For the students who gave one of the first six responses, they did not make any conceptual change. Although the students who gave the seventh response made a change to their prior conceptions, it is not a substantial conceptual change. Only the last response involves an effective conceptual change.

Based on the controversial results from empirical studies, some researchers (e.g. Zohar and Aharon-Kravetsky, 2005) argued that the evidence regarding the

effectiveness of the cognitive conflict teaching strategy was still inconclusive. In addition, some researchers (e.g. Limón 2001) went on to explore the reasons why some applications of the cognitive conflict teaching strategy failed. Limón (2001) proposed that many effects might affect the success of the cognitive conflict teaching strategy, such as motivational factors, social factors, students' prior knowledge, students' epistemological beliefs, students' values and attitudes, students' learning strategies and cognitive engagement, and students' reasoning abilities. Kang et al. (2005) carried out a study to investigate the influence of students' cognitive variables (including logical thinking ability, field dependence/independence (FDI) [3], meaningful learning approach), and motivational variables (failure tolerance, mastery of goal orientation, and self-efficacy) on cognitive conflict and conceptual change. The results show that FDI significantly affects the degree of cognitive conflict, while logical thinking ability, FDI, and failure tolerance significantly affect the conceptual changes.

Zohar and Aharon-Kravetsky (2005) also claims that the inconclusive findings regarding the effectiveness of the cognitive conflict teaching strategy can be explained by the difference in the academic levels of students. They conducted a study to compare the effectiveness of a cognitive conflict based teaching approach (ICC) and direct teaching approach (DT) for teaching the control of variables (COV) strategy to students of two academic levels (low level and high level). Teaching students the COV strategy could help them gain experience with experimental science: "*the understanding that in a multivariate system only one variable should be manipulated at a time while others are held constant*" (Moher & Wiley, 2004). 121 ninth-grade students, 67 were at a high academic level and 54 were at a low academic level, took part in this study and were divided into four groups in a 2*2 design with the academic level of students and teaching method as independent variables. The ICC teaching method started with students' independent investigation of the computerized simulation of a problem, such as a photosynthesis problem, to expose their pre-instructional thinking strategies. Students' initial thinking strategies were challenged by the conflict scenarios presented by teachers and by class

[3] FDI refers to cognitive ability of "*how successfully one can dis-embed relevant information from complex and potentially confusing contexts*" (Kang et al., 2005).

discussion. Then the teachers led the class discussions to help students construct appropriate COV rules. The DT teaching method followed a more traditional teaching model in which the teacher presents the COV rule, explains it and demonstrates how to use it, and students do not experience cognitive conflict. The results did not demonstrate significant difference between the ICC method and the DT method for teaching the COV strategy. However, there was indeed a significant interaction effect between the level of students and teaching methods. The findings show that the students with high academic level benefited from the ICC method and were hindered by the DT method, while the students with low academic level benefited from the DT method and were hindered by the ICC method. It reveals that the effectiveness of the cognitive conflict based teaching method may vary, depending on students' academic levels. Apart from Zohar and Aharon-Kravetsky (2005), Dreyfus et al. (1990) also identified the effects of students' academic levels on the effectiveness of cognitive conflict teaching strategy. They investigated two groups of junior high school students who were taught biological concepts using a cognitive conflict teaching strategy. One group of students had a successful academic history while the other had one of failure. The successful students performed with positive attitudes and reacted enthusiastically to cognitive conflict, while the unsuccessful students felt anxious, unsafe and threatened by cognitive conflict.

3.4 Summary

This chapter first presents six previous studies that investigated the mental models held by novice programmers. The research goal, method employed, and findings achieved for each study are described. In addition, the mental model elicitation methods used in these studies are specifically discussed from two dimensions: the elicitation activities and the elicitation techniques. The current state of the art of visualization used in programming education is then presented. This section first clarifies the terminology associated with visualization systems, and then describes visual programming systems, Static-Code program visualization systems, Dynamic-Algorithm program visualization systems, and Dynamic-Data program visualization systems that were specifically developed to support students when learning to program. Empirical studies of the effectiveness of visualization used in programming

education were then discussed. The last section of this chapter presented empirical studies on the effectiveness of the cognitive conflict strategy, including some possible reasons why cognitive conflict might not always succeed.

CHAPTER 4 – Investigating the Viability of Mental Models Held by Novice Programmers

It is important to study novice programmers' mental models. As Gentner (2002) pointed out: "*if typical incorrect models are understood, then instructors and designers can create materials that minimize the chances of triggering errors*". This chapter presents a study that aimed to investigate the mental models of basic programming concepts (focusing on assignment and reference) held by novice programmers and to study the relations between novice programmers' mental models and their performance in exam and programming tasks.

4.1 Introduction

This study was prompted by the recent, somewhat controversial, test conducted by Dehnadi and Bornat (2006), who explored the relationship between consistency of response and programming ability (This test is described in detail in chapter 3.1.1). In this study, Dehandi and Bornat's original test was extended to cover the concept of reference assignment as well as value assignment, and extended to capture qualitative data on participants' mental models. Although not directly comparable with the Dehnadi and Bornat study, this investigation revealed some parallels regarding consistency. However, of more interest was the variety of mental models of value and reference assignment held by participants. Many of these models were seen as nonviable, meaning that they could result in a flawed understanding of programming concepts. Perhaps unsurprisingly, it was found that students with viable mental models performed significantly better in the course examination and programming tasks than those with non-viable mental models. Of more interest is the fact that many students holding non-viable models of the more advanced concept of reference assignment still managed to perform well in the course assessments. Rather more disturbing is the discovery that some students still held non-viable models of the relatively simple concept of value assignment at the end of the course. The

problem of helping students to convert this diverse and persistent range of non-viable mental models into viable ones presents a real challenge to computer science educators.

4.2 Research Aims

Intrigued by the study of Dehnadi and Bornat, this work was to study the range and consistency of mental models held by students of both simple (value assignment) and more challenging (reference assignment) programming concepts towards the end of a first-year course. It also sought to elicit the range of models held by novice programmers and to investigate the relationship between mental models and programming tasks.

4.3 Research Method

4.3.1 Participants

Volunteers were recruited from students who were in the introductory programming course, Programming Foundations, in the University of Strathclyde. The programming language taught in this course is Java. BlueJ, the most popular educational programming IDE, was introduced as the programming tool in this course. The text book used in this course is “Objects First with Java - A Practical Introduction using BlueJ” (Barnes and Kölling, 2002). The object-oriented programming paradigm is taught in this course. Students start to learn object-oriented concepts at an early stage of the course. Object-oriented programming and Java have become the most popular programming paradigm and language in industry. Currently, more and more programming courses teach object-oriented programming and use Java as the programming language. As a result, this study chose the Programming Foundation course as a focus for the investigation.

A test was conducted at the end of the course when the participants had received about 20 lectures and 50 hours of tutorials and practical work. When the test was

conducted 124 students were in the course and 90 of them took part in this test. An analysis of student performance with in-course assessment revealed that 'weaker' students did not participate in this test.

In addition, the previous programming experiences of the participants were also investigated. The results showed that most participants had learned programming in high school although they had not taken a formal programming course before. Generally, they had been taught the procedural programming paradigm, rather than the object-oriented paradigm. The most commonly used programming languages were Comal and Basic.

4.3.2 Test Questionnaire

A questionnaire (appendix A-1) was distributed to the participants who were asked to complete it under exam conditions. This questionnaire was derived from Dehnadi and Bornat (2006) and was designed to investigate students' mental models of value assignment and reference assignment. For *value assignment*, a Java primitive type value is copied from the result of the evaluated expression on the right of the assignment operator to a variable on the left. For *reference assignment*, the address of an object is copied from the right and the copy is stored in a variable of reference type on the left. Note that Dehnadi and Bornat's test focused on value assignment.

The questionnaire contained two parts: the open-ended question part and the multi-choice questions part.

The open-ended question (figure 4-1) asked participants to describe the execution of a small program by using text or diagrams. The use of the open-ended, unstructured question was expected to reveal more unanticipated information on the participants' mental models.

Please trace the execution of the following statements. Describe what happens when each of the statements is executed. You may use both text and diagrams in your answer.

```
Person a, b;
a = new Person ("Jack");
b= new Person (“Tom”);
b = a;
```

Figure 4-1: The Open-ended question to investigate mental models

Model No.	Model Descriptions
Mr1	A copy of the object referenced by the variable on right-hand-side is given to the variable on left-hand-side
Mr2	The object referenced by the variable on right-hand-side is given to the variable on left-hand-side, and then the content of the original object on right-hand-side is erased (i.e. the object still exists, but no content is in the object)
Mr3	The object referenced by the variable on right-hand-side is given to the variable on left-hand-side, and then the variable on right-hand-side becomes null
Mr4	The object referenced by the variable on right-hand-side is also referenced by the variable on left-hand-side (**Appropriate mental model**)
Mr5	A copy of the object referenced by the variable on left-hand-side is given to the variable on right-hand-side
Mr6	The object referenced by the variable on left-hand-side is given to the variable on right-hand-side, and then the original object on left-hand-side is erased
Mr7	The object referenced by the variable on left-hand-side is given to the variable on right-hand-side, and then the variable on left-hand-side become null
Mr8	The object referenced by the variable on left-hand-side is also referenced by the variable on right-hand-side
Mr9	Nothing happens
Mr10	Variables swap values
Mr11	A test of equality

Table 4-1: Pre-defined mental models for reference assignment

The multi-choice questionnaire contained questions asking participants to predict the result of executing a small program from a collection of pre-defined answer options, each of which mapped to a possible mental model. The first part of the questionnaire used Dehnadi and Bornat’s questionnaire to study the participants’ mental models of value assignment. The second part, devised by the author, covered four questions to

study the participants' mental models of reference assignment. The programs used were similar to the one used in the open-ended question (figure 4-1). Again, the multi-choice answers were based on a full range of possible viable and non-viable mental models (table 4-1), which were devised based on teaching experience from introductory programming courses.

4.4 Results

This section presents the result of this test with the four sub-sections: 1) analyzing the participants' responses to the open-ended question; 2) analyzing the participants' responses to the multi-choice questions; 3) correlating the participants' mental models with their performance from in-course tests and final exam; 4) comparing participants' previous programming experiences with their mental models and their performance from in-course tests and final exam.

4.4.1 The Results of the Open-ended Question

25 out of the 90 participants provided too brief, or too unclear, answers to the question with the result that no valid information could be deduced from their description.

Of the remaining 65 participants, 11 were identified as using appropriate mental models of assignment and reference concepts when answering the open-ended question. Most of them were able to present a clear and proper explanation of type declarations, constructors, and other concepts. A participant even explained dereference and garbage collection. All the eleven participants used a consistently appropriate mental model when answering the multi-choice questions.

The remaining 54 participants were identified as holding at least one inappropriate mental model. These are categorized as follows.

4.4.1.1 Inappropriate mental models of assignment

Assign from the left to right

Six participants used the inappropriate model of assigning from the left to right. One of the six participants presented an interesting description of the execution of the statement *a=new Person ("Jack")* as '*the value a is assigned to the variable 'Jack' which has just been created*'. It seems the '*assign from the left to right'* model was solid in this participant's mind, and he obviously had an inappropriate mental model of object creation.

Inappropriate mental model: Compare operator

Four participants viewed '=' as a compare operator, rather than an assignment operator. They explained the execution of the statement *b=a* as follows:

"*b=a means object b is equivalent to object a*";

"*'Tom' (the string) is compared to 'Jack' (the string)*".

In addition, one participant ticked all the answer options in which the values in *b* and *a* are equal, e.g., '*a = 20 b=20*', '*a=10 b=10*'. They believed all the equal values would make the statement *b=a* work.

Inappropriate mental model: Inheritance & Instance of

A participant explained the assignment *b=a* as "*b is a subclass of a*", and drew a diagram as figure 4-2.

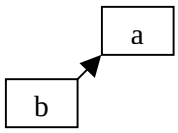


Figure 4-2: A diagram drawn by the student with the *Inheritance* model

Another participant explained the statement *b=a* as "*a instance of b*" so "*a=Tom, b=Tom*". This participant also provided an extra note to the multi-choice questions. The extra note for answering multi-choice questions provided by this participant reveals that they also used the 'instance of' model when answering the multi-choice questions.

It seems that the participants had very fuzzy mental models that are based on fragmental and improper understanding of basic programming concepts, such as assignment, object, and inheritance.

Other

A number of participants appeared to hold other inappropriate models but these were difficult to categorise from the descriptions e.g., "*b=a means both people are made equal to each other*". In addition, another participant used the following diagram (figure 4-3) to explain the execution of *b=a*.

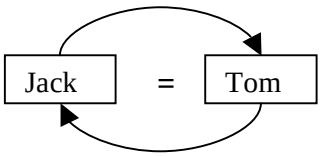


Figure 4-3: A diagram drawn by the student with an unknown mental model

4.4.1.2 Inappropriate mental model of reference and reference assignment

'Stored at' model

Seven participants were found holding a *Stored at* model, i.e., an object is 'stored at' a reference variable. For this model, a reference variable is viewed as a memory space where an object is stored. The participants with this model described the execution of the statement *a=new Person ("Jack")* as:

"*A person object with value 'Jack' is created and stored in a*";

"*The variable a now holds a new person object which has the value of 'Jack'*";

"*Sets a to store a new person object calling the person class constructor with a string 'Jack'*".

'Is' model

21 participants were found holding an *Is* model, i.e., the execution of assignment statement is just a process to name the object. For example, the execution of statement *a= new Person ("Jack")* is to name the created object as *a*. The

participants with this model described the statement *a= new Person ("Jack")* as "*a is a new object of type Person*".

It is obvious that the participants holding the 'Is' model lack an appropriate mental model of reference as well. In addition, they even ignore the variable concept. That means the participants with this model might not understand what a variable is and how it works. It highlights the question that: Is the knowledge on the memory mechanism of a computer program necessary for early programming learning, and should programming instructors teach the knowledge explicitly at an early stage of programming education?

'Assign' model

Four participants were found holding an *Assign* model[4]. For this model, an object is 'assigned' to a variable. In other words, the value or object is copied and the copy is 'sent' to the variable. This model is accordant to a classic conceptual model of 'assignment' used in programming teaching materials, especially in some animation-based program visualization tools. For example, when teaching students the execution of the statement *b=a*, an animation could show students that a copy of the value in variable *a* moves to variable *b*. the original value in *b* is overwritten, and the original value in *a* is unchanged. Four participants have been found holding this model. They described the statement *a = new Person ("Jack")* as "*create a new person object… then assigns this new person to variable a*".

'Set equals to' model

19 participants were found holding a *set equal to* model. The participants with this model described the statement *a = new Person ("Jack")* as "*A new person object is made … a is made equal to it*", and described the statement *b = a* as "*b is set to become equal to a*".

[4] The *Assign* model and the following *Set equal to* model can be viewed as appropriate if a student can understand that the address of an object rather than the object is 'assigned' or 'set equal to'.

'Component assignment' model

Four participants were found holding a *Component assignment* model, i.e., *b=a* means that some attributes of the object 'in' variable *a* replace the corresponding attributes of the object 'in' variable *b*. They described the execution of statement *b=a* as:

"*person b is given all the characteristics of person a*";

"*person b takes on all attributes of person a*";

"*instance b now contains variables in a*";

Or shown diagram graphically as:

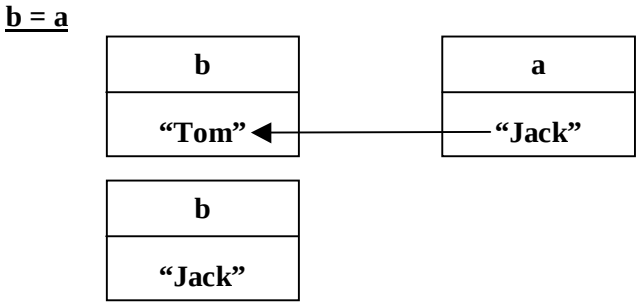


Figure 4-4: A diagram drawn by the student with component change mental model

4.4.1.3 Inappropriate mental models of other concepts:

Although this test focused on investigating novice programmers' mental models of assignment and reference concepts, some inappropriate mental models of other concepts such as variable, class, and object were also identified.

Inappropriate mental model of 'Variable' concepts

Apart from the 'Is' model mentioned above in which a variable is viewed as the name of an object, participants may have other inappropriate mental models of the variable concept. For example, one participant drew the diagram shown in figure 4-5, and appeared to believe that a variable can hold more than one value. Another participant described the execution of the statement *a=new Person ("Jack")* as "*Jack is stored at position a*". Further, one participant also described variable declaration as "*sets up two parameters a and b of type Person*". It is difficult to say what this participant was thinking when answering the question, but it is obvious that this

participant did not have a proper understanding of the variable and parameter concepts.

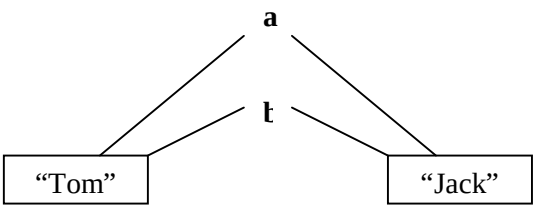


Figure 4-5: A diagram drawn by the student with an inappropriate mental model of the variable concept

Reference variable declaration

4 participants believed an object would be created automatically when a reference variable was declared. They thought two instances of the Person class were created when the statement “*Person a, b*” was executed. The statement “*a=new Person (“Jack”)*” was just to fill new contents to the objects. This misconception may come from their learning of the primitive type. In Java, an instance variable of primitive type is automatically assigned with a default value after this variable is declared. Students would not encounter problems when they did not explicitly give an initial value to an instance variable. Hence they may extend this to the case of a reference variable, and believe that a new object was created automatically when a reference variable was declared. This might explain why novices often omit the constructor in their programs and struggle with the *null pointer* error.

‘Class and Object’ concept

Inappropriate mental models of the class and object concept were also identified. For example, one participant drew the following diagram (see Figure 4-6). It is possible that they had an inappropriate mental model of the relationship between class and object.

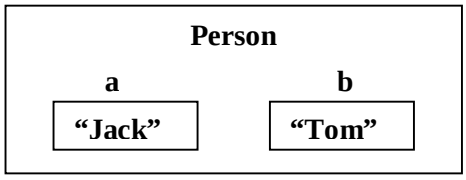


Figure 4-6: A diagram drawn by the student with an inappropriate mental model of class and object

In addition, two participants described the statement *a=new Person ("Jack")* as "*a new person contain jack*" and "*value of a set to 'Jack'*". Further, one participant described the statement *b = a* as "*Don't think it would work because Tom effectively becomes Jack, which can't happen*". They might also have an inappropriate model for an object.

4.4.2 The Results of the Multi-choice Questions

Following Dehnadi and Bornat, participants are categorized as consistent or inconsistent (no one was found in the blank group in our test). The consistent group can be further divided into a consistently appropriate group and a consistently inappropriate group. Furthermore, it is useful to compare the participants who held viable mental models (those in the consistently appropriate group) with those who held non-viable mental models (those in the consistently inappropriate group and the inconsistent group).

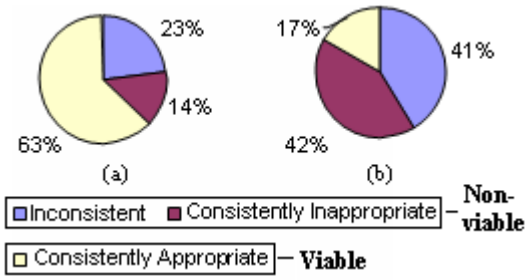


Figure 4-7: The percentage of each group separated based on mental models of value assignment (a) and reference assignment (b)

As figure 4-7a shows, the test revealed that 69 (77%) of participants used consistent (or almost consistent)[5] mental models when answering the **value assignment** questions. The remaining 21 (23%) participants used inconsistent mental models. In the consistent group, 56 (63% of all participants) participants used consistently appropriate mental models, while the other 13 (14% of all participants) used consistently inappropriate mental models, covering: 1) *Value copied from left to right*

[5] The participants who used a consistent model to answer 10 or more questions (total 12 questions) are put into the consistent group. This is to minimize the interference of carelessness.

(a => b; a unchanged); 2) *Nothing happens (a, b unchanged)*; 3) *'=' as a test of equality*; 4) *statements are executed simultaneously*.

The test found that 36 (40%) participants used consistent mental models when answering the **reference assignment** questions (the consistent group), while the remaining 54 (60%) participants were found using inconsistent mental models (the inconsistent group).

An interesting phenomenon was found in the inconsistent group, namely, 17 participants (nearly 20% of all participants) used a consistent mental model when answering the last three out of four questions, but used another mental model when answering the first question. In the program used in the first question, '*Person a, b; a = new Person ("Jack"); b = a;*', *b* does not refer to any object before the execution of the statement *b=a*. It seems unreasonable to put these participants into the inconsistent group because they only used a different mental model to answer this one question, which is significantly distinguished with the remaining questions.

When placing these 17 participants into the consistent group, 53 (59% of all participants) participants used consistent mental models, and 37 (41% of all participants) used inconsistent mental models. In the consistent group, 15 (17%) participants have consistently appropriate mental model, while 38 (42%) participants have consistently inappropriate mental models, covering: 1) *Object copied from left to right (a => b; a unchanged)*; 2) *Object copied from left to right (a => b; the content of object in a is erased)*; 3) *object copied from right to left (a <= b; b unchanged)*; 4) *Nothing happens (a, b unchanged)*. Figure 4-7b shows the percentage of participants in each group.

4.4.3 Comparison with the Assessment Results

The participants took part in four in-course assessments throughout the introductory programming course and one final examination at the end of the course. In the in-course assessments, participants were asked to complete a small program that was

evaluated on a 5-point scale. The mean of the marks for the four assessments for each participant was calculated and used in the data analysis. The final exam was paper-based, and evaluated on a 100-point scale (one participant was absent from the final exam).

To form a comparison with Dehnadi and Bornat's result, the inconsistent group and the consistent group were compared based on the participants' mental models of value assignment[6]. The results obtained match Dehnadi and Bornat's with the consistent group indeed performing significantly better than the inconsistent group in both the final exam (p^7 =0.027) and the in-course assessments (p = 0.024). However, the separation is not clean. 14 out of 21 (67%) participants in the inconsistent group scored 50% or above in the final exam, while 9 out of 68 (13%) participants in the consistent group scored below 50%.

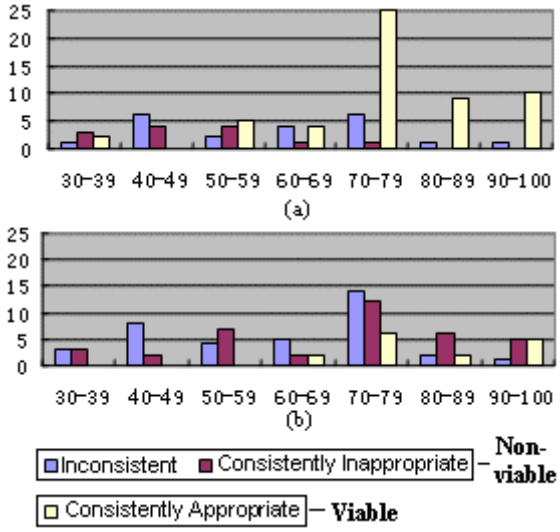


Figure 4-8: The results of the final exam for the participants in the inconsistent group, consistently inappropriate group, and consistently appropriate group based on value assignment (a) and reference assignment (b)

In this test the consistent group can be further divided into a consistently appropriate group and a consistently inappropriate group. The participants were first separated

[6] We must be cautious that Dehnadi and Bornat's test was conducted before the programming course and at week 3 when students had just learned the assignment concept, but this test was conducted at the end of a 20 week programming course.

[7] The Wilcoxon Mann-Whitney Test was used to compare the consistent group and the inconsistent group. The Wilcoxon Mann-Whitney Test can be used as the data was not normally distributed.

based on their mental models of **value assignment** (figure 4-8a). The results reveal that the consistently appropriate group performed significantly better in the final exam than the consistently inappropriate group ($p<0.001$) or the inconsistent group ($p<0.001$). In addition, the consistently inappropriate group was found to be significantly worse than the inconsistent group ($p=0.034$). This is at odds with Dehnadi and Bornat's result. For the in-course assessments, the consistently appropriate group was also found to perform better than the inconsistent group ($p = 0.049$) and the consistently inappropriate group ($p=0.070$)[8]. No statistical difference was found between consistently inappropriate group and the inconsistent group ($p = 0.883$).

The consistent group can also be divided into the consistently appropriate group and the consistently inappropriate group based on their mental models of **reference assignment** (figure 4-8b). Similar to the situation for value assignment, the results show that the consistently appropriate group performed significantly better in the final exam than the consistently inappropriate group ($p=0.011$) and the inconsistent group ($p<0.001$). For the in-course assessments, the consistently appropriate group also performed significantly better than the inconsistent group ($p<0.001$) and the consistently inappropriate group ($p=0.005$). In addition, and in contrast to the results for value assignment, the consistently inappropriate group would appear to perform better than the inconsistent group ($p = 0.079$ for in-course assessments; $p=0.071$ for the final exam).

Comparing figure 4-8a and figure 4-8b it can be seen that the majority of participants (42 out 56) who were in the consistently appropriate group for value assignment changed to the consistently inappropriate group (30 participants) or inconsistent group (12 participants) for reference assignment.

4.4.4 Comparison with Previous Programming Experiences

The results also revealed that participants with previous programming experience

[8] Although it is not statistically significant, but still strong.

performed significantly better than participants without previous programming experience in the in-course assessments ($p = 0.023$) but not in the final examination ($p=0.68$). In addition, there was a slightly higher percentage of experienced participants tending to hold a consistent mental model compared to those without previous experience.

4.5 Discussions

This study has identified a collection of non-viable mental models of assignment and reference concepts held by novice programmers. The quantitative analysis revealed that, at the completion of the first year course, one third of students still held non-viable mental models of value assignment, with only 17% of students holding viable mental models of reference assignment. This result is of significant concern. Both assignment and reference are key concepts in object-oriented programming. The high failure rates in programming courses are not surprising if students still do not understand these basic programming concepts at the end of courses.

This study also found that many participants held viable mental models of value assignment but non-viable models of reference assignment. This result is not surprising given that the concept of reference assignment is much more complex than the concept of value assignment.

In addition, the results also show that the students with viable mental models performed significantly better in the course exam and programming tasks than those with non-viable mental models. This reveals how important it is that novice programmers develop appropriate mental models of the key programming concepts.

However, the sheer diversity of mental models held for the reference assignment concept, and the different results for value assignment and reference assignment suggest that mental models may take some time to form, or that students may hold partially functional models which may work in certain circumstances (and which they consequently may be reluctant to let go of). Handling this diversity within a

cohort and helping students to develop viable models is a real challenge for computer science educators.

Finally, it is worth observing that the results appear to support those obtained by Dehnadi and Bornat in that the consistent group performed significantly better than the inconsistent group. However, the results also show that the separation is not clean, particularly so in the case of the more advanced concept of reference assignment. Many participants in the inconsistent group still passed the examination. Actually, Dehnadi and Bornat's test did not produce a clean separation either. In their test, 8 out 34 (24%) participants in the inconsistent group still passed the end-of-course exam while 6 out 21 (22%) of those in the consistent group still failed. This would indicate that the current evidence is insufficient to suggest that it is 'useless' to teach the inconsistent group and 'pointless' to teach the consistent group as they claim.

It should be noted that there were some potential weaknesses in this experiment. While the use of multi-choice questions allowed us to carry out quantitative analysis of the data related to students' mental models, it might produce some inaccurate results. For example, an answer option could map to two or more mental models. It is difficult to identify which mental model the participant is holding when they choose this answer option. In addition, the participants might choose an unwanted answer option due to carelessness. In this study, multiple questions were employed to minimize the effects of carelessness. Furthermore, the multi-choice questions can only identify the mental models that were pre-defined by the instructors based on their teaching experience.

4.6 Summary

This chapter describes an investigation into mental models of basic programming concepts held by novice programmers. In this study, mental models were captured using a multiple choice questionnaire and by getting students to describe their understanding using text and diagrams. These results were then used to compare groups based on viable and non-viable models against performance in exams and

programming tasks.

As a result, this study identified a collection of non-viable mental models of basic programming concepts (focusing on assignment and reference) held by first year programming students. The quantitative analysis revealed that at the completion of the first year course one third of students still held non-viable mental models of value assignment, with only 17% of students holding viable mental models of reference assignment. In addition, it was found that the students with viable mental models performed significantly better than those with non-viable mental models.

The study reveals the importance of helping students develop viable mental models. However, the traditional teaching approach, based on the theory of objectivism, does not do enough to challenge pre-existing ideas and to help students create viable mental models. Instead, it is proposed that teaching strategies based on the theory of constructivism, integrating cognitive conflict with visualization, be used to help students create viable mental models and to challenge and help repair non-viable models. The next chapter presents a constructivist-based learning model that integrates a cognitive conflict strategy and visualization.

CHAPTER 5 – A Teaching Model Integrating a Cognitive Conflict Strategy and Program Visualization

The investigation described in Chapter 4 revealed the importance of improving novice programmers' mental models of basic programming concepts. This chapter proposes a constructivist-based learning model that integrates a cognitive conflict strategy and program visualization. The first section of this chapter discusses the synerginistic benefits of uniting these two strategies when striving to improve a novice programmer's mental models of fundamental programming concepts. The second section describes a computer-supported learning tool developed by the author to support the proposed teaching model.

5.1 Teaching Model

The participants in the mental model test (chapter 4) have learned to program under a traditional learning model and using traditional, static learning materials for one academic year when the test was carried out. However, they still held non-viable mental models for basic programming concepts. It implies that the traditional learning model along with the traditional learning materials may be inefficient when used to improve students' mental models of programming concepts.

Compared to the construction of mental models for physical devices, it is more difficult to built viable mental models of programming concepts. While physical devices are visible and tangible, it is relatively easy for users to create a mental model of how the devices operate. On the other hand, programming concepts are invisible and untouchable. Students cannot 'see' what is happening 'in' the computer when a program is executed. It follows that students often misuse their previous knowledge or adopt intuitive models to understand program execution. Program visualization provides a potential solution to address this problem. This technique is

capable of simulating how a programming concept operates by using a graphical representation and animation, providing students with a concrete model of the programming concept.

As mentioned in Chapter 3, a large number of visualization tools are currently available to support students while learning to program. However, they have not been as successful as hoped, even though many researchers have strived hard to improve their effectiveness through the use of facilities such as interactivity and 3D techniques. One possible cause of this failure is that the educational effectiveness of visualization does not merely rely on the quality of the visualization tools themselves, but also on the way they are employed.

In the programming education domain, most visualization tools have been used from a traditional, objectivist perspective. Objectivism emphasizes that instruction is to transfer the objective knowledge into the learners' head (Vrasidas, 2000). Learning is actually a passive process of accumulating knowledge, and a learner is actually a passive 'receiver' of objective knowledge. Objectivism ignores a student's pre-existing knowledge. The instructors who hold the belief of objectivism, design teaching approaches and materials without consideration of students' pre-existing knowledge and believe students can naturally accept the knowledge. However, the performance of the participants in the mental model test reveals that many students are often unable to accept knowledge as expected. One possible reason is that students' pre-existing knowledge does indeed affect students when learning new knowledge. If the learning materials are not 'compatible' with students' pre-existing knowledge, they often do not work. This assumption is supported by constructivist and is actually being widely accepted in the education domain. Constructivism claims that students actively construct knowledge by combining the experiential world with existing cognitive structures (Ben-Ari, 2001). A student's prior knowledge plays a key role in learning new knowledge.

Many educators (e.g. Ben-Ari, 2001) claim that the design of learning materials has to take into consideration a student's pre-existing concepts and ideas. Any learning

materials, including visualization-based materials, and no matter how well they are designed, will not be as educationally beneficial as expected when they are designed without prior consideration of a student's pre-existing concepts and ideas. However, current program visualization tools and other computer simulation systems often neglect this prior knowledge. As Li et al. (2006) argued, "*while computer simulations are often used to offer opportunities for students to explore scientific models, they do not give them the space to explore their own conceptions, and thus cannot effectively address the challenge of changing students' alternative conceptions*"

Research (e.g. Baser, 2006) from the conceptual change domain suggests that a student's pre-existing concepts and ideas often conflict with the scientific ones. This is especially true in the programming domain where, due to the invisibility of program execution, programming students often misuse their prior knowledge, or adopt intuitive models, to understand programming concepts (Ben-Ari, 2001). When students are satisfied with their pre-existing concepts, they tend not to accept the new, scientific ones (Nussbaum and Novick, 1982). In this case, as proposed by Posner et al (1982), a precondition for students to accept new concepts is to drive them to feel unsatisfied with their pre-existing concepts. A potential way to do this is to use the cognitive conflict teaching strategy, a strategy that explicitly challenges a student's existing ideas in order to drive them to recognise problems in their understanding and motivate them to construct a scientific understanding.

However, it should be noted that the cognitive conflict teaching strategy alone is unlikely to be sufficient to achieve a change from non-viable models and students must be supported when creating new, viable models. This is not an easy task, especially for programming students, where programming concepts are invisible and untouchable. The traditional, static teaching materials are often not effective for students attempting to construct viable models. Instead, program visualization along with an animation technique would provide more powerful support. It is therefore proposed that a potential way forward is to adopt this approach, employing cognitive conflict to help students realise that there is a potential problem with their current

understanding, and to using a visualization-oriented learning environment to support them in constructing viable mental models.

A teaching model adopting this approach and comprising four stages is shown in figure 5-1:

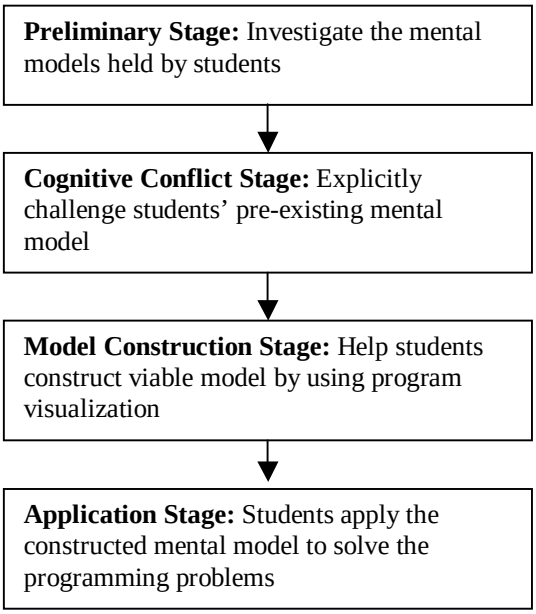


Figure 5-1: The proposed learning model to improve mental models

Preliminary Stage - Instructors investigate the pre-existing mental models held by programming students and identify typical inappropriate models. This stage plays a key role in the teaching model. The learning materials and activities would be designed based on the understanding of students' inappropriate models.

Cognitive Conflict Stage - Trigger a discrepant event to explicitly challenge students' pre-existing mental models and push students into cognitive conflict status. In a programming context, a potential and practical way to trigger a discrepant event is to ask students to predict the execution of a fragment of program that is designed based on the understanding of students' inappropriate mental models and provide students with an immediate response about whether or not their understanding is appropriate.

Model Construction Stage - Help students construct viable mental models by using visualization along with an animation technique. The visualization-based materials have to be able to cover the inappropriate mental models held by most of students.

Application Stage - Students go on to solve a programming problem by using the constructed mental model. This stage would strengthen the new model built by students.

A single version of a 'cognitive conflict' question or visualization-based material will often not be able to cover the inappropriate mental models held by all students. Hence multiple versions might be required to be designed and used to suit different categories.

5.2 A Computer-Supported Learning Tool Based on the Proposed Learning Model

To facilitate this teaching model a computer-supported, web-based learning tool was developed integrating a cognitive conflict strategy and visualization technique. This tool first presents students with a small fragment of program (figure 5-2). Students are then asked to predict the result of its execution. This fragment of program is designed to cover a collection of inappropriate mental models previously identified by instructors at the preliminary stage. The intention being that students who hold one of those inappropriate mental models would make an incorrect prediction. These students would then receive an immediate response that tells them their prediction is incorrect (figure 5-3).

Figure 5-2: The task to trigger cognitive conflict

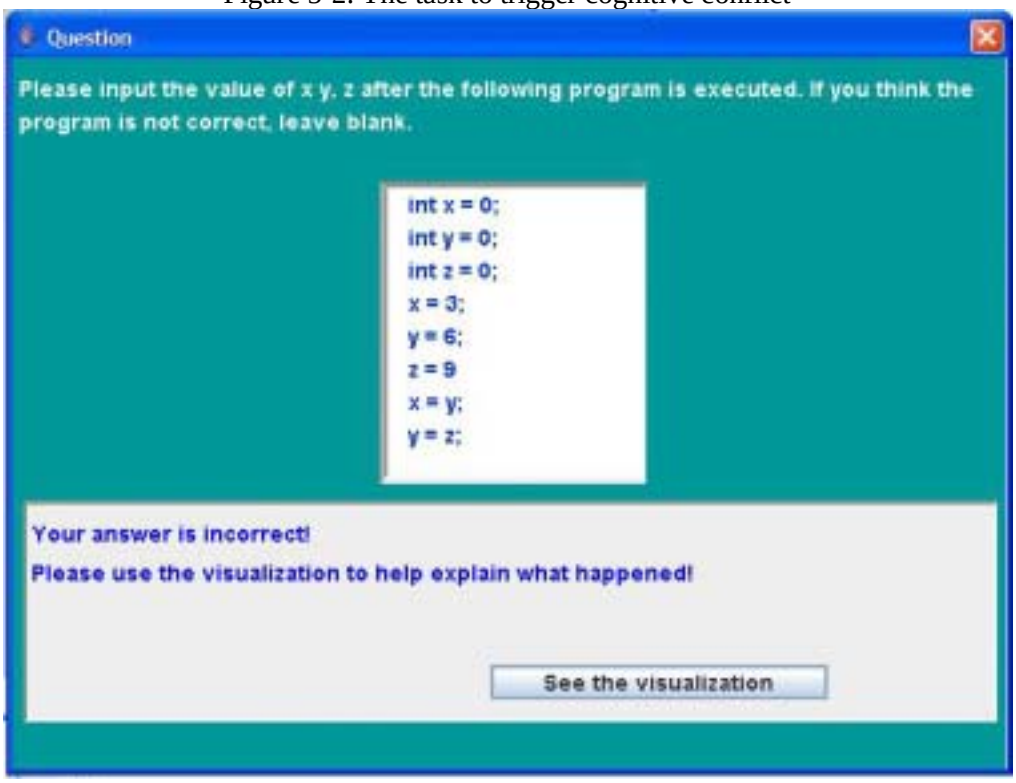


Figure 5-3: The result of the student's prediction

After this event, students who are now aware that their prediction was incorrect would be asked to use the visualization tool (figure 5-4) to simulate the dynamic

execution of a program fragment that is similar to the one used to trigger cognitive conflict. The visualization tool allows students to execute the program step by step. The 'step through' function is important for the visualization tool. With this function students have the control of the execution pace and able to 'see' what happened when a single statement is executed. When each statement is executed, the dynamic execution process is visualized using graphical representations and an animation. As mentioned in chapter 3, animation is an important tool to help students to construct viable mental models of dynamic phenomena such as programming concepts. So this visualization tool employs animation to reveal the dynamic process of how a concept works. As Stasko et al. (1993) suggested, animation might be more beneficial under two conditions. Firstly, animation needs to be used in coordination with an additional explanation of the animation. Therefore, this tool provides students with textual explanation of the animation for each step of the program execution (see bottom pane of the window in figure 5-4). In addition, Stasko et al. (1993) suggested that animation systems need to have 'rewind' or 'replay' functions. This is provided by the left pointing arrow button in the animation tool (figure 5-4). This tool currently supports value assignment (figure 5-4) and reference assignment concepts (figure 5-5).

The tool is easy to use. Students can use the right pointing arrow button to move the execution flow to the next statement or use the left pointing arrow button to move the execution backward. In addition, students are also allowed to edit the value of the variables. However, at present this tool does not provide the functions to visualize students' customized program.

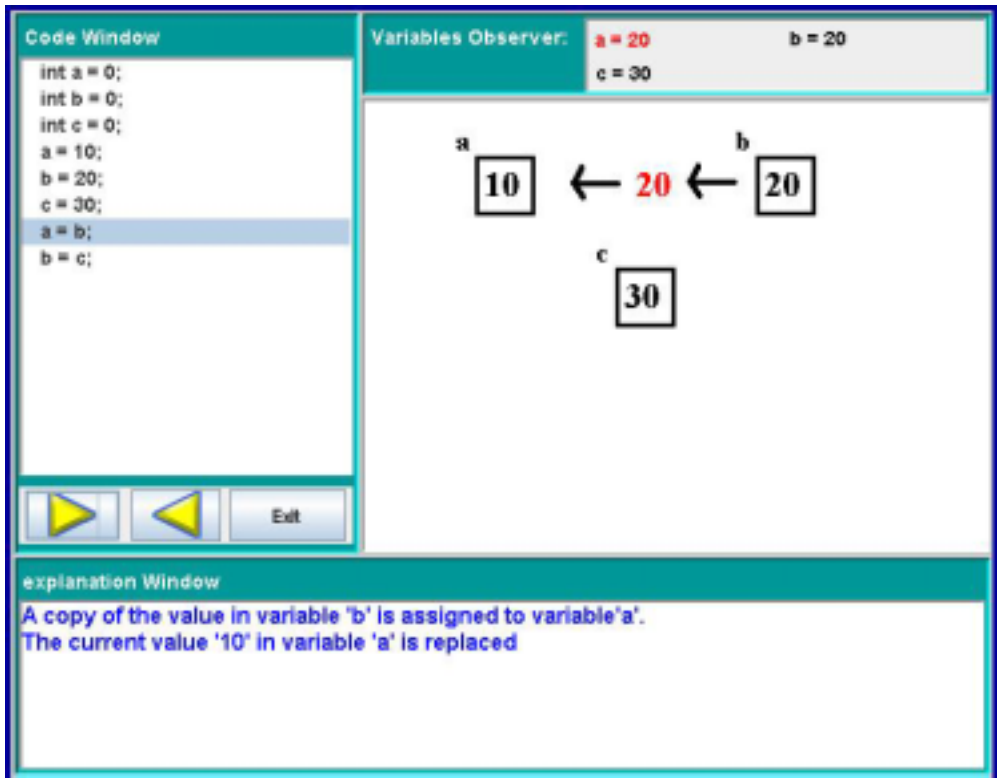


Figure 5-4: The interface of the visualization tool (value assignment)

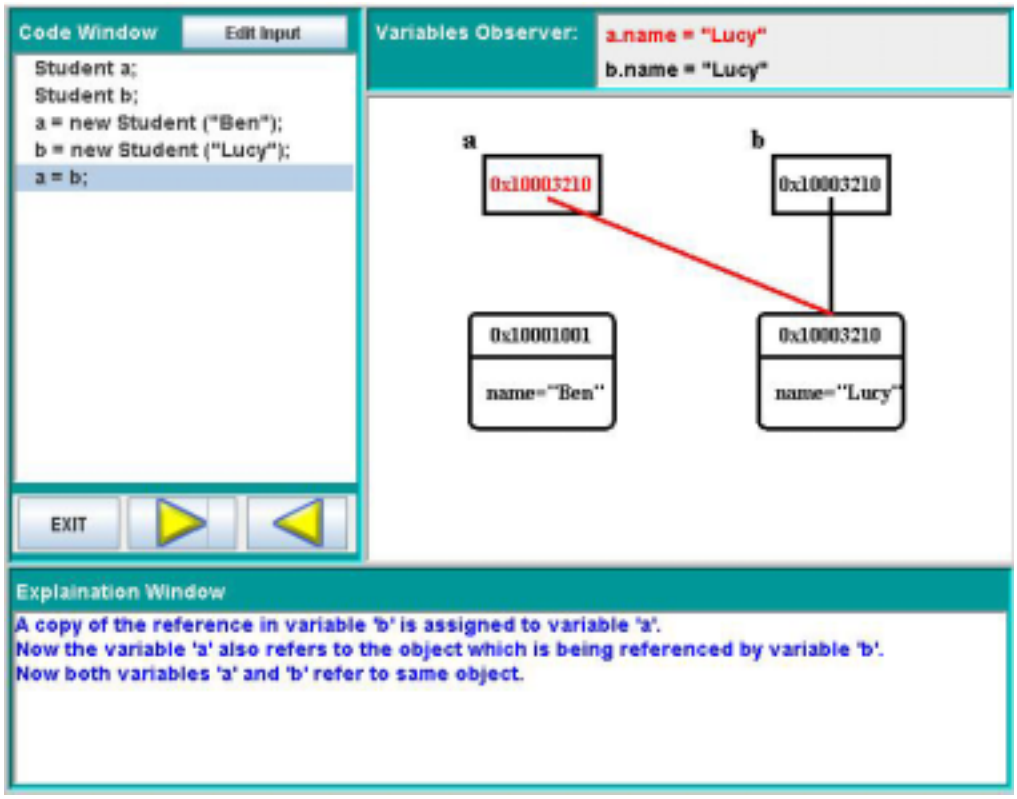


Figure 5-5: The interface of the visualization tool (reference assignment)

The following is an example of how the tool visualizes the execution of reference assignment. Firstly, when a reference variable is declared, e.g. '*Student b;*', a square with the name of the reference variable is drawn. At this stage, this variable does not refer to any object (figure 5-6). When an object is created and the reference of this object is assigned to a reference variable, e.g. '*b=new Student("Lucy")*', a square that represents this object 'flies' onto the canvas, and then the address[9] of this object 'flies' into the square that represents the reference variable (figure 5-7). When a reference is assigned from a reference variable to another reference variable, e.g. '*a=b*', the address of this object 'flies' from the right hand side variable into the left hand side variable (figure 5-8).

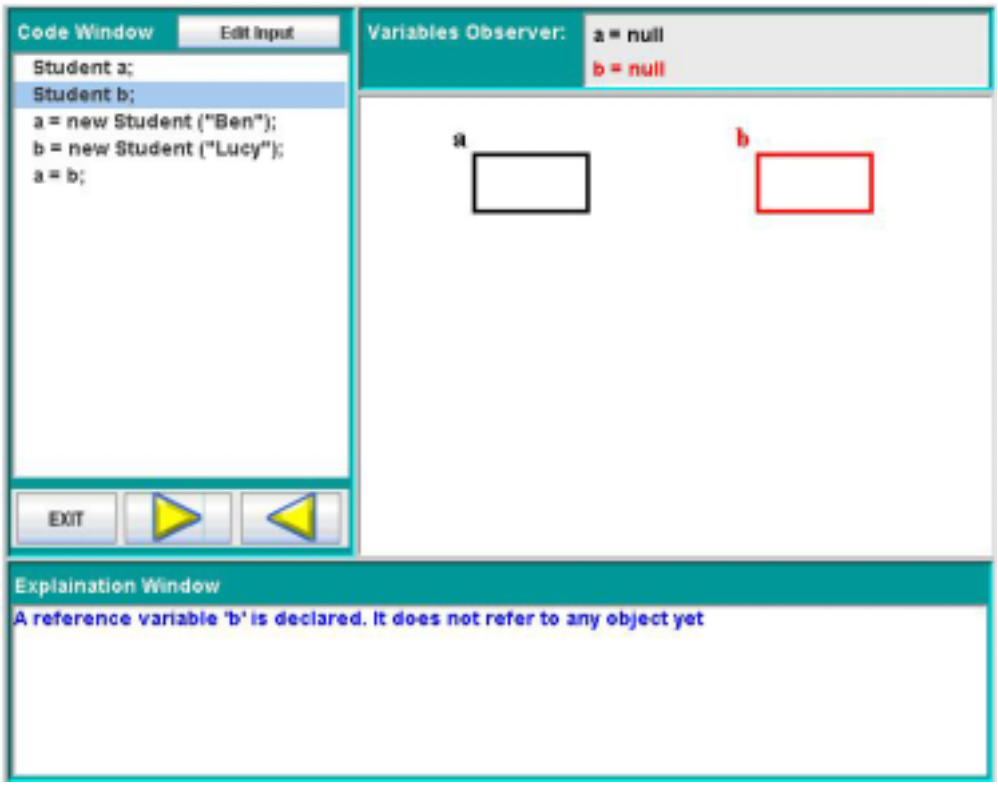


Figure 5-6: Declare a reference variable

[9] The actually implementation of 'reference' in the Java Virtual Machine is more complex than the way described in this tool. It is pointless to teach novice programmers this complexity at this stage.

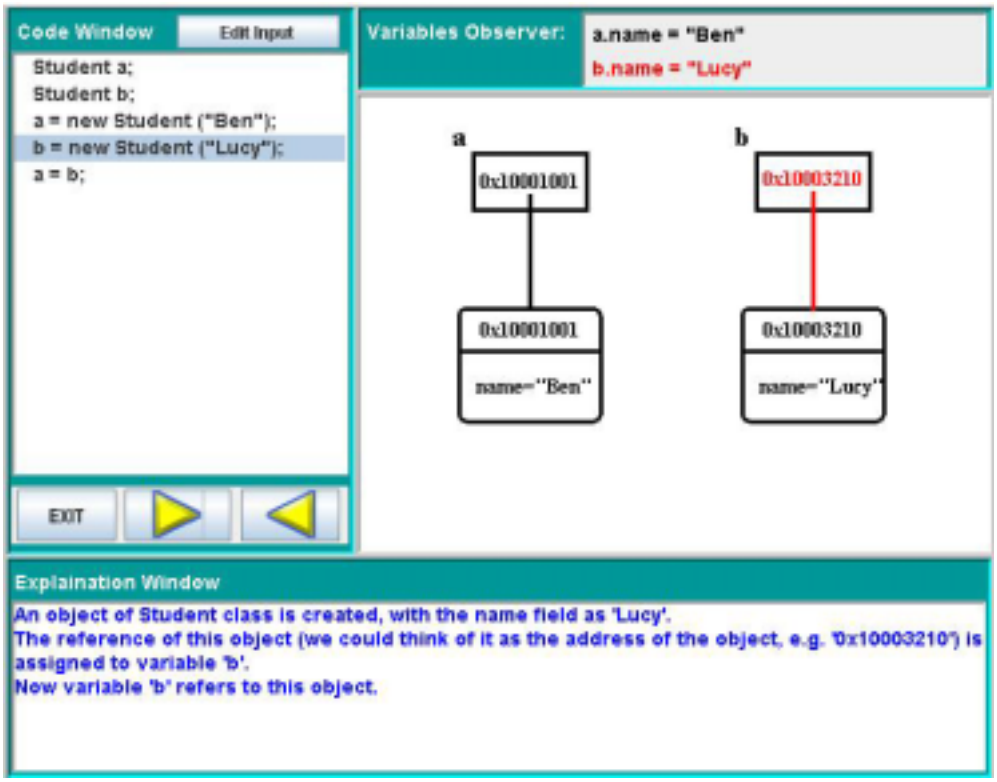


Figure 5-7: Create new objects and assign their reference to the reference variables

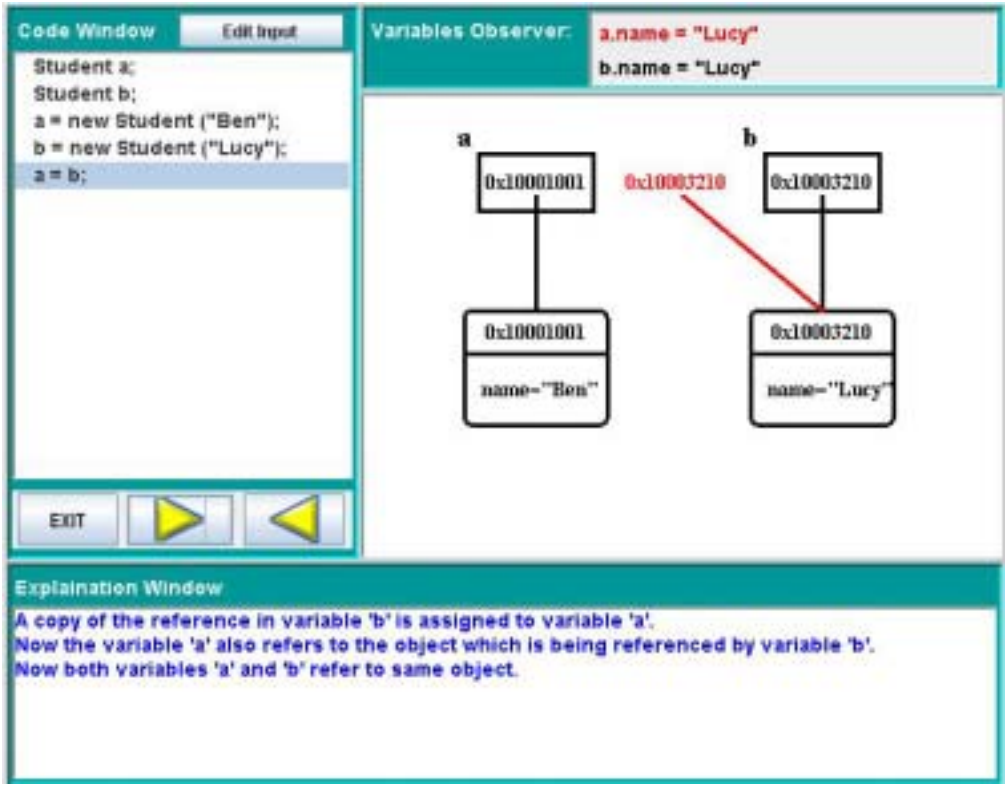


Figure 5-8: Assign a reference from a reference variable to another reference variable

5.3 Summary

This chapter proposes a constructivist-based teaching model that integrates a cognitive conflict strategy along with program visualization to improve novice programmers' mental models of fundamental programming concepts. A computer-supported learning tool to support the proposed learning model, developed by the author, is also described.

The next chapter presents a series of empirical studies investigating the effectiveness of the learning model.

CHAPTER 6 – Evaluation of the Proposed Teaching Model

To improve novice programmers' mental models of fundamental programming concept Chapter 5 proposed a constructivist-based learning model that integrates a cognitive conflict strategy and program visualization. This chapter presents three studies that were conducted to evaluate the effectiveness of the proposed learning model. The first study (section 6.1) focused on a relatively simple programming concept (value assignment) while the second study (section 6.2) focused on a relatively complex programming concept (reference assignment). The last study (section 6.3) assessed the students' mental models of both the value assignment concept and the reference assignment concept sometime after they completed the first two studies. Results of the last study were compared to the results achieved from the first mental model study (Chapter 4) conducted in the previous year when the students did not experience the proposed learning model.

6.1 An Evaluation of the Effectiveness of the Proposed Learning Model for Value Assignment

6.1.1 Research Aim

This study aimed to investigate the effectiveness of the proposed teaching model for changing novice programmers' mental models of a relatively straightforward concept, value assignment. In addition, special emphasis was on whether or not the application of the cognitive conflict teaching strategy would engage students into the visualization-based learning materials and improve the effectiveness of visualization.

6.1.2 Research Method

60 volunteers were recruited from students who were in the introductory programming course. These were different students from those who participated in the first 'mental model' test (The first 'mental model' test took place in the academic year 2005-2006, this study was carried in the year 2006-2007). The courses in both years had the same course structure and used the same teaching materials. In addition, the courses in both years were taught by same lecture. The experiment was arranged in a lab session (1 hour) in the fifth week of the course, after the participants had been introduced to and practiced the assignment concept. In the pre-test, participants' pre-existing mental models of the assignment concept were elicited using a simplified version of Dehnadi and Bornat's questionnaire (Appendix B-1). In addition, there was an additional answer option in each question for participants to state whether or not they thought the program fragment in the question could execute correctly. If they thought there was any problem in the program, they were asked to explain it. Participants who were found to be holding non-viable mental models were separated randomly and equally into two groups: the *CC+Viz* group, which used all the functions of the proposed teaching environment, i.e. they first experienced the conflict event and then used the visualization tool; and the *Viz* group which only used the visualization tool, but without experiencing a conflict event. In the post-test, the participants' mental models of the assignment concept were investigated again using a simplified version of Dehnadi and Bornat's questionnaire, but with different questions (Appendix B-2). In addition, the post-test questionnaire also included an open-ended question that asked the participants to describe the execution of a program which included assignment statements. Furthermore, a questionnaire (Appendix B-3) was employed to collect the participants' qualitative feedback on the teaching environment.

6.1.3 Results

The pre-test identified a list of mental models held by the participants (Table 6-1). **M2**, **M9**, **MIncon**, **M11Ss**, and **M2Ss** are the mental models from Dehnadi and

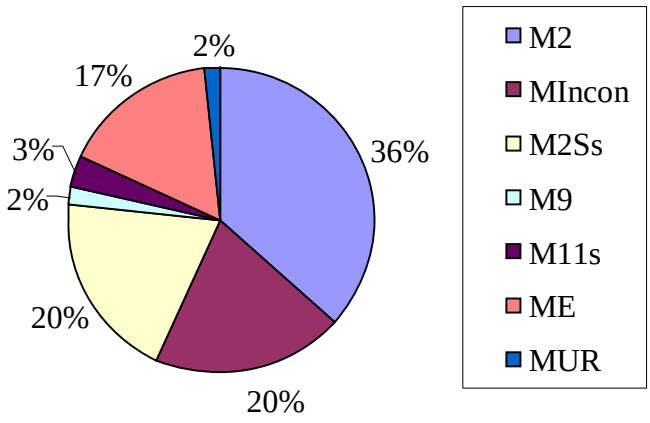
Bornat's mental model list, **ME** and **MUR** are the mental models identified in this research.

Model	Description of the Model
M2	A Java primitive type value is copied from the result of the evaluated expression on the right of the assignment operator to a variable on the left (**appropriate mental model**).
MIncon	Different models are used to answer the collection of questions.
M9	Nothing happens when an assignment statement is executed.
ME	Viewing '=' as a compare operator.
MUR	A variable can not be 'rewritten', i.e., the variable can be only written once.
M11Ss	Variables swap values when an assignment statement is executed + Ss Model
M2Ss	M2 + Ss

Table 6-1: The mental models identified in the value assignment study

22 (37%) participants were found to be consistently using the appropriate mental model, i.e. M2 on Dehnadi and Bornat's mental model list, while 12 (20%) participants used inconsistent models (MIncon), and 26 (43%) consistently used inappropriate models, covering M9, ME, MUR, M11Ss and M2Ss. The inappropriate models can be separated into two categories: the inappropriate models of the assignment process, covering M9, ME, MUR, and M11Ss; and the inappropriate models of execution flow, covering M11Ss and M2Ss. (Note that M11Ss actually covers two inappropriate models: '*M11*' is an inappropriate model of assignment; and '***Ss***' is the inappropriate model of execution flow.) According to Dehnadi and Bornat, the *Ss* model is "*derived from the misconception that assignments execute simultaneously; each line of code is an individual statement and should be treated separately*"

Figure 6-1 shows the number and percentage of participants for each model. The M2Ss model and ME model were most widely used inappropriate models.



M2	MIncon	M2Ss	M9	M11Ss	ME	MUR	Total
22	12	12	1	2	10	1	60

Figure 6-1: The number and percentage of participants for each model of value assignment

10 out of the 38 participants who held non-viable mental models (inconsistent models or consistently inappropriate model) did not finish the post-test. The data from the remaining 28 participants were available for analysis. Table 6-2 shows the distribution of these participants. The analysis of the participants' performance in the course assessment revealed that there was not a significant difference (p=74%) between those in the CC+Viz group and the Viz group.

Group	MIncon	M9	ME	MUR	M11Ss	M2Ss	Total
CC+Viz	6	0	2	1	1	4	14
Viz	2	1	4	0	1	6	14
Total	8	1	6	1	2	10	28

Table 6-2: The distribution of the participants whose data is available to analyse

As Table 6-2 shows, 18 participants (10 were in the CC+Viz group and 8 were in the Viz group) held inappropriate models of the assignment process (M9, ME, MUR, and M11Ss) while 12 participants (5 were in the CC+Viz group and 7 were in the Viz group) held inappropriate models of execution flow (M11Ss and M2Ss).

The result show that all 18 participants who held inappropriate mental models of the assignment process, no matter which group (CC+Viz or Viz) they were in, made

changes to their mental model of the assignment process (Table 6-3). 14 of them (78%) successfully changed their models into an appropriate one (only for the assignment process), while the remaining 4 participants changed their model from ME and M11Ss to inconsistent models. The CC+Viz group appears to perform slightly better than the Viz group: 9 out of 10 (90%) participants in the CC+Viz group changed their model of the assignment process to an appropriate one while 5 out of 8 (62.5%) participants made the successful change in the Viz group.

With regard to execution flow, the results showed that 6 out of 12 (50%) participants who held the *Ss* model changed their model to the appropriate one, while the remaining 6 participants did not make changes to their models (Table 6-4). Similar to the situation of model changing for the assignment concept, the CC+Viz group appears to perform slightly better than the Viz group: 3 out of 5 (60%) participants changed their models in the CC+Viz group while 3 out of 7 (43%) participants changed their models in the Viz group.

	Model Change Successfully					Model Change Failed		
Group	**MIncon => M2/Ss[10]**	**M9 => M2/Ss**	**M11Ss => M2/Ss**	**ME=> M2/Ss**	**Total**	**M11s => MIncon**	**ME => MIncon**	**Total**
CC+Viz	6	0	1	2	9	1	0	1
Viz	2	1	1	1	5	0	3	3
Total	8	1	2	3	14	1	3	4

Table 6-3: The distribution of participants who changed their mental model of assignment process

	Model Change Successfully			Model Change Failed		
Group	**M2Ss => M2**	**M11Ss => M2**	**Total**	**M2Ss => M2Ss**	**M11Ss => M2Ss**	**Total**
CC+Viz	2	1	3	2	0	2
Viz	3	0	3	3	1	4
Total	5	1	6	5	1	6

Table 6-4: The distribution of participants who changed their mental model of execution flow

[10] M2/SS means the model is M2 or M2Ss

Along with the quantitative data, qualitative data was also collected using a questionnaire. This feedback revealed three important characteristics of the teaching environment. Firstly, the animation to simulate the dynamic process of assignment could challenge a participant's pre-existing understanding of the assignment concept. Secondly, the animation was viewed as being very helpful in promoting understanding of the concept. Finally, the step by step execution was viewed as another helpful feature, which in the words of one student "*broke down the changes taking place*".

6.1.4 Discussion

The pre-test, which investigated the pre-existing mental models held by participants, revealed that students often held similar inappropriate mental models. There were 10 participants who held the ME model and 12 participants who held the M2Ss model, while only 4 participants held other inappropriate models. In this case, it would seem a relatively simple task for instructors to design learning materials that could change the inappropriate mental models held by most students. In addition, well-designed learning material can cover many different kinds of mental models, i.e. the same learning material is capable of changing different kinds of inappropriate mental models. For example, it is argued that the visualization tool used in this study is capable of changing all the inappropriate mental models of the assignment process identified in the pre-test. In addition, the cognitive conflict question is also capable of triggering cognitive conflict for all the inappropriate mental models of the assignment process identified in the pre-test. This implies that the 'conceptual change' based teaching strategy is a practical proposal.

In this study, all the participants, no matter which group they were in, made changes to their mental model of the assignment process. This implies that the visualization tool, even though not using an explicit cognitive conflict strategy, was also able to challenge a student's pre-existing ideas of the assignment concept.

One possible explanation for this is that the assignment concept is relatively straightforward. When the animation simulates the process of assignment, it is not difficult for participants (e.g. those who viewed '=' as an equal sign) to realised they were holding an inappropriate mental model. In addition, it also implies that the animation may be a better tool to promote conceptual change than the traditional textual and static learning materials. This study was conducted after the participants had covered the assignment concept using traditional learning materials delivered in a traditional lecture-based course. Those traditional learning materials did not help many of the participants to realise that their understanding of the assignment concept was inappropriate. It is even possible that the participants did not engage, or only engaged superficially, with the traditional learning materials.

Furthermore, the visualization/animation has been found to be an effective way to help students construct viable mental models of the assignment concept. Nearly 80% of participants constructed a viable mental model from their non-viable one by using the visualization tool. On the other hand, a few students still did not manage to construct a viable mental model of the assignment concept, even though they had realized their pre-existing mental model was inappropriate. For those students, the construction of viable mental models might require more time and support.

While the visualization tool helped improve the participants' mental models of the assignment process successfully, it seemed less effective for improving the mental models of execution flow. Half of the 12 participants who held inappropriate models of execution flow did not realize they were holding inappropriate models after using the visualization tool, even when some of them were challenged with a cognitive conflict question. As mentioned earlier, the inappropriate mental model derived from two misconceptions, namely: 1) assignments execute simultaneously; and 2) each line of code is an individual statement and should be treated separately. On reflection, while the step by step execution mode of the visualization tool is capable of correcting the first misconception, unfortunately, the example (figure 6-2a) used in the cognitive conflict question and the visualization tool failed to trigger cognitive conflict when a participant held the second misconception. As figure 6-2a shows, the

execution of Line1 does not affect the result of the execution of Line2, i.e. no matter whether or not the Line1 is executed, the result of Line2 is always 'b = 30; c=30'. In this case, even though the participants held the second misconception, they can still pass the example successfully. Actually, it is easy to solve this problem by using another example, e.g. the example in figure 6-2b.

```
int a =10, b=20, c=30;
Line1: a = b;
Line2: b = c;
```

(a) the current example

```
int a =10, b=20, c=30;
Line1: a = b;
Line2: c = a;
```

(b) the proposed example

Figure 6-2: The current example and modified example used in this study

In this case, it is not difficult to understand why so many participants in the experiment did not change their mental models of execution flow. This finding explains why some teaching materials (including visualization-based materials) are not always helpful for improving students' understanding, even though those materials have been viewed as well-designed by instructors. When instructors design materials based on their views, but without considering students' pre-existing mental models, those materials might miss models held by some students. In these cases, those students may not change their mental models, even though they are engaging with the materials. This reveals a weakness of the objectivism-based teaching approach and highlights the value of using a constructivism-based teaching approach.

It should be noted that there were some limitations and weaknesses in this experiment. Firstly, in order to minimize any external inferences on the results the participants experienced the learning model under examination conditions. However, a learning model should be more beneficial when it is used in a 'real' learning environment where students are allowed to communicate with other, to obtain assistance from tutors and to user other learning materials. Future work would be needed to investigate the performance of the model in a 'real' pedagogical context.

In addition, the study adopted an aggregated approach to assess the effectiveness of the proposed learning model. The studies did not provide an in-depth exploration of how an individual's mental models evolved, driven by the cognitive conflict strategy and the program visualization technique. Furthermore, the pedagogical benefits of the cognitive conflict strategy combined with a visualization technique might take some time before becoming effective. A long-term, continuous study is required to investigate the development of mental models.

When participants were confronted with the cognitive conflict event (i.e. they were told that their answers were incorrect), some participants might not achieve cognitive conflict. They might just ignore or resist the cognitive conflict event. This study employed a series of scaled questions to investigate the participants' attitude to the cognitive conflict event. However, the use of closed questions might give participants hints on how to answer the questions. In addition, some participants might have other reactions to the cognitive conflict event that were not predicted and were not taken into account when designing the scaled questions.

In this study, open questions were used to collect qualitative data that was expected to reveal how participants reacted to a cognitive conflict event and visualization. However, some participants provided too little, or invalid, information to allow an analysis of those participants' reactions to the proposed learning model.

In order to elicit participants' mental model before and after they had experienced the learning model, a pre-test and a post-test were carried out. There were six questions in each test. The participants claimed that there were too many questions that they had to answer, especially for the post-test when they believed that their understanding was appropriate. In this case, the participants might not answer the questions seriously.

This section has evaluated the effectiveness of the proposed learning model for improving novice programmers' mental models of value assignment. The results suggest that, for the relatively straightforward concept of assignment, tight

integration of program visualization with a cognitive conflict event that highlights a student's inappropriate understanding can help improve students' non-viable mental models. While most students successfully constructed a viable mental model of the assignment concept using the learning model, the importance of the cognitive conflict component within the model remains less obvious, perhaps due to the simplicity of the assignment concept. As mentioned above, the assignment concept is straightforward, and hence it was relatively easy for students to become engaged with the learning materials, no matter whether or not they were challenged explicitly by a cognitive conflict event. However, when learning a more complex concept, cognitive conflict is expected to play a more significant role in ensuring that students become more engaged with the learning materials. A further study is proposed to investigate the effectiveness of the teaching model for a more complex concept, reference assignment.

6.2 An Evaluation of the Effectiveness of the Proposed Learning Model for Reference Assignment

6.2.1 Research Aim

This study investigates the effectiveness of the proposed learning model that integrates a cognitive conflict strategy along with visualization technology with the aim of improving students' mental models of a relatively complex programming concept, namely, reference assignment.

6.2.2 Research Method

Participants

43 volunteers were recruited from students who were in the Programming Foundations course. 17 of them participated in the 'value assignment' experiment (Chapter 6.1). The study was conducted in the twelfth week of the course after the participants had covered fundamental object-oriented programming concepts. The

teaching materials presented in the course covered the reference concept, as well as a comparison between reference assignment and value assignment.

Procedure

This study was conducted in one lab session (1 hour) in the 12 week of the course. The procedure carried out for this experiment was similar to that employed in the 'value assignment' experiment described in the previous section. Firstly, the participants' pre-existing mental models of reference assignment were elicited by the pre-test. The participants who held non-viable mental models were separated randomly and equally into two groups: the *CC+Viz* group and the Viz group. In a similar manner to the 'value assignment' experiment, the CC+Viz group were first challenged by a cognitive conflict question designed to explicitly trigger the cognitive conflict event and then exposed to the visualization based material; the *Viz* group only experienced the visualization based material without the explicit cognitive conflict event. After the exercise, the participants' mental models were analyzed again as a post-test. At end of the experiment, participants were also asked to complete a 'feedback' questionnaire used to collect their views of the exercise.

Experiment Material

Pre-Test and Post-Test Questionnaire

The pre-test and post-test investigated the participants' mental models of the reference concept before and after an exercise designed based on the proposed teaching model. The test questionnaires cover three close-ended questions and one open-ended question (Appendix C-1). The close-ended questions ask participants to predict the execution result of a program which contains one or two reference assignment statements. Figure 6-3 shows the program used in the first close-ended question in the pre-test. The programs used in the second and third questions are slightly more complex: the program used in the second question contains two reference assignment statements; and the program used in the third question contains three reference variables and two reference assignment statements. The pilot study using three postgraduate students and three teaching staff in particular found that the third program was very challenging because it created a heavy cognitive load for the

participants. Even experienced programmers claimed that they needed to work carefully when tracing the execution of this program. Although this program seems overly complex for first year students, it is still interesting to investigate a student's performance when applying their mental model to solve a complex problem. Therefore, this program was not replaced by a simpler one. The questions in the post-test use the same question style but within a different problem context (Appendix C-2).

A *Staff* class has been defined that holds an *ID* field as an integer. Constructors like '*new Staff(1)* ' are used to create an object of *Staff* class with an ID of '*1* '.

Methods in the form '*changeID(5)* ' are used to change the value of the *ID* field of an object to be '*5* '.

The following questions require you to determine the value the *ID* field of the objects after the execution of the program fragment.

```
Staff a;
Staff b;
a = new Staff (1);
b = new Staff (2);
a = b;
b.changeID (5);

a.ID = _____;        b.ID = _____;
```

Figure 6-3: A close-ended question used in the pre-test

Compared to the test questions used in the early mental model study (chapter 4), there are two differences with the questions (figure 6-3) in this experiment. Firstly, there is an additional statement '*b.changeID(5)*' in the programs. Secondly, participants are asked to 'fill in' the result of program execution rather than choosing an answer from a list of diagram-based answer options. The reasoning behind this question style was to avoid possible 'pattern matching' with the pre-test occurring when participants answered the post-test question. The conceptual model used in the visualization employed similar diagrams to present the memory states. It was possible that some participants would choose the correct diagram-based answer option by matching it with the diagram shown in the visualization but without substantial understanding of the reference concept. However, the use of the statement

'*b.changeID(5)*' solved this problem. With this approach it was difficult for participants to select the correct answer if they did not hold a good understanding of the concept.

The early 'Mental Models' study (Chapter 4) identified a collection of mental models of reference and reference assignment held by novice programmers. A list of possible answers is predicted based on the collection of mental models. Table 6-5 shows the mapping between the answer options and mental models. The first answer A1 is mapping to the appropriate mental model, while the remaining answers are mapped to improper mental models. As table 6-5 has shown, the remaining answers, e.g. A2, are mapping to more than one inappropriate mental models. So, the close-ended question can only test whether or not participants hold an appropriate mental model, but can not test which inappropriate model a participant holds. In this case, an open-ended question is designed to help identify the inappropriate model held by a participant. This open-ended question asks participants to describe what happens when the statements '*a = new Staff (1);*' and '*a = b*' are executed. In addition, the open-ended question is also able to investigate whether or not participants held other models that can not be identified by the close-ended questions, such as the **ME** model, i.e. *viewing assignment as a test of equality*. This model has been found as the most widely used inappropriate model of assignment in the 'value assignment' experiment.

No.	Answer	Mental Model
A1	a.ID = 5 b.ID = 5 **(Q1)** a.ID = 5 b.ID = 5 **(Q2)** a.ID = 2 b.ID = 5 c.ID = 5 **(Q3)**	**MA – Appropriate mental model** of reference and assignment. Variables hold a reference; and an reference is assigned from right side to left side

A2	a.ID = 2 b.ID = 5 **(Q1)** a.ID = 2 b.ID = 5 **(Q2)** a.ID = 2 b.ID = 5 c.ID = 3 **(Q3)**	**MStored** – An object is stored at a variable. **MIs –** *a* and b are not variables, but rather the name of objects. **MComponentAssign –** the value of ID field in an object is replaced by the value of ID field in another object When the participants held any one of the models of the reference concept above, they also held the following models of the assignment concept: **MRtoL –** the assignment is from right-hand side to left-hand side (appropriate model of assignment)
A3	a.ID = 2 b.ID = 5 **(Q1)** a.ID = 1 b.ID = 5 **(Q2)** a.ID = 2 b.ID = 5 c.ID = 3 **(Q3)**	The participants held one of the **MStored**, **MIs**, and **MComponentAssign** models of the reference concept, and the following models of the assignment concept: **MRtoL(Ss)–** the assignment is from right-hand side to left-hand side (appropriate model of assignment); but the participants held an inappropriate mental model of execution flow
A4	a.ID = 2 b.ID = 5 **(Q1)** a.ID = 1 b.ID = 5 **(Q2)** a.ID = 2 b.ID = 5 c.ID = 1 **(Q3)**	The participants held one of the **MStored**, **MIs**, and **MComponentAssign** models of the reference concept, and the following models of the assignment concept: **MSwap –**Variables swap values
A5	a.ID = 2 b.ID = 5 **(Q1)** a.ID = 1 b.ID = 5 **(Q2)** a.ID = 2 b.ID = 5 c.ID = 2 **(Q3)**	The participants held one of the **MStored**, **MIs**, and **MComponentAssign** models of the reference concept, and the following models of the assignment concept: **MSwap(Ss) –**Variables swap values; but the participants held an inappropriate mental model of execution flow
A6	a.ID = 1 b.ID = 5 **(Q1)** a.ID = 1 b.ID = 5 **(Q2)** a.ID = 1 b.ID = 5 c.ID = 1 **(Q3)**	The participants held one of the **MStored**, **MIs**, and **MComponentAssign** models of the reference concept, and the following models of the assignment concept: **MLtoR –** the assignment is from left-hand side to right-hand side.
A7	a.ID = 1 b.ID = 5 **(Q1)** a.ID = 2 b.ID = 5 **(Q2)** a.ID = 1 b.ID = 5 c.ID = 2 **(Q3)**	The participants held one of the **MStored**, **MIs**, and **MComponentAssign** models of the reference concept, and the following models of the assignment concept: **MLtoR(Ss) –** the assignment is from left-hand side to right-hand side; but the participants held an inappropriate mental model of execution flow
A8	a.ID = 1 b.ID = 5 **(Q1)** a.ID = 1 b.ID = 5 **(Q2)** a.ID = 1 b.ID = 5 c.ID = 3 **(Q3)**	The participants held one of the **MStored**, **MIs**, and **MComponentAssign** models of the reference concept, and the following models of the assignment concept: **MUnchange –** Nothing happens **MUnchange(Ss) –** Nothing happens; and the participants held the inappropriate mental model of execution flow

Table 6-5: The mapping between the answer options for the close-ended questions and mental models

Apart from the close-ended questions and the open-ended question, there is also a question to investigate how confident the participants felt about their answer. The participants evaluated their level of confidence against a 5 point scale (from 1– strongly unconfident to 5-strongly confident).

The Learning Environment

The learning environment designed to assist the proposed learning model was extended with support for the object and reference assignment concept. The learning environment first challenges the participant with a question in order to trigger cognitive conflict. This question (Appendix C-3) is in similar style to those used in the pre-test but uses a different problem context. After participants present their answer to the question, the system will tell them whether or not the answer is correct. Then the visualization tool (figure 6-4) is used to help participants develop mental models of the reference concept. As the Code Window in figure 6-4 shows, the segment of the program does not cover the statement of '*b.changeName("Tom")*'. This is to avoid a potential 'pattern match' problem occurring when participants answer the post-test question. If participants are presented with the visualization of this statement, it is possible that the participants just remember the pattern superficially without substantial understanding of the reference concept. In this case, they may be able to produce a correct answer to the post-test question but with inappropriate or unstable mental models.

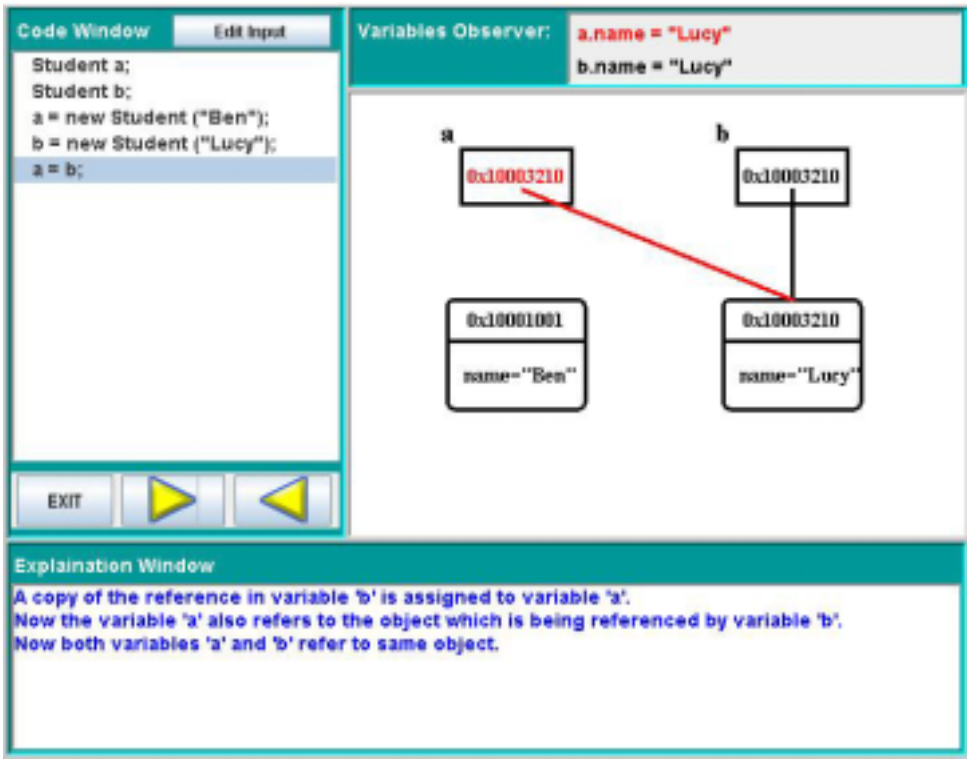


Figure 6-4: The visualization of reference assignment

The Feedback Questionnaire

The Feedback Questionnaire was designed to collect participants' feedback on the proposed teaching model. In the 'value assignment' experiment, all the questions in the feedback collection questionnaire were open-ended. Although open-ended questions are capable of collecting unanticipated information, the disadvantage of this type of questions is also obvious. A large number of participants did not provide valid information in response to the open-ended questions but rather presented fragments such as "the visualization is useful" without any explanation why the visualization is useful. To address this, some closed questions are used in this experiment in order to gather more valid information.

Appendix C-4 shows the questions for both the CC+Viz group and the Viz group. The first question asks participants whether or not they have changed their understanding of any programming concepts as a result of the exercise and what are the changes. The second question asks participants when they realized their original understanding was incorrect. The third question asks how much attention they have paid to the textual explanation in the visualization tool, and how much they have

understood it. The reason for using this question is that the pilot study found the users of the visualization tool only focused on the graphical materials but ignored the textual explanation. It is interesting to investigate whether or not the participants who experienced a cognitive conflict event pay more attention to the textual materials. The last question asks participants to describe any questions or issues they have after the exercise.

Apart from the questions above, the participants in the CC+Viz group were asked to answer two extra questions (Appendix C-5). The first question, which appeared just after the participants answered the cognitive conflict question, asked the participants about their reaction to the cognitive conflict event, while the second question, which appeared after the exercise, asked the participants about the effects of the cognitive conflict event to their knowledge construction. Please refer to Appendix C-5 for the details of the answer options.

6.2.3 Results

The Results of Pre-test

The result of the pre-test shows that only 2 participants consistently used answer **A1** (i.e. the correct answer) in table 6-5 to answer all the pre-test questions. Both of them gave appropriate and clear explanations of reference assignment in the open-ended question. One participant also provided an appropriate description of the difference between value assignment and reference assignment.

24 out of the remaining 41 participants consistently used answer **A2** in table 6-5 to answer all the pre-test questions. The open-ended question identifies that: 6 of them held the MStored model; 11 of them held the MIs model; 2 of them held both the MIs and MComponentAssign model. In addition, 3 participants left the answer blank or gave uninterpretable information (i.e. "*it finds what b is then assigns a to be the same thing*"). A participant described the statement 'a = b' as "*a is referenced to b*". More interestingly, a participant described the statement 'a = new Staff (1)' as "*creates new array*" (the participants had just learned the array concept before the experiment).

The pre-test also found that 2 participants consistently used the answer **A6** in table 6-5 (i.e. viewing assignment as from the left-hand side to the right-hand side) to answer all of the pre-test questions. One of them held the MStored model when the other one held the MIs model.

The remaining 15 participants used the inconsistent model to answer the close-ended questions. The open-ended question found that: one participant used the Mstored model; 2 participants used the MIs model; 2 participants used the MIs and MComponentAssign model; and 7 participants left it blank. In addition, 4 participants provide strange descriptions, such as:

"a is set to whatever is in location 1 in the array staff"

"a is made equal to 1"

"A new staff element is created at index 1."

"a will be given the variable new Staff(1)"

It is difficult to identify the mental model from their descriptions, but they seem to hold a very inappropriate understanding of the variable and object concepts.

In addition, the results of the pre-test also show that the participants with a consistent mental model were more confident than those with inconsistent models ($p^{11} = 0.042$).

The Result of Post-Test

The 41 participants who held inappropriate or inconsistent mental models were separated into the CC+Viz group and the Viz group. 3 participants in the CC+Viz group did not finish the post-test. One of them went back to the cognitive conflict question after using the visualization rather than going forward to the post-test. He or she kept doing the 'cognitive conflict question -> visualization -> cognitive conflict question' cycle until they got the correct answer. Apart from the 3 participants, 18 participants were in the CC+Viz group while 20 participants were in the Viz group. The analysis of the participants' performance in the course assessment revealed that there was not significant difference (p=11.1%) between those in the CC+Viz group and the Viz group.

The results of the post-test along with the result of the first question in the feedback questionnaire (i.e. asking participants to describe what changes they had made to their understanding of the programming concepts) show that the participants can be grouped into five categories:

Category 1 – the participants can answer at least the first two close-ended questions correctly.

Category 2 – the participants cannot answer the close-ended questions correctly, but their answers to the open-ended questions show that they improved their understanding of the reference concept.

[11] The Wilcoxon Mann-Whitney Test was used to compare the consistent group and the inconsistent group. The Wilcoxon Mann-Whitney Test was used as the data was not normally distributed.

Category 3 – the participants cannot answer the close-ended questions correctly, and they cannot give appropriate descriptions of the reference concept in the open-ended question. However, their understanding of other concepts such as variable and object were changed after the exercise, even though they may not have changed to the appropriate one.

Category 4 – the participants realized their understanding of reference or other concepts were inappropriate, but they did not make any change of their understanding.

Category 5 – the participants did not realize their understandings of reference or other concepts were inappropriate after the exercise.

Table 6-6 shows the number of participants in each category.

	Category 1	**Category 2**	**Category 3**	**Category 4**	**Category 5**
CC+Viz	4	1	9	4	0
Viz	0	5	4	4	7

Table 6-6: The distribution of participants in terms of the mental model changes

Category 1

The result shows that only one participant answered all the three close-ended questions correctly. However, another 3 participants gave correct answers to the first two questions but failed the third one. As mentioned above, the third question, which places a heavy cognitive load, is relatively challenging for novice programmers. Even experienced programmers who hold appropriate mental models of the reference concept found it easy to make a mistake. In this case, the participants who could answer the first two questions correctly were put into category 1. All of the 4 participants were from the CC+Viz group.

Apart from one participant who did not provide any valid information to interpret, all the remaining 3 participants in the category 1 presented appropriate descriptions to the open-ended questions, such as:

"*A reference to an instance of Account with a value of 100 for balance is stored in acc1.*"
"*The reference inside acc2 is made equal to the reference inside acc1.*"

The results of pre-test reveal that: 2 participants in the category 1 held MIs model before the exercise; one held both the MIs model and the MComponentAssign model; and one held inconsistent models.

Category 2

6 participants were in category 2: one was from the CC+Viz group while 5 were from the Viz group. In this category, the participants gave incorrect answers to the post-test questions (5 participants consistently used answer **A2** for all the questions, while one participant gave inconsistent answers) even though their answers to the open-ended question show that their understanding of reference concept was improved, such as:

From "*The staff object assigned to variable b is assigned to variable a*" **To** "*The variable acc2 now refers to the same object as the variable acc1"*

From *"The value in b is being assigned to a."* ***To*** *"Object acc2 points to the location of the acc1 object. When an object is assigned to another object it doesn't copy the object to this object but it points to the location of the object"*

From "*The value of b is transposed onto the value of a. b stays the same but a changes.*" **To** "*The reference of the object in acc1 is copied, replacing the reference to acc2 meaning that the two objects are now being referenced at the same time.*"

However, the descriptions of reference presented by some of the participants are still not accurate. For example, in the third example description, the first half of the sentence is correct that "*The reference of the object in acc1 is copied, replacing the reference to acc2*", but the second half of the sentence, "*the two objects are now being referenced at the same time*", shows that the participant did not construct an appropriate mental model of reference.

The results of the pre-test reveal that: one participant in the category 2 held the MStored Model before the exercise; 2 participants held the MIs model; one held the MComponentAssign model; and one described the statement 'a = b' as "*a is referenced to b*".

Category 3

The results of the close-ended questions show that the participants in this category did not change their model to an appropriate one. However, the results of the open-ended questions show that they seem to have made changes to their understanding of some programming concepts, such as variable and assignment. In addition, the participants in this category seem able to construct an appropriate mental model of value assignment that is a simplified version of the mental model of reference assignment. For example, a participant explained the statement '*a = new Staff(1)*' as "*a is set to whatever is in location 1 in the array staff*" in the pre-test and explained the statement '*acc1 = new Account(100)*' in the post-test as "*whatever was in acc1 has been changed to 100*" in the post-test. The participant's answer to the open-ended question shows that they held a very inappropriate understanding of object concept. In their understanding, the class Staff is an array and the parameter is the index of the array. After the exercise, their understanding seems to have changed to the appropriate one of value assignment.

However, most of the participants in this category seem just to have made a very superficial adaptation of their understanding, rather than making a change to their mental model. They just used the phrases that appeared in the visualization to re-describe their model. For example, a participant described the statement '*a=b*' as "*a's value, 1, is changed to b's value, 2*" in the pre-test, and "*acc2 is made equal to the value stored in acc1*" in the post-test.

The comparison of the pre-test and the post-test shows that 9 participants are in this category: 5 changed from inconsistent model to a consistent model; 3 changed in the opposite direction; and 1 changed from the consistent model, MIs (assign from left to

right) to another consistent model, MStored. The remaining 4 participants in this category did not change their answer options to the close-ended questions.

Category 4

The participants in this category meet all of the following conditions:

- They answered 'No' to the first question in the Feedback questionnaire that asks them to describe the changes they had made to their understanding of programming concepts.
- They used the same answer option for the close-ended questions in the pre-test and the post-test.
- They used the same descriptions for the open-ended questions in the pre-test and the post-test.
- They gave an answer to the second question in the feedback questionnaire, which asks them when they realized their original understanding was incorrect.

There are 8 participants in this category: 4 were in the CC+Viz group and 4 were in the Viz group.

Category 5

The participants in this category did not realize that their understandings were inappropriate. Unsurprisingly, all the participants were in the Viz group (the participants in the CC+Viz group were explicitly told that their answer to the cognitive conflict question was incorrect). 7 participants were in this category.

The Data from the Feedback Questionnaire

There is a question in the feedback questionnaire which investigated when the participants realized their original understanding was inappropriate. The result identifies four periods:

Period 1: when they were answering the pre-test questions

Period 2: when they were told that their answer was incorrect to the cognitive conflict question

Period 3: when they were viewing the visualization tool

Period 4: when they were answering the post-test questions

In addition, there were some participants who were not aware that their understanding was incorrect.

	Period 1	**Period 2**	**Period 3**	**Period 4**	**Not Aware**
CC+Viz	0	9	5	4	0
Viz	5	0	6	2	7
Total	5	9	11	6	7

Table 6-7: The distribution of the participants in term of the period when they achieved cognitive conflict

As Table 6-7 shows, 9 out of 18 participants in the CC+Viz group realized their original understanding was inappropriate when they were told that their answer was incorrect to the cognitive conflict question. The remaining participants in the CC+Viz group realized the inappropriateness when they were viewing the visualization tool or when they were answering the post-test questions. None of the participants in the CC+Viz group realized the inappropriateness when answering the pre-test questions

In the Viz group, none of participants answered the cognitive conflict question. So, none of participants realized the inappropriateness of their understanding during period 2. 6 of the remaining participants realized their original understanding was incorrect when they were viewing the visualization tool. 2 participants realized the inappropriateness when they were answering the post-test questions. In addition, 5 participants realized the inappropriateness when they were answering the pre-test questions.

There was also a question to investigate how much attention the participants paid to the textual explanation in the visualization tool, and how much they understood it. The results show that the participants in the CC+Viz group did not pay more attention than those in the Viz group ($p = 0.803$). In addition, the CC+Viz group did not have any better understanding of the textual materials than the Viz group ($p=0.363$). However, perhaps unsurprisingly, the results show that the participants in

the **Categories 1 and 2** have better understanding of the textual materials than those in the **Categories 3, 4, 5** (p=0.046). All of the participants in Categories 1 and 2 understood at least 'most' of the materials. In particularly, 3 out of 4 participants in Category 1 understood 'all' of the materials.

The participants in the CC+Viz group had two extra questions. The first question asked their reaction when they were told their answer to the cognitive conflict question was incorrect. They could choose one or more answers from a pre-defined answer list (please refer to C-2). The results show that 5 participants claimed "*I was surprised because I thought my answer was correct*" and 4 picked "*I was surprised and even wondered if there is something wrong with the system*". 10 participants claimed "*I was interested to know why my answer was incorrect*". None of participants choose the following answer options:

- *To be honest, I did not really care about whether or not my answer is correct.*
- *I was not surprised because I was not confident with my answer.*
- *I felt upset*

The participants who felt 'surprised' seem to perform slightly better than the others. 3 of them were in Category 1, while only 1 of the other 10 participants were in this category.

The second extra question asks the participants about the effects of the cognitive conflict event to the learning exercise. As with the first extra question, the participants can choose one or more answers from a pre-defined answer list (please refer to C-2). The results show that most of the participants gave positive feedback to the cognitive conflict strategy. 7 participants claimed "*It helped me to concentrate on the learning materials when I was told beforehand that my understanding was incorrect*". 4 participants picked "*It helped increase my interest in the learning materials when I was told beforehand that my understanding was incorrect*". 10 participants chose "*It encouraged me to actively seek the reason why my understanding was mistaken when I was told beforehand that my understanding was incorrect*". Only 2 participants chose the negative comment "*It did not make any*

difference when I was told beforehand that my understanding was incorrect". None of the participants claimed "*It decreased my interest in the learning materials when I was told beforehand that my understanding was incorrect*".

The Observation Data

Some participants seemed not to engage with the learning materials. When these participants were using the visualization tool, they just went through each statement quickly. After the exercise, the participants still used inappropriate mental models.

Some participants raised questions during the experiment. This provided some useful information to the experimenters. A participant questioned the answers of the post-test provided by the learning environment, and believed his answer was correct. The experimenter asked him to explain the visualization of the execution of each statement. His explanation shows that he held the MStored model, but more interestingly, he interpreted the visualization as matching the MStored model. It proposed that the visualization did not change his original mental model, but rather he used his original mental model to interpret the visualization.

6.2.4 Discussion

The results of the pre-test reveal that novice programmers find difficulty when grasping the reference concept, one of the most important concepts in object-oriented programming. Only 2 out of 43 participants in this study held viable mental models of the reference concept. This matches the findings of the earlier mental model study (presented in Chapter 4) where only 17% of all participants held viable mental models when they had been learning object-oriented programming for one year. It is not difficult to explain why so many students were not able to change their mental model of the reference concept to an appropriate one after experiencing traditional instruction. In many situations, even though the students held inappropriate mental models of the reference concept such as the 'store at' model, i.e. 'an object is stored at a variable', they can still make many programs execute successfully. This

strengthens many students' confidence in their inappropriate models. In this case, extra efforts are required to help students to discover their inappropriate mental models and construct an appropriate one.

The results of the post-test separated the participants into five categories. Those in the first category could answer at least the first two questions correctly. A necessary condition to do that is that the participant had to hold an appropriate mental model of the reference concept, i.e. the reference rather than the object is stored in the variable, and both variables 'point to' the same object when an reference is assigned from the right hand side variable to the left hand side variable.

The participants in the second category were those who failed to present correct answers to the close-ended questions, but whose answers to the open-ended question showed that they had improved their understanding of the reference and reference assignment. There are two possible explanations for this category of participants. Firstly, the participants realized that their original understanding of the reference concept was not appropriate and tried to construct an appropriate one. However, they did not succeed and only managed to construct a mental model that was only partially functional. It is possible that these students require more time to construct a viable model. Secondly, it was also possible that the participants just superficially remembered the 'phrases' as they appeared in the visualization tool and then 'copied' them to their answer the open-ended question, but without substantial change of their mental models.

The participants in the third category failed to construct an appropriate mental model of the reference concept, but they appeared to be able to build an appropriate mental model of variable and assignment concepts. They tended to construct a mental model that was appropriate for value assignment, which can be viewed as a simplified version of the mental model of reference assignment. Previous researchers (Norman, 1983; Doyle et al., 2001) found that people often build a simplified version of the mental model for a complex system. An appropriate mental model of the reference assignment concept is much more complex than that of the value assignment concept.

Students have to understand how an object is created, as well as its relation with the reference variable, in order to construct an appropriate mental model of reference assignment. In addition, it is difficult for a novice programming student, who does not have appropriate mental models of the memory mechanism, to understand why reference assignment works in this way. For example, their prior experience, especially when they are holding the appropriate model of the value assignment concept, may drive them to believe that 'an object is stored in a variable and a copy of it replaces the object stored in the left side variable' is a more reasonable explanation. In this case, it is possible that they use a more reasonable way, in their opinion, to understand the learning materials. According to the Conceptual Change Model (CCM) from Posner et al. (1982) and Hewson & Hewson (1984), a necessary condition for conceptual change is that the new conception has to be plausible. Obviously, students tend to accept an idea that they view as reasonable.

The participants in the fourth category knew that there might be something wrong with their understanding of the programming concepts presented in the learning materials, but they did not manage to improve their understanding. A comparison of their answers in the pre-test and post-test also reveals that they had not made any change to their understanding after using the learning materials. The learning materials might have helped them to realize the complexity of reference assignment and perhaps led them to feel apprehensive about their existing understanding. However, their base knowledge was not enough to help them to understand the basic components of the mental model of reference assignment, such as how an object is created and stored in memory. In this case, they just knew their ideas were inappropriate but did not understand why.

The participants in the fifth category did not realize their mental models were inappropriate. One possible reason for this was that these participants were not engaged with the learning materials. In addition, other researchers on conceptual change (Hewson & Hewson, 1984) have found that students who hold prior conceptions are often prevented from accepting new conceptions. They do not realize their conceptions are inappropriate even though they are confronted with scientific

conceptions that are presented in lectures and tutorials. Moreover, it was possible that the participants interpreted the learning materials, such as the visualization, based on their existing mental model. Certainly, it was also possible that those participants might be totally lost. They could understand nothing of the learning materials.

In the previous study presented in Chapter 6.1, the visualization technique was found to be very effective in helping students construct viable mental models of the value assignment concept, a relatively straightforward concept. However, the visualization technique appeared much less effective in this study when it was applied to a complex concept, namely, reference assignment. Compared to value assignment, to construct a viable mental model of reference assignment requires students to have more base knowledge, such as those related to object creation and storage. It is very possible that the students had not built that base knowledge when they started to learn the reference assignment. In addition, Ben-Ari (2001a) also suggests that novice programmers lack the necessary model of a computer such as the memory mechanism to support their learning of computer science courses. It is difficult for these participants to construct a viable mental model of reference assignment without this base knowledge, especially when some participants even described an object as an array. Even though the visualization tool simulated the process of object creation and reference assignment, many participants seem unable to understand the visualization and the textual explanations. It matches the suggestion from other researchers (Ben-Ari, 2001b) that the visualization has to be designed to suit students' levels in order to achieve pedagogical benefits. In other words, no matter how well the visualization is designed, it cannot help students construct viable mental models if the students' prior knowledge is not enough for them to understand the visualization.

In addition, this finding also provides evidence to support Ben-Ari (2001a)'s idea, that can be announced as "*Don't start with abstraction*". Object-oriented programming concepts, such as class, object, and reference, are at a relatively high level of abstraction. Abstraction is crucial for programming. However, programmers

have to understand the low level underlying model of the abstraction in order to use it properly (Ben-Ari, 2001a). As Ben-Ari (2001a) mentioned, professional software engineers who can understand and use abstraction properly have a fairly good idea of the underlying model. However, novice programming students have not constructed these underlying models when the object-oriented programming concepts are presented at an early stage. In this case, it would be more reasonable to help students construct the low level models underlying the object-oriented programming concepts before teaching those more abstract concepts.

Moreover, the finding also suggests that programming concepts have to be taught to students in an appropriate order. For example, the student has to first construct appropriate mental models of the base concepts, such as variables, objects and assignment, before learning the reference concept. However, programming instructors are often confronted with a large population of students with different learning paces. It is impossible for instructors to wait for those 'slow' students. In this case, additional, self-managed learning sessions may be practical and helpful to 'mend' students' mental models. Actually, these learning sessions can be easily implemented based on the proposed teaching model: students can do the cognitive conflict–based practices by themselves to help identify the problems in their understanding, and try to improve their mental models by using the visualization-based materials.

Another explanation for students' failures to present viable mental models in the post-test is that the construction of viable mental models may take longer. Researchers (e.g. Hand & Treagust, 1988) who studied the conceptual changes in the science education domain proposed that students may have two conceptions, the inappropriate one and the scientific one, for some period of time after they receive teaching of the new scientific concept. They often have not fully accepted the new concept and in the meantime have not rejected the pre-existing concept. The new concept can only be accepted when it makes sense to the students with enough experience of using the concepts in more conflicting situations.

The previous study that focused on the value assignment concept did not identify the importance of a cognitive conflict teaching strategy. However, the results of this study show that the participants experiencing the explicit cognitive conflict event performed better than the others. Four participants from the CC+Viz group constructed appropriate mental models of reference assignment and answered the post-test questions correctly. On the other hand, nobody from the Viz group was in this category. In addition, without experiencing the cognitive conflict event, 7 out of 20 participants from the Viz group did not even realize their original understanding was inappropriate. The data from participants' responses to the feedback questionnaire reveals that the cognitive conflict teaching strategy is indeed capable of increasing students' interest in the learning topic, and engages them in the learning materials.

However, the results also show that there were still many participants from the CC+Viz group who could not locate where the problem was and construct an appropriate mental model. Although they knew their understandings were not appropriate after experiencing the cognitive conflict event, they could not find what and where the problem was. Perhaps, it is not surprising because these participants lacked the necessary base knowledge to explore the problems. In this case, these participants actually did not reach a cognitive conflict state because they did not identify and understand the conflict between their existing mental model and the scientific model.

The study identified four possible stages during which the participants realized their original understanding was inappropriate. In the CC+Viz group, half of the participants realized the inappropriateness of their understanding when they were experiencing the cognitive conflict event. The remaining participants realized theirs during use of the visualization tool or answering the post-test questions. For these participants, although they were told their answer was incorrect in the cognitive conflict event, it was still not enough to convince them that they were holding an inappropriate understanding. For example, some participants even suspected there was something wrong with the system. When they continued to use the visualization-

based materials or were confronted with other problem contexts in the post-test, there was more information to challenge their beliefs. In the Viz group, the participants did not experience the cognitive conflict event. Interestingly, some participants claimed that they realized their understanding was inappropriate during the pre-test. It perhaps implies that the common, traditional learning activity such as exercises and exams could drive students to doubt their understanding as well.

This study was carried out in similar way with the study for evaluating the effectiveness of the learning model for the value assignment concept (section 6.1). Therefore, this study has similar limitations and weaknesses as those mentioned in section 6.1.4. In addition, this study has some extra limitations and weaknesses. For example, to avoid 'pattern match' risk, as mentioned above, the participants did not answer a series of diagram-based, multi-choice questions (that were used in the mental models test) in the pre-test and post-test. Instead, they were asked to provide their own answer. As table 6-5 shows, one possible answer may map to more than one mental model. Therefore, the pre-test and post-test can only test whether or not the mental model held by a participant was viable or non-viable, but cannot automatically identify which mental model the participants were holding. In this case, an additional open question was required to identify the participants' mental model. However, when a participant did provide valid information to the open question, their mental model could not be identified.

This section reported a study to assess the effectiveness of the proposed learning model for a relatively complex concept, reference assignment. This study used a similar research method and experiment procedure to the ones employed in the first study for value assignment. The results show that the proposed learning model was capable of motivating students to engage with the learning materials and help students construct viable mental models, although the visualization technique seems less effective when students lack enough base-knowledge to interpret the visualization.

6.3 A Comparison between the Original Study and the Proposed Teaching Model Study.

As chapter 4 presents, the first year programming students' mental models of the value assignment and reference assignment concepts were investigated at the end of the Programming Foundations course in the 2005-2006 academic year. In that year, students did not experience the proposed learning model. The studies (chapter 6.1 and chapter 6.2) conducted to apply and assess the learning model were carried out in the 2006-2007 academic year. It is of interest to compare the mental models held by the students in 2005-2006 when they did not experience the learning model to those held by the students in 2006-2007 when they experienced the learning model. This section presents a study to elicit students' mental models at the same time as the mental model test conducted in 2005-2006 and to compare the mental models between these two years.

6.3.1 Research Aim

The aim of repeating the mental model test was to investigate whether or not the students who experienced the proposed teaching model during the academic year (2006-2007) performed better than those who did not experience this model in the previous year (2005-2006). In addition, this test would be used to investigate the long-term effects of the proposed teaching model.

6.3.2 Research Method

In order to form a comparison between the mental models held by the students in this year and those held by the students in the previous year, this test used the same test questions and was conducted under the same conditions as the test conducted in the previous year.

Sixty-six first year programming students from the Programming Foundations course participated in this test in week 22 of the course: 36 of them participated in the 'value assignment' experiment (chapter 6.1) in week 6 of the course; 32 of them

participated in the ‘reference assignment’ experiment (chapter 6.2) in week 12 of the course; and 12 of them had not participated any experiment.

The Programming Foundations course taught this year was very similar to the one taught in the previous year. Both courses adopted the same course structure, teaching materials, and were taught by the same lecturer. In addition, the students have been found to hold similar backgrounds on programming. The only difference was that some of the students from this year participated in the experiments and experienced the proposed teaching models.

6.3.3 Results

This study aimed to compare the mental models held by the students who experienced the proposed teaching model during the academic year (2006-2007) and those held by the students who did not experience the teaching model in the previous year (2005 - 2006). In this case, only the results from the students who participated in the ‘value assignment’ or/and ‘reference assignment’ experiment were analyzed.

There were 36 out of 66 students that participated in the ‘value assignment’ experiment. The results show that 30 out of them (84%) used consistently appropriate mental models to answer the ‘value assignment’ questions in this test. The figure is higher than that (63%) in last year (Figure 6-5). There was a smaller percentage of students using inconsistent or consistently inappropriate mental models this year compared to last year.

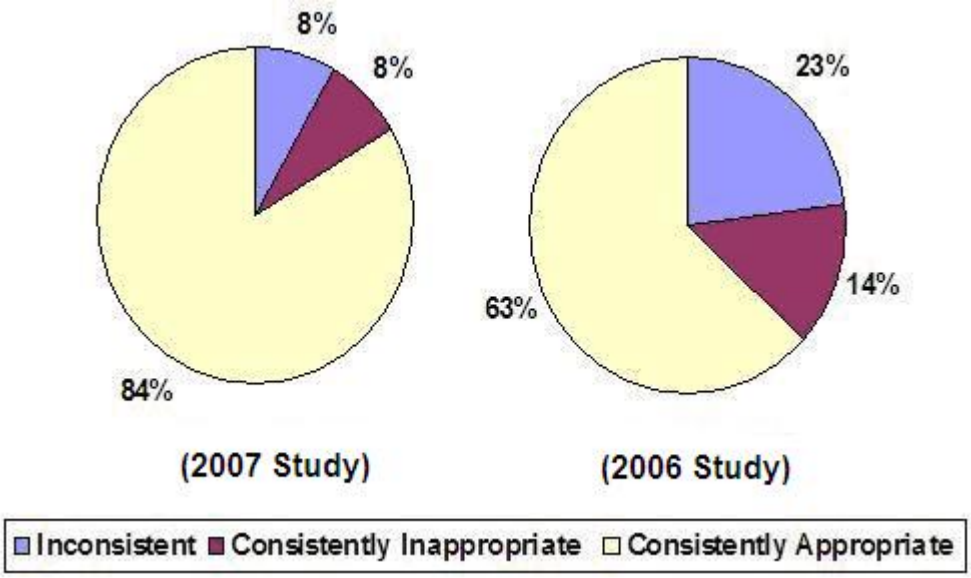


Figure 6-5: The comparison of students' mental models of value assignment

Thirty-two out of 66 students participated in the 'reference assignment' experiment. Half of them used consistently appropriate mental models to answer the 'reference assignment' questions in this test while only 17% of students used consistently appropriate mental models in last year (Figure 6-6).

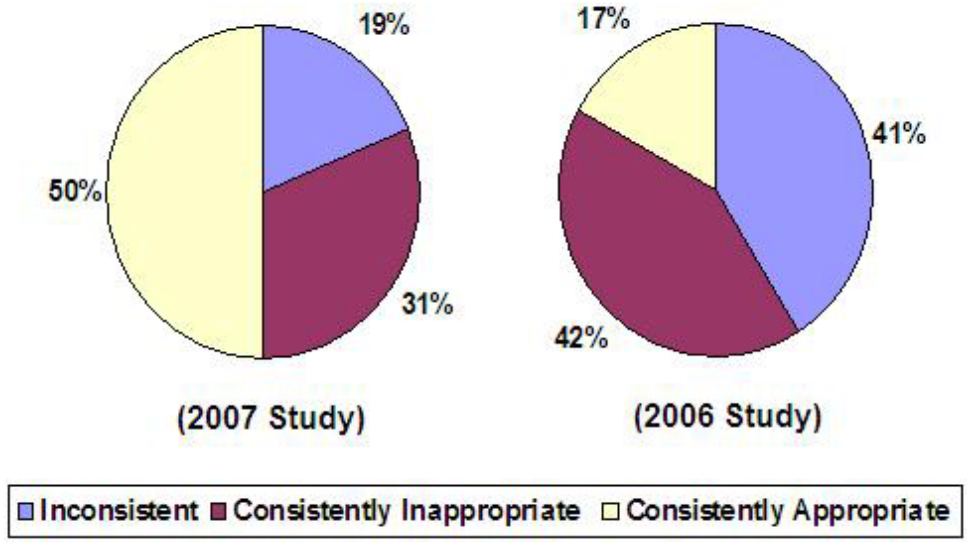


Figure 6-6: The comparison of students' mental models of reference assignment

The results of this test were also compared to the results achieved in the 'Value Assignment' and 'Reference Assignment' experiments in order to see whether or not

the students had made any change to their mental models during the period between finishing the experiment and starting to do this test.

Among the 36 students who participated in the 'Value Assignment' experiment, five students only completed the pre-test but not the post-test. It is difficult to evaluate whether or not they had used the proposed learning tool. So those students were not taken into account. Table 6-8 shows the evolution of the mental models of the remaining 31 students. 15 of them have been found as holding viable mental models of the value assignment concept before the 'value assignment' experiment. All of them still used viable mental models in the mental model test at the end of the course.

The Mental Models Used in the 'Value Assignment' Experiment		**The Mental Models Used in the Test**	**Number of Students**
Pre-Test	**Post-Test**		
Viable	-	Viable	15
Non-Viable	Viable	Viable	7
Non-Viable	Viable	Non-Viable	1
Non-Viable	Non-Viable	Viable	5
Non-Viable	Non-Viable	Non-Viable	3

Table 6-8: Evolution of the mental models of value assignment under the proposed learning model

Eight students held non-viable mental models before the 'value assignment' experiment but changed their mental models to a viable one after using the proposed learning materials. At end of this course, seven of them kept their mental models as viable, but the remaining one changed their mental model to be non-viable.

In addition, there were eight students who held non-viable mental models before the 'value assignment' experiment and did not change their mental models to the viable ones after experiencing the proposed learning materials. At end of this course, five of them changed their non-viable mental models to a viable one (four students changed from the '**Ss**' model, i.e. view statements execution as simultaneous; and one changed from the inconsistent model). The remaining three students kept their non-viable mental models unchanged (two held the '**Ss**' model; and one held the inconsistent model).

Among the 32 students who participated in the 'Reference Assignment' experiment, one student only finished the pre-test but not the post-test. This student was not taken into account during the analysis. Table 6-9 shows the evolution of the mental models of the remaining 31 students. One student who had already held a viable mental model of reference assignment before experiencing the proposed learning materials used consistently appropriate mental models to answer all the reference assignment questions in the test at the end of the course. This student also provided a proper and detailed answer to the open-ended question on the subject.

The Mental Models Used in the 'Reference Assignment' Experiment		**The Mental Models Used in the Test**	**Number of Students**
Pre-Test	**Post-Test**		
Viable	-	Viable	1
Non-Viable	Viable	Viable	2
Non-Viable	Viable	Non-Viable	1
Non-Viable	Non-Viable	Viable	11
Non-Viable	Non-Viable	Non-Viable	16

Table 6-9: Evolution of the mental models of reference assignment under the proposed learning model

Three students held non-viable mental models before the 'reference assignment' experiment but changed their mental models to a viable one after using the proposed learning materials. At end of this course, two of them kept their mental models as viable. The remaining one demonstrated an inconsistent mental model, and explained the reference assignment statement '*b=a*' as "*object b becomes equivalent to the object a*" in the open-ended question.

Twenty-seven students held non-viable mental models before the 'reference assignment' experiment and did not change their mental models to the viable ones after experiencing the proposed learning materials. At end of the course, 11 of them changed their non-viable mental models to be a viable one (apart from two students who did not answer or provided too little information for the open-ended answer, all the other night students demonstrated proper explanations of reference assignment).

The remaining 16 students still used non-viable mental models in this test at the end of the course.

In the first mental models test carried out in the previous academic year (2005-2006), the relationship between the viability of participants' mental models and their performance on course assessment, was investigated. However, this investigation was not repeated in the academic 2006-2007 year's mental models test. The reason was that the number of participants was too small to allow a statistical analysis (e.g. there were only 3 participants in the consistently inappropriate group and 3 participants in the inconsistent group when the participants were separated based on their mental models of value assignment).

6.3.4 Discussion

The results show that a higher percentage of students who experienced the proposed learning materials in 2007 held viable mental models compared to the percentage in 2006. The students from both years had similar backgrounds in programming, and the learning contexts of both years were the same: same instructor, same course structure, and same learning materials. The only difference is that the students in 2007 had the opportunity to use the proposed learning materials. In this case, it could be deduced that the higher percentage of students who held viable mental models this year was caused by the proposed learning materials. However, the results also show that some students could not change their mental models (especially for reference assignment) from non-viable to viable immediately after using the proposed materials. Instead, some of them changed their mental models after a period of time. A possible reason is that some students with limited base knowledge might need more time to construct a viable mental model. Even though they have experienced cognitive conflict, their existing knowledge is not sufficient to support them in solving the conflict. A period of further learning and practice in programming might help them accumulate enough knowledge to solve the conflict and construct a new cognitive balance. However, it should be noticed that the current evidence is not enough to support this assumption. These students might change their mental models due to other reasons. In this case, further long-term, qualitative studies are required

to investigate students' reactions to cognitive conflict events and monitor the subsequent evolution of students' mental models.

On the other hand, there were far fewer students, only two, who changed their mental model from viable to non-viable after a period. A possible reason is that the mental models constructed by them were unstable. However, it is also possible that these students did not answer the questions in the mental model test seriously.

It should be noted that there were some potential weaknesses in this experiment. Firstly, some threats may affect the accuracy of the results obtained from this experiment. For example, as mentioned above, some students who held viable mental models might not answer the questions seriously. In addition, the answer options for the 'reference assignment' questions used in this test are diagram-based. It was possible that some participants might have remembered the patterns that appeared in the visualization tool presented in the 'reference assignment' experiment and answer the questions based on patterns matching, even though this mental model test was conducted more than four months after the 'reference assignment' experiment. Furthermore, this experiment might encourage a student's interest in the topics of value assignment and reference assignment. In this case, they might spend more time and effort learning these topics. Therefore, the improvement of students' mental models might be caused by the students' extra effort rather than the proposed learning model.

6.4 Summary

This chapter presents three studies that were conducted to evaluate the effectiveness of the proposed learning model. In the first two studies, the learning model was implemented to help first year programming students construct viable mental models of value assignment and reference assignment. The results reveal that the learning model was effective for improving students' mental models of those two concepts, although visualization seems less useful to help students construct a viable mental model of a complex concept when the students lack enough base knowledge to

understand the visualization. The third study compared the mental models held by the students in 2006-2007 who experienced the proposed learning model with those held by the students in 2005-2006 who did not experience the learning model. The results show that, in general, the students who experienced the learning model performed better than those who did not experience the learning model.

CHAPTER 7 – Conclusion

This final chapter summarises the research work presented in this thesis and discusses its achievements and limitations along with suggestions for future work.

7.1 Summary of the Work

The research described in this thesis investigated the viability of mental models held by novice programmers, and proposed and evaluated a constructivist-based teaching model that integrates a cognitive conflict strategy and program visualization to improve novice programmers' mental models.

The mental model study (chapter 4) captured the first year programming students' mental models of fundamental programming concepts (focusing on value assignment and reference assignment) using a multiple choice questionnaire and by getting students to describe their understanding using text and diagrams. These results were then used to compare groups based on viable and non-viable models against performance in exams and programming tasks.

To improve the mental models of novice programmers a constructivist-based learning model was proposed (chapter 5). This learning model emphasizes the importance of students' prior knowledge and takes advantage of the potential benefits from the cognitive conflict strategy and program visualization. In addition, a computer-supported learning tool was developed to support the proposed teaching model.

The evaluation of the proposed teaching model consists of three parts (chapter 6). The first part investigated the effectiveness of the proposed teaching model for a relatively straightforward concept, value assignment, with special emphasis on the capacity of the cognitive conflict strategy for engaging students with the

visualization-based learning materials and improving the effectiveness of visualization. In this study, a learning session based on the proposed teaching model and facilitated by the developed computer-supported learning tool, was conducted to help the first year programming students change their mental models to be viable. Pre-tests and post-tests were carried out to elicit the students' mental models of the value assignment concept before and after the learning session. The mental model elicitation method used in the pre-test and post-test was similar to the one employed in the first mental model study (Chapter 4). While the first part of the evaluation focused on a straightforward programming concept, the second part of the evaluation targeted a relatively complex programming concept, reference assignment. This part employed the same research method as the one used in the first part of the evaluation, but the participants were challenged with the reference assignment concept. The last part of the evaluation duplicated the first mental model test conducted in the previous year (Chapter 4) in order to make a comparison between the performance of the students who had experienced the proposed teaching model and those who did not experience the model. In addition, the last part of the evaluation also studied the long-term effects of the proposed teaching model for the evaluation of mental models.

7.2 Research Conclusions

This thesis aims to answer four research questions as proposed in Chapter 1.

- **What is the viability of mental models held by novice programmers?**

The mental model study (Chapter 4) chose two typical and important programming concepts, value assignment and reference assignment, and investigated the viability of novice programmers' mental models of them. The results identified a variety of mental models of value and reference assignment held by first year programming students. Many of these models were seen as non-viable, meaning that they could result in an inappropriate understanding of programming concepts and drive programmers to create improper solutions to programming problems. The quantitative analysis revealed that, at the completion of the first year course, one

third of students still held non-viable mental models of value assignment, with only 17% of students holding viable mental models of reference assignment. This result is of significant concern. Both assignment and reference are key concepts in object-oriented programming. The high failure rates in programming courses are not surprising if students still do not understand these basic programming concepts at the end of courses.

- **Does the viability of the mental models held by novice programmers affect their performance in solving programming problems?**

When the viability of students' mental models was compared to their performance in exams and programming tasks, the results show that the students with viable mental models performed significantly better than those with non-viable mental models. This reveals how important it is for novice programmers to develop viable mental models of key programming concepts.

- **Is a cognitive conflict strategy able to improve the effectiveness and pedagogical benefits of the program visualization technique?**

The first part of the evaluation of the effectiveness of the proposed teaching model, which focused on a related simple programming concept, value assignment, did not reveal the importance of the cognitive conflict strategy to change mental models. In this study, all the participants, no matter whether or not they experienced cognitive conflict event, made changes to their mental models of the assignment process based on the support from programming visualization. One possible explanation for this is that the assignment concept is straightforward. When the visualization along with animation simulates the process of assignment, it is not difficult for participants to realise that they were holding an inappropriate mental model.

The second part of the evaluation investigated the effectiveness of the proposed teaching model using a relatively complex concept, reference assignment. The results show that the participants who experienced the explicit cognitive conflict event

performed better than the others. All the participants who changed their mental model of reference assignment to viable were from the CC+Viz group, i.e. the group that experienced the explicit cognitive conflict event. On the other side, all the participants who did not even realize their original understanding was inappropriate were from the Viz group, i.e. the group that did not experience the explicit cognitive conflict event. In addition, the feedback from the participants revealed that the cognitive conflict event was indeed capable of increasing students' interest in the learning topic, and engaging them in the learning materials.

This suggests that the cognitive conflict strategy can be effective to help improve students' interest and engagement in learning materials, including visualization-based materials, especially when the learning topics are not straightforward.

- **Is a constructivist-based learning model that integrates cognitive conflict and program visualization able to improve novice programmers' mental models of basic programming concepts?**

The evaluation of the effectiveness of the proposed teaching model took place after the participants had learned assignment and the reference concept using traditional learning materials that were delivered in a traditional lecture-based teaching style. However, those traditional learning materials did not help many of the participants construct viable mental models. After following the traditional learning materials, the participants still did not realise that their understanding of the concepts was inappropriate. It is even possible that the participants did not engage, or only engaged superficially, with the traditional learning materials. On the other hand, the proposed teaching model that integrates cognitive conflict and program visualization has been found to be effective to help students change their mental models to be viable. The cognitive conflict strategy appeared to be capable of engaging students with the learning materials and the visualization then helped students construct viable mental models

The visualization technique alone seems less effective when it was applied to a

complex concept. A much smaller portion of participants who constructed a viable mental model of reference assignment concept after using the visualization-based learning materials, compared to those for value assignment. This might be explained by Ben-Ari (2001) who proposed that students had to hold enough base knowledge in order to construct a viable mental model of a new concept. To construct a viable mental model of reference assignment requires students to have more base knowledge, such as that related to object creation and storage, than to build a model of value assignment. So, it may be difficult for the participants to construct a viable mental model of reference assignment without this base knowledge, particularly when some participants even described an object as an array.

In this case, even though the visualization tool simulated the process of object creation and reference assignment, many participants were still unable to understand the visualization and the textual explanations. The result suggests that instructors must ensure the students have built sufficient, appropriate base models of fundamental knowledge before teaching new concepts. In addition, this result also challenges the currently popular object-first teaching paradigm. Object-oriented programming concepts such as class, object, and reference are at a relatively high level of abstraction. Students without the low-level underlying model of the abstraction might have difficulty constructing an appropriate understanding of those concepts. To teach object-oriented programming concepts at an early stage when students have not built the low-level underlying model is obviously risky. In summary, this result might suggest the requirements for instructors and programming educators to rethink current object-oriented programming course structure.

7.3 Limitations and Future Work

The work carried out in this thesis has led to a number of issues that could form the focus of further investigation.

This study has focussed on the mental models held by first year programming students. However, it did not investigate how these mental models originated or how they came to be adopted. It would be of interest to investigate the backgrounds and

prior experiences of participants, to identify the sources and causes of these inappropriate models. This information might help in preventing the initial adoption of non-viable models or in changing them to viable models.

In this research, the studies conducted to assess the effectiveness of the proposed learning model adopted an aggregated approach to the range of models held by participants. The studies did not provide an in-depth exploration of how an individual's mental models evolved, driven by the cognitive conflict strategy and the program visualization technique. In other words, these studies focused more on the question of whether or not the proposed learning model could improve a novice programmer's mental models, as opposed to finding out how an individual's mental models changed under the proposed learning model. Future work would be required to explore this issue. An in-depth, qualitative study that focuses on a smaller number of participants could be carried out to seek a participant's reactions and thoughts to the proposed learning model, and to investigate the underlying mechanisms of how the cognitive conflict strategy combined with program visualization, actually work.

In addition, it is important to conduct a long-term, continuous investigation of the development of mental models. The pedagogical benefits of the cognitive conflict strategy combined with a visualization technique might take some time before becoming effective. Continuous monitoring of the development and evolution of students' mental models over an extended time period might reveal any long-term effects of the proposed approach.

The studies that were used to evaluate the effectiveness of the proposed learning model were conducted under experimental conditions. That means that this learning model was not assessed in a 'real' learning environment. In order to minimize any external inferences on the results, participants were not allowed to communicate with others or to seek assistance from tutors. In addition, they could not use other learning materials and tools to support their learning. Future work would be needed to investigate the performance of the model in a 'real' pedagogical context. The learning materials forming the basis of this study would then need to be seamlessly integrated into the programming course, and students would have to be given the

rights to access any of the learning resources and to access assistance from tutors and other students. Actually, the proposed learning model can also be implemented in a group-based learning style and it might be easier for students to achieve cognitive conflict, and to have a better understanding of the visualization materials, through discussions with each other.

Chapter 3 suggested that the efficiency of the cognitive conflict strategy for science education may be shaped by other effects such as a student's attitudes to conflict events. Meanwhile, the efficiency of the visualization technique may also be influenced by other effects such as a student's learning style (Melis & Monthienvichienchai, 2004; Thomas et al, 2002). It is possible that the proposed learning model brings more benefits to students who demonstrate positive attitudes to conflict events and a visual learning style, as opposed to those who present negative attitudes to this approach. This is an area that requires further investigation to gauge its influence on the effectiveness of the proposed learning model. One approach would be to recruit participants from different backgrounds and group them based on criteria such as learning styles, attitudes, motivational factors, prior knowledge, epistemological beliefs, and reasoning abilities. The participants' reactions to the learning model could then be analyzed and compared.

The utility of the learning model proposed in this thesis is not restricted solely within the domain of programming education. It has the potential to support students in other disciplines and within other computer science topics, such as hardware, databases and computer networks. Future work could investigate the effectiveness of the proposed learning model for any of these topics, investigating whether or not this approach has more general applicability in education and training.

7.4 Conclusion

This thesis presented work aimed at investigating and improving the viability of the mental models held by novice programmers. The work commenced with a study of the mental models held by first year programming students of the fundamental

programming concepts of value and reference assignment. The results of this study revealed that a large number of students held non-viable mental models of these simple concepts. Further, the study showed that students with viable mental models performed significantly better in programming tasks than those with non-viable mental models. This result highlighted the importance of improving a novice programmer's mental models of these fundamental programming concepts. To achieve this aim, a constructivist-based learning model, integrating the cognitive conflict strategy along with a program visualization technique was proposed and adopted. The evaluation of this learning model revealed that this approach was effective in enhancing a student's interest in, and engagement with, the learning materials and helped them to construct viable mental models. However, it was noted that the visualization technique was found to be less effective with a complex concept in situations where a student lacks the necessary base knowledge to interpret the visualization.

REFERENCE

Aebli, H. (1970). Piaget, and beyond. *Journal of Interchange,* 1, pp.12-24.

Barnes, D. J., Fincher, S., & Thompson, S., (1997). Introductory Problem Solving in Computer Science. In Daughton, G. and Magee, P. (eds.) *5th Annual Conference on the Teaching of Computing,* Dublin City University, pp.36–39

Barnes, G. M., & Kind, G. A., (1987). Visual simulations of data structures during lecture. In *Proceedings of the Eighteenth SIGCSE Technical Symposium on Computer Science Education*, St. Louis, Missouri, United States.

Barnes, D., & Kölling, D., (2002), Objects First with Java - A Practical Introduction using BlueJ, Prentice Hall / Pearson Education.

Baser, M., (2006). Fostering Conceptual Change by Cognitive Conflict Based Instruction on Students' Understanding of Heat and Temperature Concepts. *Eurasia Journal of Mathematics, Science and Technology Education, 2*(2), pp. 94-115.

Bayman, P., & Mayer, R. E., (1983). A Diagnosis of Beginning Programmers' Misconceptions of BASIC Programming Statements. *Commun. ACM*, 26(9), pp. 677-679.

Bednar, A.K., Cunningham, D., Duffy, T.M., & Perry, J.D., (1991). Theory into practice: How do we link? In G. Anglin (Ed.), *Instructional Technology: Past, Present and Future*. Englewood, CO: Libraries Unlimited, Inc.

Ben-Ari, M., (2001a). Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching*, 20 (1), 45--73.

Ben-Ari, M., (2001b). Program visualization in theory and practice. *Informatik/*

Informatique, 2, pp. 8–11.

Ben-Bassat Levy, R., Ben-Ari, M. & Uronen, P.A., (2001), An Extended Experiment with Jeliot 2000, *Proceedings of the First Program. Visualization Workshop*, Joensuu, Finland, pp. 131-140.

Besnard, D. & Baxter, G. (2006). Cognitive conflicts in dynamic systems. In D. Besnard, C., Gacek, C. & C. Jones (Eds.) *Structure for Dependability: Computer-based Systems from an Interdisciplinary Perspective.* (pp. 107-126). London, UK: Springer.

Bhattacharya, K. & Han, S., (2001). Piaget and cognitive development. In M. Orey (Ed.), *Emerging perspectives on learning, teaching, and technology*. Retrieved 23th January, 2007, from http://www.coe.uga.edu/epltt/piaget.htm.

Bruce, K., (2004). Controversy on How to Teach CS 1: A Discussion on the SIGCSE-members Mailing List, *SIGCSE Bulletin*, 36(4).

Brusilovsky, P., (1993). Program visualization as a debugging tool for novices. In *INTERACT '93 and CHI '93 Conference Companion on Human Factors in Computing Systems*, Amsterdam, The Netherlands.

Byrne, M. D., Catrambone, R., & Stasko, J. T. (1999). Evaluating animations as student aids in learning computer algorithms. *Comput. Educ.* 33(4), pp. 253-278.

Calloni, B. (1997). Iconic Programming Proves Effective for Teaching the First Year programming Sequence. *Proceedings of the 28th SIGCSE Symposium*, pp. 262-266.

Canas, J. J., Bajo, M. T., & Gonzalvo, P. (1994). *Mental models and computer programming*. Journal of Human-Computer Studies, 40(5), pp. 795-811.

Carlisle, M. C., Wilson, T. A., Humphries, J. W., & Hadfield, S. M., (2005).

RAPTOR: a visual programming environment for teaching algorithmic problem solving. In. *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*. ACM Press, pp. 176-180.

Chan, C., Burtis, J., & Bereiter, C., (1997). Knowledge building as a mediator of conflict in conceptual change. *Cognition and Instruction, 15 (1)*, pp. 1-40.

Chinn, C. A., & Brewer, W. F., (1993). The role of anomalous data in knowledge acquisition: A theoretical framework and implications for science instruction. *Review of Educational Research*, 63, pp. 1-49.

Chinn, C. A, & Brewer, W. F. (1998). An empirical test of a taxonomy of responses to anomalous data in science. *Journal of Research in Science Teaching, 35*, pp. 623-654.

Cohen, M.S., Thompson, B. B., Adelman, L., Bresnick, T. A., Tolcott, M. A., & Freeman, J. T. (1995). *Rapid capturing of battlefield mental models.*. Arlington, VA: Cognitive Technologies, Inc.

Collins, A., & Gentner, D., (1987). How people construct mental models. In D. Holland and N. Quinn, editors, *Cultural Models in Thought and Language*, pp. 243--265. Cambridge University Press, Cambridge, UK.

Craik, K.J.W. (1943). *The Nature of Explanation*. Cambridge UK: Cambridge University Press.

Crews, T., & Ziegler, U., (1998). The Flowchart Interpreter for Introductory Programming Courses" In. Proceedings of *FIE '98 Conference*, Tempe, Arizona, USA, pp. 307-312.

Cross, J.H., Hendrix, T.D., & Barowski, L.A., (2002). Using the debugger as an integral part of teaching CS1. *In 32nd ASEE/IEEE Frontiers in Education*

Conference.

Davis, J., (2001). Conceptual Change. In M. Orey (Ed.), *Emerging perspectives on learning, teaching, and technology*. 19th January, 2007, from: http://www.coe.uga.edu/epltt/conceptualchange.htm.

Dehnadi, S., & Bornat, R., (2006). *The Camel has Two Humps*, Middlesex University Working Paper. Retrieved 17th May, 2006, from http://www.cs.mdx.ac.uk/research/PhDArea/saeed/

Demircioglu, G., Ayas, A., & Demircioglu, H., (2005). Conceptual change achieved through a new teaching program on acids and bases, *Chemistry Education Research and Practice*, 6, 36-51.

Denning, P. J. & McGettrick, (2005). A. Recentering computer science. *Commun. ACM*, 48(11), pp.15–19.

Dittrich, J., van den Bercken, J., Schäfer, T., & Klein, M., (2001), *Data Structures Navigator (DSN)*. Retrieved 28th January, 2007, from http://dbs.mathematik.uni-marburg.de/research/projects/dsn/

Dougiamas, M. (1998). *A journey into Constructivism*. Retrieved 5th January, 2007, from http://dougiamas.com/writing/constructivism.html

Doyle, J. K., Ford, D.N., Radzicki, M.J., & Trees, S.W. (2001). Mental Models of Dynamic Systems *Encyclopedia of Life Support Systems*. EOLSS Publishers. September.

Du, W., (2004), Researches on Mental Models and Elicitation Techniques, *Psychological Science*, 2004, 27(6), pp.1473-1476.

Duncan, R. M. (1995). Piaget and Vygotsky revisited: Dialogue or assimilation? *Developmental Review*, 15, pp. 458-472.

Ernest, P. (1995). The one and the many. In L. Steffe & J. Gale (Eds.). *Constructivism in education* (pp.459-486). New Jersey: Lawrence Erlbaum Associates,Inc.

Eryilmaz, A. (2002). Effects of Conceptual Assignments and Conceptual Change Discussions on Students' Misconceptions and Achievement Regarding Force and Motion. *Journal of Research in Science Teaching*, 39, pp1001-1015.

Flavell, J. H. (1996). Piaget's legacy. *Psychological Science*, 7(4), pp.200-203.

Fleury, A. (1991). Parameter Passing: The Rules the Students Construct. *SIGCSEBuiletin* 23(1), pp.283-286.

Galles, D. (2006), *Data Structure Visualizations (DSV)*, retrieved 28th January, 2007, from http://www.cs.usfca.edu/galles/visualization/.

Gentner, D., & Stevens A., (Ed.). (1983). *Mental Models*. Hillsdale NJ: Lawrence Erlbaum Associates.

Gentner, D. (2002). Mental models, Psychology of. In NJ. Smelser & P. B. Bates (Eds.),. *International Encyclopedia of the Social and Behavioral Sciences* (pp. 9683-9687), Amsterdam: Elsevier Science.

George, C.E. (2000). Experience with novices: The importance of graphical representations in supporting mental models. *Proceedings of the 12th workshop of the Psychology of Programming Interest Group*, pp.33-44.

Good, T. L., Brophy, J. E. (1990). Educational psychology: A realistic approach. (4th ed.).White Plains, NY: Longman

Gorsky, P. & Finegold, M., (1994). The role of anomaly and cognitive dissonance in restructuring students' concepts of force. *Instructional Science*, 22, pp.75-91.

Gotschi, T., Sanders, I. & Galpin, V. (2003), Mental models of recursion, *SIGCSE 2003*, pp. 346-350.

Hamer, J., (2004), A Lightweight Visualizer for Java, *3rd Program Visualization Workshop*, Warwick, UK.

Hand, B. V. & Treagust, D. F. (1988). Application of a conceptual conflict teaching strategy to enhance student learning of acids and bases. *Research in science education*, 18, pp.53 - 63.

Hand, B. & Treagust, D.F., (1988), Application of a conceptual conflict teaching strategy to enhance student learning of acids and bases, *Research in Science Education*, 18, pp. 53-63.

Hannafin, M.J. (1997), The case for grounded learning systems design: What the literature suggests about effective teaching, learning, and technology. *Conference Proceedings, ASCILITE 1997*, Perth, Western Australia, pp. 255-262.

Hewson, P. W. & Hewson, M. G. (1984) The role of conceptual conflict in conceptual change and the design of science instruction, *Instructional Science*, 13, pp.1-13.

Hu, M., (2004). Teaching novices programming with Core Language and Dynamic Visualization. In *Proceedings of the 17th NACCQ.*

Huitt, W., & Hummel, J. (2003). Piaget's theory of cognitive development. *Educational Psychology Interactive*. Valdosta, GA: Valdosta State University.

Retrieved 9th January 2007 from http://chiron.valdosta.edu/whuitt/col/cogsys/piaget.html

Hundhausen, C., Douglas, S., & Stasko, J. (2002). A Meta-Study of Algorithm Visualization Effectiveness, *Journal of Visual Languages and Computing*, 13(3), pp. 259-290.

Hyrskykari, A. (1993). Development of Program Visualization Systems, *2nd Czech. British Symposium of Visual Aspects of Man-Machine*, Systems, Praha,. 1993.

Jarc, D.J., (2005), *Interactive Data Structures Visualizations (IDSV)*, Retrieved 28th January, 2007, from http://nova.umuc.edu/~jarc/idsv.

Johnson-Laird, P.N. (1983). *Mental Models - Towards a Cognitive Science of Language, Inference and Consciousness*. Cambridge MA: Harvard University Press.

Johnson-Laird, P.N., Girotto, V. and Legrenzi, P. (1998). *Mental models: a gentle guide for outsiders*. Retrieved 21st January, 2007, from http://www.si.umich.edu/ICOS/gentleintro.html

Jonassen, D., (1995). Operationalizing mental models: Strategies for assessing mental models to support meaningful learning and design supportive learning environments. In J. Schnase and E. Cunnius (Eds.), *Proceedings of the Computer Supported Collaborative Learning Conference*.

Kahney, H. (1983). What do novice programmers know about recursion? *Proceedings of the CHI '83 Conference on Human Factors in Computer Systems*, pp.235-239. Boston, MA.

Kang, S., Scharmann, L., Noh, T., & Koh, H. (2005). The influence of students; Cognitive and motivational variables in respect of cognitive conflict and conceptual change. *International Journal of Science Education*, 27(9), p.1037.

Karagiorgim, Y., and Symeou, L. (2005), Translating Constructivism into Instructional Design: Potential and Limitations. *Educational Technology & Society*, 8(1), pp.17-27.

Kasmarik, K. & Thurbon, J. (2003). Experimental Evaluation of a Program Visualisation Tool for Use in Computer Science Education. In Proc. *Australian Symposium on Information Visualisation*, (invis.au'03), Adelaide, Australia. *Conferences in Research and Practice in Information Technology*, 24. Pattison, T. and Thomas, B., Eds., ACS. Pp.111-116.

Kohoe, C., Stasko, J., Taylor, A. (1999). Rethinking the Evaluation of Algorithm Animations as Learning Aids: An Observational Study. *Technical Report GIT-GVU-99-10 March 1999.*

Kurland, D. M., & Pea, R. D. (1985), Children's mental models of recursive Logo programs. *Journal of Educational Computing Research*, 1(2), pp.235-243.

Lahtinen, E., Ala-Mutka, K., & Järvinen, H.-M, (2005), A Study of the Difficulties of Novice Programmers. *ITiCSE'05*, Monte de Caparica, Portugal.

Lakoff, G., (1987), Women, fire, and dangerous things, Chicago: University of Chicago Press.

Law, S. Li, N. & Lui, A.K.F. (2006). Cognitive Perturbation through Dynamic Modelling: a. Pedagogical Approach to Conceptual Change in Science. *Journal of Computer Assisted. Learning*, 22(6), pp.403-421.

Limón, M., (2001), On the cognitive conflict as an instructional strategy for conceptual change: a critical appraisal. *Learning and Instruction*, 11, pp.357-380.

Limón, M., & Carretero, M. (1997). Conceptual change and anomalous data: A case.

study in the domain of natural sciences. *European Journal of Psychology of education*, 12 (2), pp.213-230.

Lister, R., Adams, E., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M. McCartney, R., Moström, J. E., Sanders, K., Seppälä, O., Simon, B. & Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers, *SIGCSE Bulletin*, 36(4), pp.119-150 (ITiCSE Working Group Report).

Lui, A. K., Kwan, R., Poon, M., & Cheung, Y. H, (2004), Saving weak programming students: applying constructivism in a first programming course. *SIGCSE Bull.* 36(2), pp.72-76.

Maier, S. (2004), Misconception Research and Piagetian Models of Intelligence. *Oklahoma Higher Education Teaching and Learning Conference*, 2004.

Marcelino, M., Gomes, A., Dimitrov, N. & Mendes, A., (2004), Using a computer-based interactive system for the development of basic algorithmic and programming skills. In Proceedings of *International Conference on Computer Systems and Technologies* (CompSysTech'2004).

Markri, S., (2004), Investigating users' mental models of traditional and digital libraries. unpublished Master Thesis, Retrieved 24th April, 2007, from http://www.uclic.ucl.ac.uk/people/s.makri/stephmscthesis.pdf

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B., Laxer, C., Thomas, L., Utting, I., & Wilusz, T., (2001), A Multi-National, Multi-Institutional Study of Assessment of Programming Skills of First-year CS Students, *SIGCSE Bulletin*, 33(4). pp 125-140.

Melis, E. & Monthienvichienchai, R. (2004). They Call It Learning Style But It's So Much More. In G. Richards (Ed.), *Proceedings of World Conference on E-Learning*

in Corporate, Government, Healthcare, and Higher Education 2004 (pp. 1383-1390). Chesapeake, VA: AACE.

Moher, T. & Wiley, J. (2004). Technology Support for Learning Scientific Control as a Whole Class. Paper presented at *the Annual Conference of the American Education Research Association*.

Moody, J.W., Blanton, J.E. & Augustine, M. (1996). Enhancing End-User Mental Models of Computer Systems Through the Use of Animation, Proceedings of *the Twenty-Ninth Hawaii International Conference of System Sciences*.

Myers, B. A. (1990). Taxonomies of visual programming and program visualization. *Journal of Visual Languages and Computing.* 1(1), pp.97-123.

Naps, T.L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., & Velázquez-Iturbide, J.Á. Exploring the Role of Visualization and Engagement in Computer Science Education. *ACM SIGCSE Bulletin* 35(2), pp.131-152.

Norman, D. A. (1983). Some observations on mental models. In D. Gentner & A. L. Stevens (Eds.), *Mental Models*, Hillsdale, NJ: Lawrence Erlbaum Associates Inc.

Nurrenbern, S.C. (2001). Piaget's Theory of Intellectual Development Revisited. *J. Chem. Educ.*, 78, pp. 1107-1110.

Nussbaum, J. & Novick, S., (1982), Alternative frameworks, conceptual conflict and accommodation: toward a principled teaching strategy. *Instructional Science*, 11, pp. 183-200.

O'Kelly, J., Bergin S., Gaughran P., Dunne S., Ghent J., & Mooney A., (2004), *Initial finding on the impact of an alternative approach to Problem Based Learning in Computer Science*, present at Pleasure By Learning (PBL) conference, Cancun,

Mexico.

Piaget, J. (1952). The origins of intelligence in children. New York: International Universities Press.

Piaget, J. (1977). The development of thought: Equilibration of cognitive structures. New York: Viking Penguin.

Pines, A.L. & West, L.H.T. (1986). Conceptual understanding and science learning: An. interpretation of research within a sources-of-knowledge framework. *Science Education*. 70(5), pp.583-604.

Posner, G. J. (1995). Analyzing the curriculum. New York.

Posner, G. J., Strike, K. A., Hewson, P. W. & Gertzog. W. A. (1982). Accommodation of a scientific conception: Toward a theory of conceptual change. *Science Education*, 66, pp.211-227.

Proulx, J. (2006). Constructivism: A re-equilibration and clarification of the concepts and some potential implications for teaching and pedagogy. *Radical Pedagogy*, Volume 8: Issue 1. Retrieved 03rd January, 2007, from http://radicalpedagogy.icaap.org/content/issue8_1/proulx.html.

Romero, P., du Boulay, B., Cox, R., Lutz, R. & Bryant, S. (2004). Dynamic rich-data capture and analysis of debugging processes. *16th Annual Workshop of the Psychology of Programming Interest Group (PPIG)*, Institute of Technology, Carlow, Ireland.

Sasse, A., (1997). Eliciting and Describing Users' Models of Computer Systems. Ph.D. Thesis, Computer Science, University of Birmingham, UK.

Schwartz, D.L. & Black, J.B., (1996). Analog imagery in mental reasoning: Depictive models. *Cognitive Psychology*. 30, pp.154-219.

Scott, P., Asoko, H., & Driver R., (1992). Teaching for conceptual change: A review of strategies. In R. Duit, F. Goldberg, & H. Niedderer, editors, *Research in Physics Learning: Theoretical Issues and Empirical Studies*, pp.310–329, 1992.

Shaffer, C.A., Cooper, M., & Edwards, S.H., (2007), Algorithm Visualization: A Report on the State of the Field, *38th ACM Technical Symposium on Computer Science Education*. Covington, Kentucky.

Skoumios, M., & Hatzinikita, V., (2005). The Role of Cognitive Conflict in Science Concept Learning. *International Journal of Learning*, 12(7), pp.185-194.

Soloway, E., & Spohrer, J. (1989). Some difficulties of learning to program. In E Soloway & James C Spohrer, editors, *Studying the Novice Programmer*, pp. 283–299. Lawrence Erlbaum Associates, Hillsdale, NJ.

Stasko, J. & Patterson, C. (1992). Understanding and Characterizing Software Visualization Systems. Proceedings of *the 1992 IEEE International Workshop on Visual Languages*, pp.3-10.

Stasko, J., Bradre, A., Lewis, C. (1993). Do Algorithm Animations Assist Learning? An Empirical Study and Analysis. *ACM INTERCHI*, pp.61-66.

Stasko, J.T. & Hundhausen, C.D., (2004). Algorithm Visualization. In S. Fincher & M. Petre (eds.), *Computer Science Education Research* (pp.199-228). Lisse, The Netherlands: Taylor & Francis.

Thomas, L., Ratcliffe, M., Woodbury, J., & Jarman, E., (2002). Learning styles and performance in the introductory programming sequence. *SIGCSE Bull.* 34(1), pp.33-37.

von Glasersfeld, E. (1992). Questions and Answers About Radical Constructivism. In M.K. Pearsall (ed.), *Scope, Sequence, and Coordination of Secondary Schools Science,* Vol. 11, Relevant Research, (pp. 169-182). Washington DC: NSTA.

von Glasersfeld, E. (1984). An introduction to radical constructivism. In P. Watzlawick, *The Invented Reality*, (pp.17-40). New York: W.W. Norton & Company.

von Glasersfeld, E. (1998). Why Constructivism Must Be Radical. pp. 23-28 in *Constructivism and Education*. Larochelle, Marie, Nadine Bednarz, and Jim Garrison (Eds.). Cambridge: Cambridge University Press.

von Glasersfeld, E. (1989). Constructivism. pp. 162-163 in *The International Encyclopedia of Education*, 1 st edition, Supplement Vol.1. Husen, Torsten and T Neville Postlethwaite (Eds.). Oxford, England: Pergamon.

von Glasersfeld, E. (1979). Radical Constructivism and Piaget's Concept of Knowledge. In F. B. Murray *The Impact of Piagetian Theory*. Baltimore, Md.: University Park Press.

Vrasidas, C. (2000). Constructivism versus objectivism: Implication for interaction，course design, and evaluation in distance education. *International Journal of Educational Telecommunications, 6(4),* pp.339-362.

Watts, T., (2004), The SFC Editor - A Graphical Tool for Algorithm Development. *Consortium for computing sciences in colleges.*

Yehezkel, C., Ben-Ari, M., & Dreyfus, T. (2005). Computer architecture and mental models. *SIGCSE Bull.*, 37(1), pp.101–105.

Young, R. M. (1983), Surrogates and Mappings: two Kinds of Conceptual Models for Interactive Devices. In Gentner, D. & Stevens, A. L. [Eds.]: *Mental models*. Hillsdale, NJ: Erlbaum.

Ziegler, U., & Crews, T., (1999), An Integrated Program Development Tool for Teaching and Learning How to Program. Proceedings of the *30th SIGCSE Symposium*.

Appendix A – The Questionnaire for Investigating Novice Programmers' Mental Models

Investigation into Mental Models of Programmers

My name is Linxiao Ma and as part of my PhD research I am investigating the ways in which people learn to program. This test is designed to explore the kind of mental models that programmers create when problem-solving. I am very grateful for your assistance in completing this study.

Instructions

1. **This test questionnaire is divided into three parts: part1, part2, part3. Please finish each part in order, and please do not go back to previous part after you have finished it.**
2. **Please complete the test as quickly as possible and not refer to any other information (books, web pages) or execute the statements on a computer.**
3. **Please return the paper to me when you have finished.**

Please fill your personal information:

Name:	
Registration No:	
Is this your first course in programming? **Did you learn any other programming language before? If so, please list the languages here.**	

The questionnaire results will be recorded in a database. They will never be revealed to any person who could in any way identify you from the data given above. They will never be used for assessment purposes.

I consent to the research use of my questionnaire ……

Participant's signature

A-1: The Close-ended Questions for Value Assignment

[* This part is from Dehnadi & Bornat (2006)]

1. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int  a = 10;`
`int  b = 20;`
 	
`a = b;`
	The new values of a and b are:
	
	☐ `a = 20  b =  0`
	☐ `a = 20  b = 20`
	☐ `a = 0  b = 10`
	☐ `a = 10  b = 10`
	☐ `a = 30  b = 20`
	☐ `a = 30  b =  0`
	☐ `a = 10  b = 30`
	☐ `a = 0  b = 30`
	☐ `a = 10  b = 20`
	☐ `a = 20  b = 10`
	
	Any other values for a and b:
	
	`a =  b =`
	`a =  b =`
	`a =  b =`
		Use this column for your rough notes please

2. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int  a = 10;`
`int  b = 20;`
 	
`b = a;`
	The new values of a and b are:
	
	☐ `a = 20  b = 0`
	☐ `a = 20  b = 20`
	☐ `a = 10  b = 30`
	☐ `a = 0  b = 30`
	☐ `a = 30  b = 10`
	☐ `a = 30  b = 0`
	☐ `a = 10  b = 20`
	☐ `a = 20  b = 10`
	☐ `a = 0  b = 10`
	☐ `a = 10  b = 10`
	
	Any other values for a and b:
	
	`a =  b =`
	`a =  b =`
	`a =  b =`	

3. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int big = 10;`
`int small = 20;`
 	
`big = small;`
	The new values of big and small are:
	
	☐ `big = 20 small = 0`
	☐ `big = 10 small = 20`
	☐ `big = 20 small = 10`
	☐ `big = 20 small = 20`
	☐ `big = 0 small = 10`
	☐ `big = 10 small = 10`
	☐ `big = 30 small = 20`
	☐ `big = 30 small = 0`
	☐ `big = 10 small = 30`
	☐ `big = 0 small = 30`
	
	Any other values for big and small :
	
	`big = small =`
	`big = small =`
	`big = small =`
		Use this column for your rough notes please

4. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int a = 10;`
`int b = 20;`
 	
`a = b;`
`b = a;`
	The new values of a and b are:
	
	☐ `a = 0 b = 30`
	☐ `a = 40 b = 30`
	☐ `a = 30 b = 0`
	☐ `a = 0 b = 20`
	☐ `a = 20 b = 20`
	☐ `a = 10 b = 0`
	☐ `a = 10 b = 10`
	☐ `a = 30 b = 50`
	☐ `a = 10 b = 20`
	☐ `a = 20 b = 10`
	☐ `a = 30 b = 30`
	
	Any other values for a and b:
	`a = b =`
	`a = b =`
	`a = b =`	

5. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int a = 10;`
`int b = 20;`
 	
`b = a;`
`a = b;`
	The new values of a and b are:
	
	☐ `a = 0 b = 30`
	☐ `a = 10 b = 20`
	☐ `a = 10 b = 20`
	☐ `a = 20 b = 10`
	☐ `a = 10 b = 0`
	☐ `a = 10 b = 10`
	☐ `a = 0 b = 20`
	☐ `a = 20 b = 20`
	☐ `a = 40 b = 30`
	☐ `a = 30 b = 0`
	☐ `a = 30 b = 50`
	☐ `a = 30 b = 30`
	
	Any other values for a and b:
	`a = b =`
	`a = b =`
	`a = b =`	

6. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int a = 10;`
`int b = 20;`
`int c = 30;`
 	
`a = b;`
`b = c;`
	The new values of a and b and c are:
	
	☐ `a = 30 b = 50 c = 0`
	☐ `a = 30 b = 30 c = 50`
	☐ `a = 0 b = 30 c = 50`
	☐ `a = 20 b = 30 c = 0`
	☐ `a = 20 b = 30 c = 30`
	☐ `a = 0 b = 10 c = 0`
	☐ `a = 10 b = 10 c = 10`
	☐ `a = 60 b = 20 c = 30`
	☐ `a = 60 b = 0 c = 0`
	☐ `a = 10 b = 30 c = 40`
	☐ `a = 0 b = 0 c = 60`
	☐ `a = 10 b = 20 c = 30`
	☐ `a = 20 b = 30 c = 20`
	☐ `a = 20 b = 20 c = 20`
	☐ `a = 30 b = 30 c = 30`
	☐ `a = 0 b = 10 c = 20`
	☐ `a = 10 b = 10 c = 20`
	☐ `a = 30 b = 50 c = 30`
	
	Any other values for a and b and c :
	
	`a = b = c =`
	`a = b = c =`
	`a = b = c =`
		Use this column for your rough notes please

7. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int a = 5;`
`int b = 3;`
`int c = 7;`
 	
`a = c;`
`b = a;`
`c = b;`
	The new values of a and b and c are:
	
	☐ `a = 3 b = 5 c = 5`
	☐ `a = 12 b = 14 c = 22`
	☐ `a = 0 b = 0 c = 15`
	☐ `a = 8 b = 15 c = 12`
	☐ `a = 3 b = 12 c = 0`
	☐ `a = 5 b = 3 c = 7`
	☐ `a = 3 b = 5 c = 7`
	☐ `a = 5 b = 5 c = 5`
	☐ `a = 3 b = 3 c = 3`
	☐ `a = 7 b = 5 c = 3`
	☐ `a = 3 b = 7 c = 5`
	☐ `a = 12 b = 8 c = 10`
	☐ `a = 10 b = 8 c = 12`
	☐ `a = 0 b = 0 c = 7`
	☐ `a = 7 b = 7 c = 7`
	☐ `a = 3 b = 5 c = 5`
	
	Any other values for a and b and c :
	
	`a = b = c =`
	`a = b = c =`
	`a = b = c =`	

8. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int a = 5;`
`int b = 3;`
`int c = 7;`
 	
`c = b;`
`b = a;`
`a = c;`
	The new values of a and b and c are:
	
	☐ `a = 15 b = 8 c = 10`
	☐ `a = 10 b = 5 c = 0`
	☐ `a = 15 b = 10 c = 22`
	☐ `a = 0 b = 0 c = 15`
	☐ `a = 5 b = 3 c = 7`
	☐ `a = 3 b = 5 c = 7`
	☐ `a = 5 b = 5 c = 5`
	☐ `a = 3 b = 3 c = 3`
	☐ `a = 7 b = 5 c = 3`
	☐ `a = 3 b = 7 c = 5`
	☐ `a = 12 b = 8 c = 10`
	☐ `a = 8 b = 10 c = 12`
	☐ `a = 3 b = 5 c = 0`
	☐ `a = 3 b = 5 c = 3`
	☐ `a = 0 b = 0 c = 7`
	☐ `a = 7 b = 7 c = 7`
	
	Any other values for a and b and c :
	
	`a = b = c =`
	`a = b = c =`
	`a = b = c =`
		Use this column for your rough notes please

9. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int a = 5;`
`int b = 3;`
`int c = 7;`
 	
`c = b;`
`a = c;`
`b = a;`
	The new values of a and b and c are:
	
	☐ `a = 7 b = 0 c = 5`
	☐ `a = 7 b = 7 c = 5`
	☐ `a = 15 b = 18 c = 10`
	☐ `a = 0 b = 15 c = 0`
	☐ `a = 15 b = 10 c = 12`
	☐ `a = 10 b = 0 c = 5`
	☐ `a = 5 b = 3 c = 7`
	☐ `a = 7 b = 3 c = 5`
	☐ `a = 5 b = 5 c = 5`
	☐ `a = 7 b = 7 c = 7`
	☐ `a = 7 b = 5 c = 3`
	☐ `a = 3 b = 7 c = 5`
	☐ `a = 12 b = 8 c = 10`
	☐ `a = 8 b = 10 c = 12`
	☐ `a = 0 b = 3 c = 0`
	☐ `a = 3 b = 3 c = 3`
	
	Any other values for a and b and c :
	
	`a = b = c =`
	`a = b = c =`
	`a = b = c =`	

10. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int a = 5;`
`int b = 3;`
`int c = 7;`
 	
`b = a;`
`c = b;`
`a = c;`
	The new values of a and b and c are:
	
	☐ `a = 3 b = 7 c = 3`
	☐ `a = 20 b = 8 c = 15`
	☐ `a = 15 b = 0 c = 0`
	☐ `a = 8 b = 10 c = 15`
	☐ `a = 0 b = 7 c = 8`
	☐ `a = 5 b = 3 c = 7`
	☐ `a = 5 b = 7 c = 3`
	☐ `a = 3 b = 3 c = 3`
	☐ `a = 7 b = 7 c = 7`
	☐ `a = 7 b = 5 c = 3`
	☐ `a = 3 b = 7 c = 5`
	☐ `a = 12 b = 8 c = 10`
	☐ `a = 8 b = 10 c = 12`
	☐ `a = 5 b = 0 c = 0`
	☐ `a = 5 b = 5 c = 5`
	☐ `a = 0 b = 7 c = 3`
	
	Any other values for a and b and c :
	
	`a = b = c =`
	`a = b = c =`
	`a = b = c =`
		Use this column for your rough notes please

11. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int a = 5;`
`int b = 3;`
`int c = 7;`
 	
`b = a;`
`a = c;`
`c = b;`
	The new values of a and b and c are:
	
	☐ `a = 12 b = 8 c = 15`
	☐ `a = 7 b = 0 c = 8`
	☐ `a = 8 b = 18 c = 15`
	☐ `a = 0 b = 15 c = 0`
	☐ `a = 5 b = 3 c = 7`
	☐ `a = 7 b = 3 c = 5`
	☐ `a = 5 b = 5 c = 5`
	☐ `a = 7 b = 7 c = 7`
	☐ `a = 7 b = 5 c = 3`
	☐ `a = 3 b = 7 c = 5`
	☐ `a = 12 b = 8 c = 10`
	☐ `a = 8 b = 10 c = 12`
	☐ `a = 7 b = 0 c = 5`
	☐ `a = 7 b = 5 c = 5`
	☐ `a = 0 b = 3 c = 0`
	☐ `a = 3 b = 3 c = 3`
	
	Any other values for a and b and c :
	
	`a = b = c =`
	`a = b = c =`
	`a = b = c =`	

12. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int a = 5;`
`int b = 3;`
`int c = 7;`
 	
`a = c;`
`c = b;`
`b = a;`
	The new values of a and b and c are:
	
	☐ `a = 20 b = 15 c = 12`
	☐ `a = 15 b = 0 c = 0`
	☐ `a = 5 b = 3 c = 7`
	☐ `a = 5 b = 7 c = 3`
	☐ `a = 3 b = 3 c = 3`
	☐ `a = 7 b = 7 c = 7`
	☐ `a = 7 b = 5 c = 3`
	☐ `a = 3 b = 7 c = 5`
	☐ `a = 12 b = 8 c = 10`
	☐ `a = 8 b = 10 c = 12`
	☐ `a = 0 b = 7 c = 3`
	☐ `a = 7 b = 7 c = 3`
	☐ `a = 5 b = 0 c = 0`
	☐ `a = 5 b = 5 c = 5`
	☐ `a = 12 b = 15 c = 10`
	☐ `a = 0 b = 12 c = 3`
	
	Any other values for a and b and c :
	
	`a = b = c =`
	`a = b = c =`
	`a = b = c =`
		Use this column for your rough notes please

A-2: The Open-ended Question

[* This part was an original contribution of this thesis]

Please trace the following statements. Describe what happens when each of the statements is executed. You may use both text and diagrams in your answer.

```
Person a, b;
a = new Person ("Jack");
b= new Person (“Tom”);
b = a;
```

A-3: The Close-ended Questions for Reference Concept

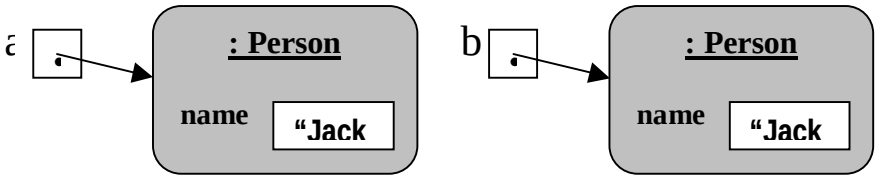
[* This part was an original contribution of this thesis]

Given that a Person class has been defined that holds a name field as a String, which of the following diagrams most closely represents the state of memory after all the statements have been executed? Please circle the letter of the correct answer:

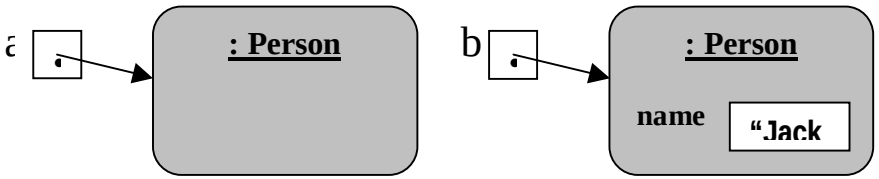
Question 1:

```
Person a, b;
a = new Person ("Jack");
b = a;
```

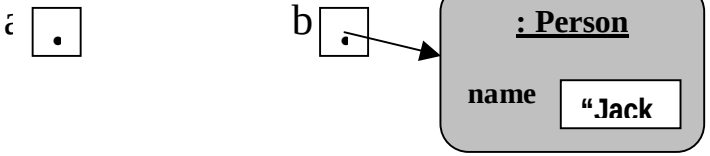
A)



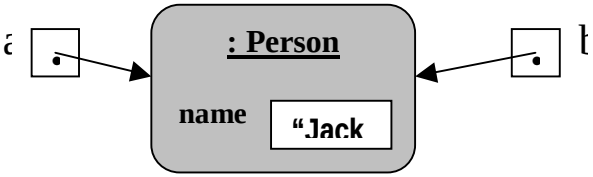
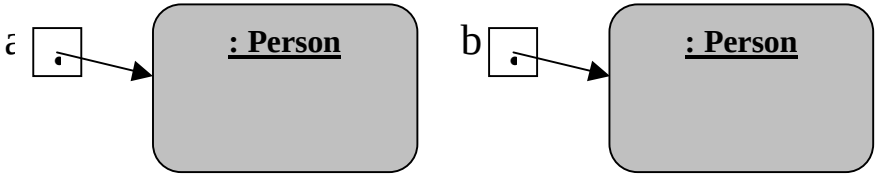
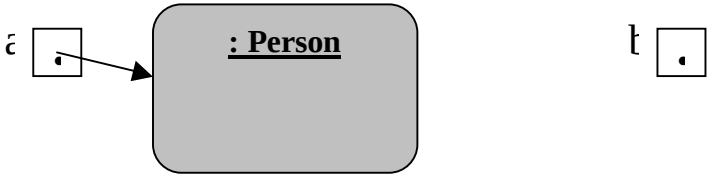
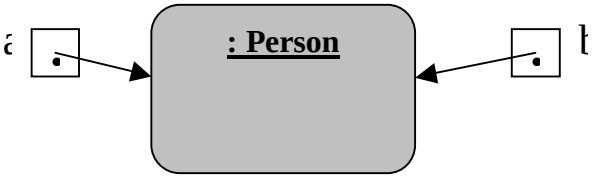
B)

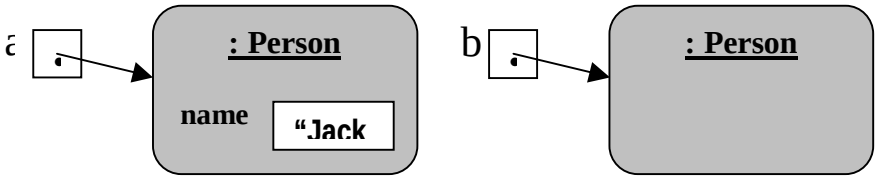
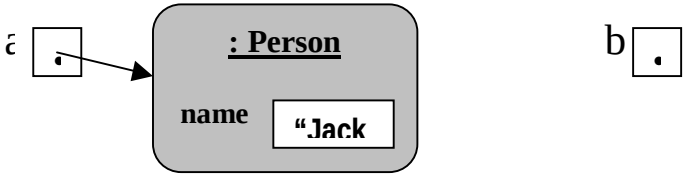


C)



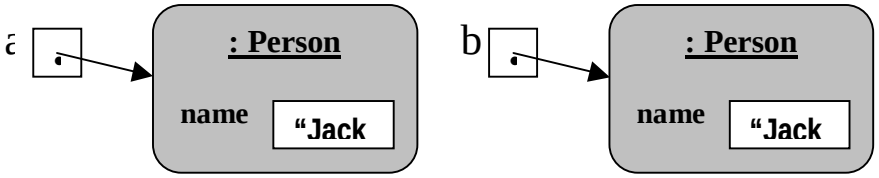
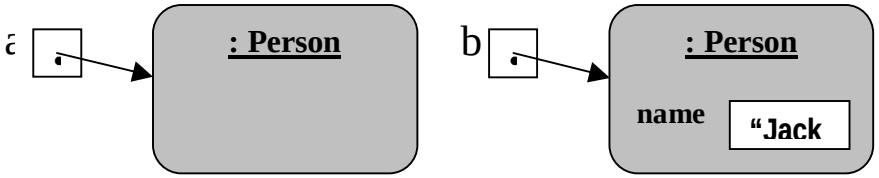
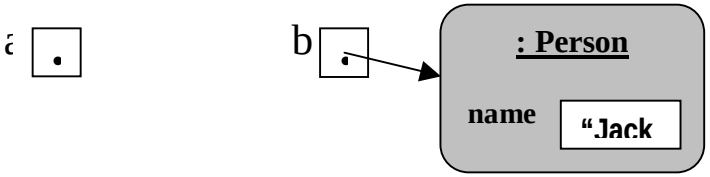
Use this column for your rough notes please

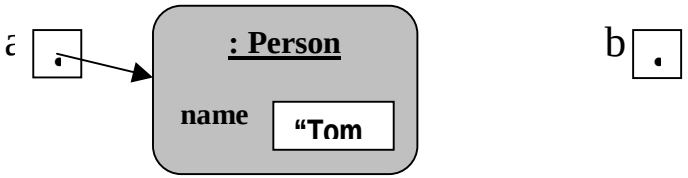
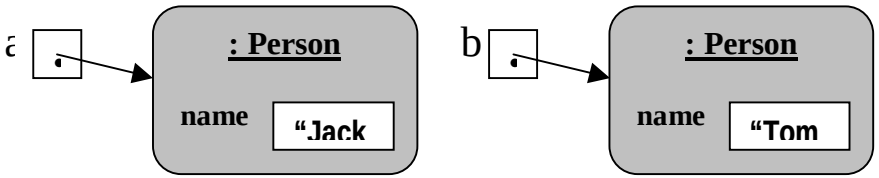
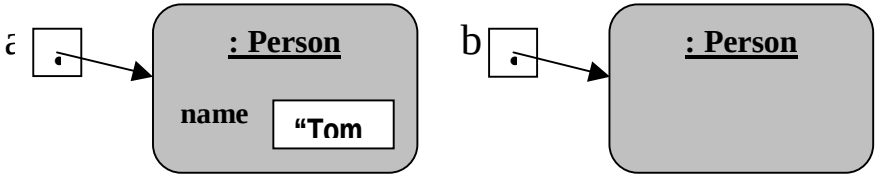
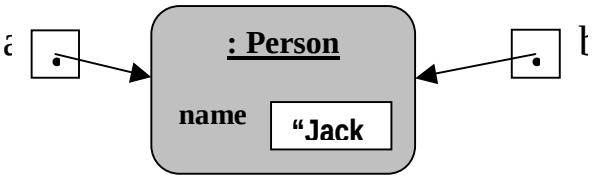
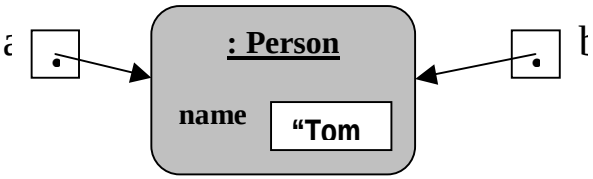
D)	 a : Person name "Jack b	
E)	 a : Person b : Person	
F)	 a : Person b	
G)	 a b	
H)	 a : Person b	

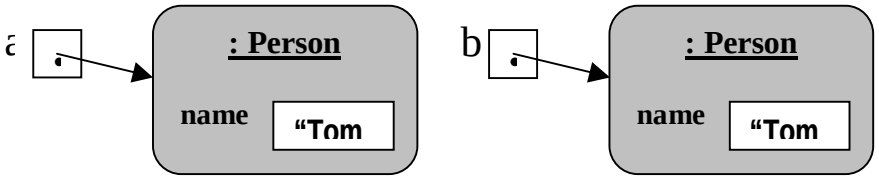
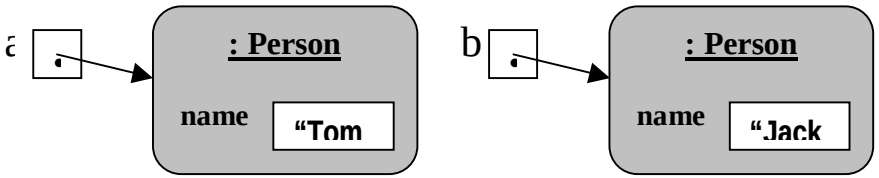
J)		
K)		
If the answer is not list above, please give your answer here:		

Question 2:

```
Person a, b;
a = new Person ("Jack");
b = new Person (“Tom”);
b = a;
```

A)		**Use this column for your rough notes please**
B)		
C)		

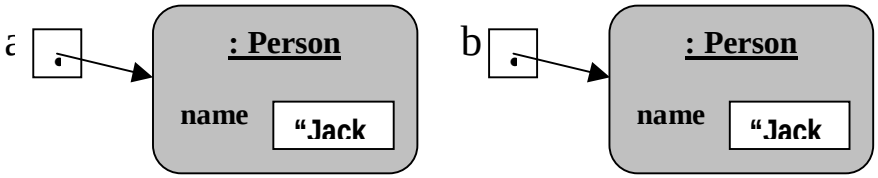
D)	 a : Person name “Tom b	
E)	 a : Person name “Jack b : Person name “Tom	
F)	 a : Person name “Tom b : Person	
G)	 a : Person name “Jack b	
H)	 a : Person name “Tom b	

I)		
J)		
If the answer is not list above, please give your answer here:		

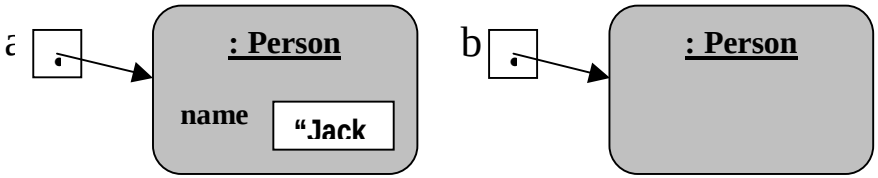
Question 3:

```
Person a, b;
a = new Person ("Jack");
b = new Person (“Tom”);
b = a;
a = b;
```

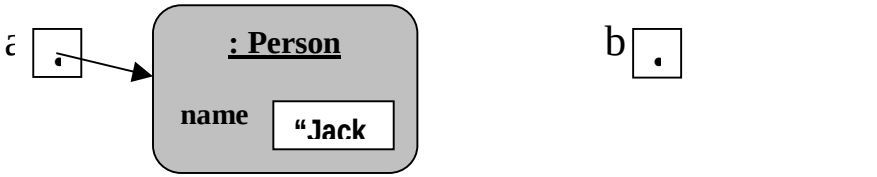
A)



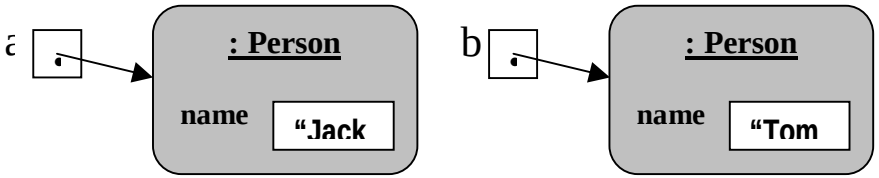
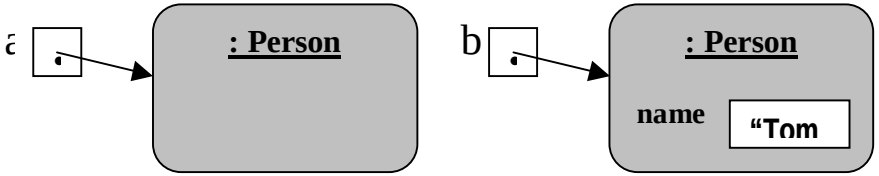
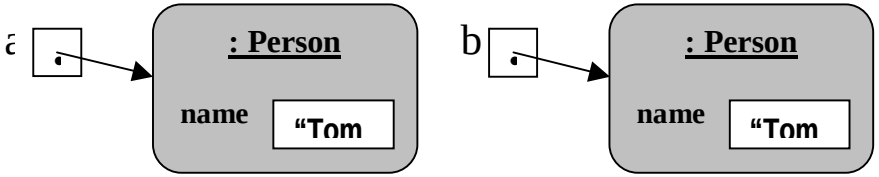
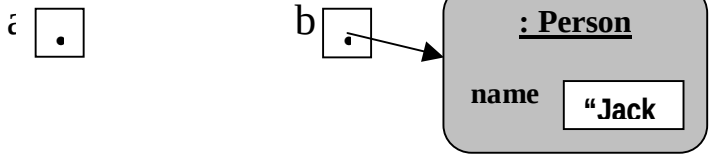
B)

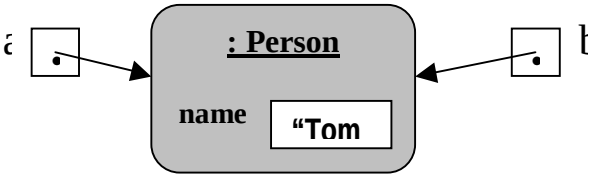
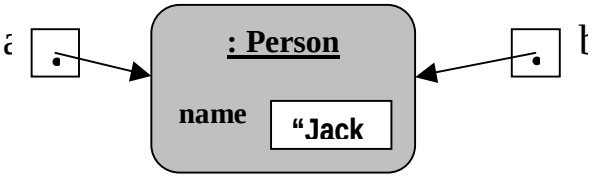


C)



Use this column for your rough notes please

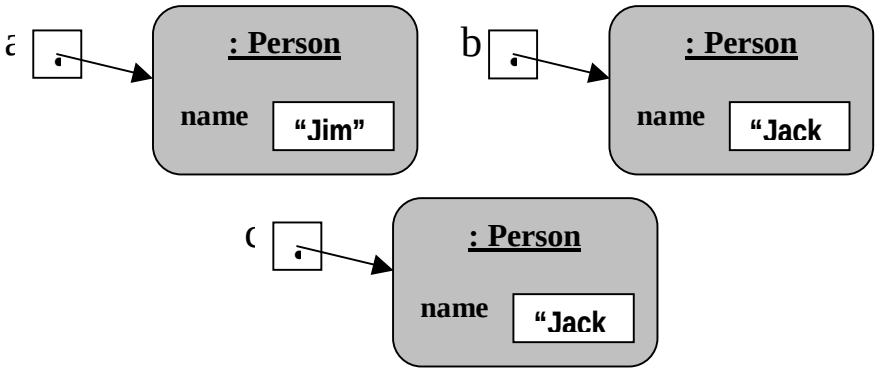
D)	 a : Person name "Jack b : Person name "Tom	
E)	 a : Person b : Person name "Tom	
F)	 a : Person name "Tom b : Person name "Tom	
G)	 a b : Person name "Jack	

H)		
I)		
If the answer is not list above, please give your answer here:		

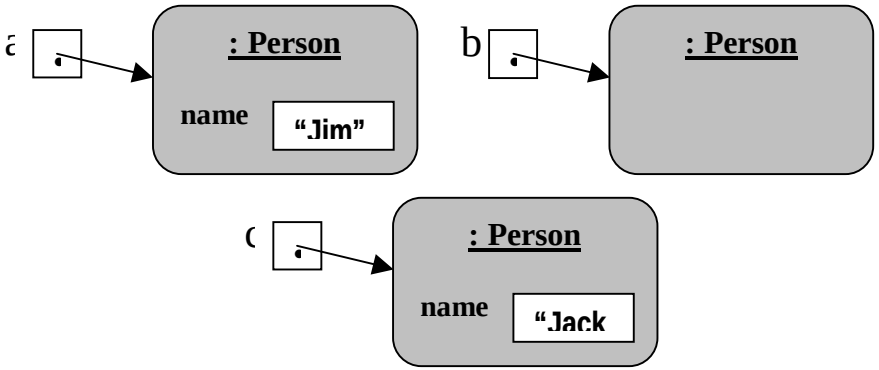
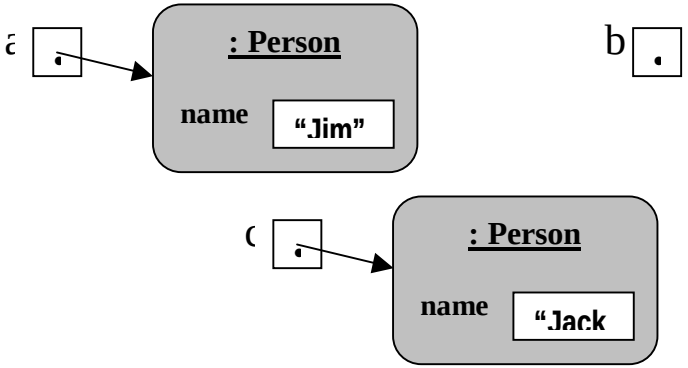
Question 4:

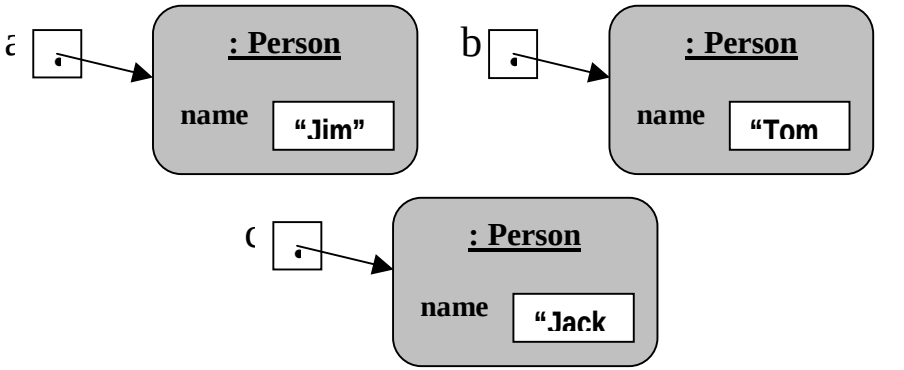
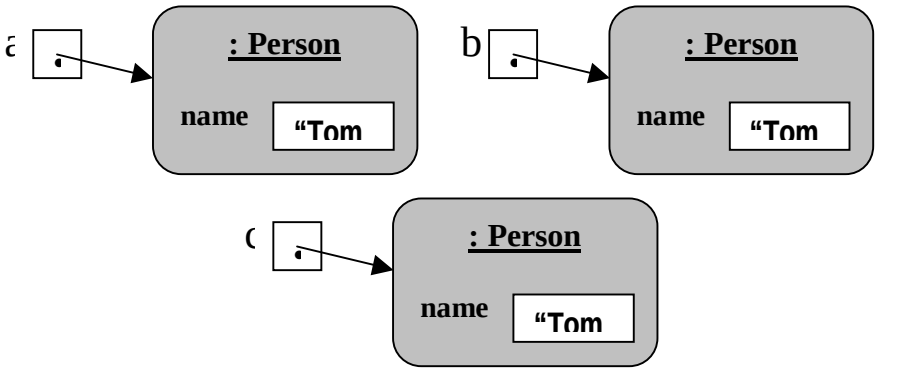
```
Person a, b, c;
a = new Person ("Jack");
b = new Person (“Tom”);
c = new Person (“Jim”);
b = a;
a = c;
c = b;
```

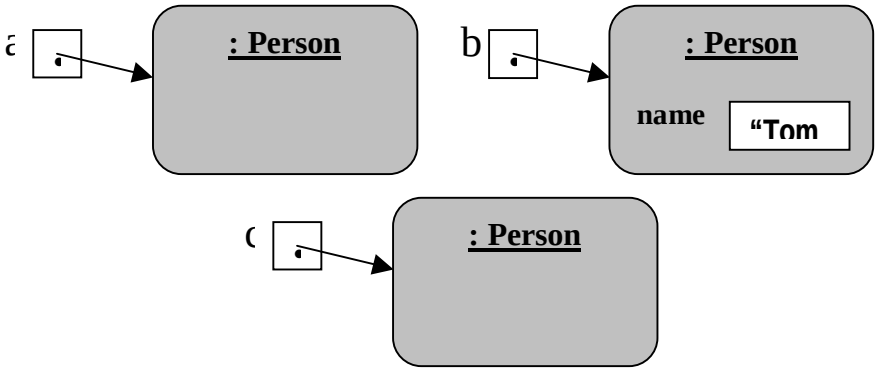
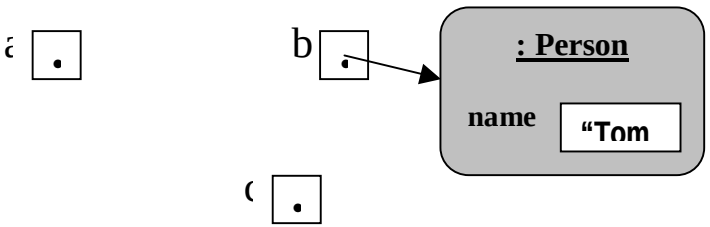
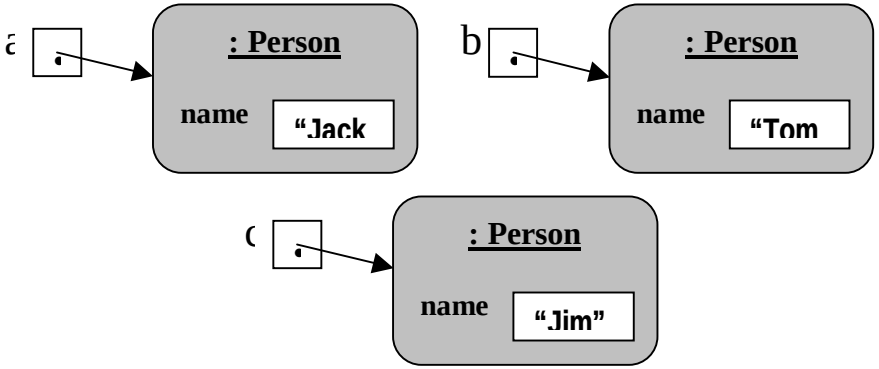
A)

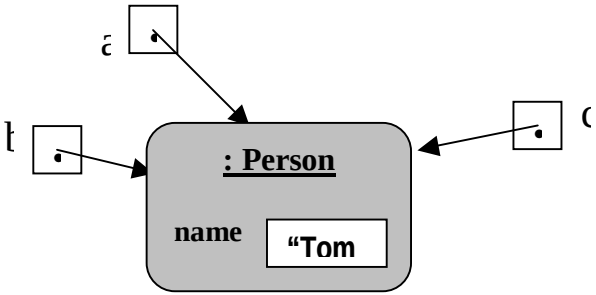
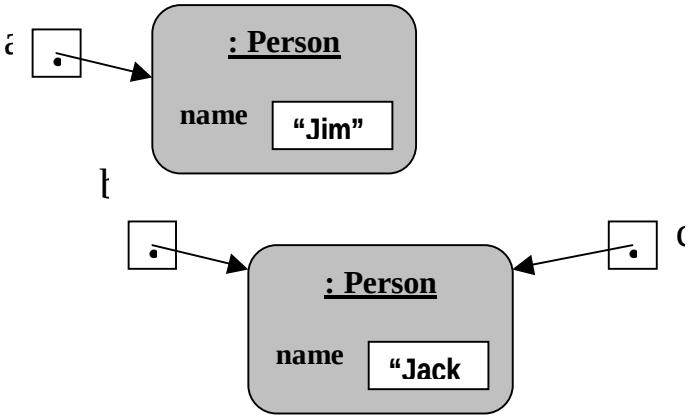


Use this column for your rough notes please

B)	 a : Person name "Jim" b : Person c : Person name "Jack	
C)	 a : Person name "Jim" b c : Person name "Jack	

D)	 a : Person name "Jim" b : Person name "Tom c : Person name "Jack	
E)	 a : Person name "Tom b : Person name "Tom c : Person name "Tom	

F)	 a : Person b : Person name “Tom c : Person	
G)	 a b : Person name “Tom c	
H)	 a : Person name “Jack b : Person name “Tom c : Person name “Jim”	

I)		
J)		
If the answer is not list above, please give your answer here:		

Appendix B – The Questionnaire for Investigating the Effectiveness of the Proposed Learning Model for Value Assignment Concept

B-1: Questionnaire for Pre-Test

Question 1:

Please examine the following Java program and then answer the question:

```
int a = 30;
int b = 60;
a = b;
```

If you think the program is correct, please input the result:

a = _____ b = _____

or please explain why the program is incorrect:

Question 2:

Please examine the following Java program and then answer the question:

```
int a = 30;
int b = 60;
b = a;
```

If you think the program is correct, please input the result:

a = _____ b = _____

or please explain why the program is incorrect:

Question 3:

Please examine the following Java program and then answer the question:

```
int a = 30;
int b = 60;
a = b;
b = a;
```

If you think the program is correct, please input the result:

a = _____ b = _____

or please explain why the program is incorrect:

Question 4:

Please examine the following Java program and then answer the question:

```
int a = 30;
int b = 60;
int c = 90;
a = b;
b = c;
```

If you think the program is correct, please input the result:

a = _____ b = _____ c = _____

or please explain why the program is incorrect:

Question 5:

Please examine the following Java program and then answer the question:

```
int a = 15;
int b = 9;
int c = 21;
a = c;
b = a;
c = b;
```

If you think the program is correct, please input the result:

a = _____ b = _____ c = _____

or please explain why the program is incorrect:

Question 6:

Please examine the following Java program and then answer the question:

```
int a = 15;
int b = 9;
int c = 21;
c = b;
b = a;
a = c;
```

If you think the program is correct, please input the result:

a = _____ b = _____ c = _____

or please explain why the program is incorrect:

B-2: Questionnaire for Post-Test

Question 1:

Please examine the following Java program and then answer the question:

```
int a = 20;
int b = 40;
a = b;
```

If you think the program is correct, please input the result:

a = _____ b = _____

or please explain why the program is incorrect:

Question 2:

Please examine the following Java program and then answer the question:

```
int a = 20;
int b = 40;
b = a;
```

If you think the program is correct, please input the result:

a = _____ b = _____

or please explain why the program is incorrect:

Question 3:

Please examine the following Java program and then answer the question:

```
int a = 20;
int b = 40;
a = b;
b = a;
```

If you think the program is correct, please input the result:

a = _____ b = _____

or please explain why the program is incorrect:

Question 4:

Please examine the following Java program and then answer the question:

```
int a = 20;
int b = 40;
int c = 60;
a = b;
b = c;
```

If you think the program is correct, please input the result:

a = _____ b = _____ c = _____

or please explain why the program is incorrect:

Question 5:

Please examine the following Java program and then answer the question:

```
int a = 10;
int b = 6;
int c = 14;
a = c;
b = a;
c = b;
```

If you think the program is correct, please input the result:

a = _____ b = _____ c = _____

or please explain why the program is incorrect:

Question 6:

Please examine the following Java program and then answer the question:

```
int a = 10;
int b = 6;
int c = 14;
c = b;
b = a;
a = c;
```

If you think the program is correct, please input the result:

a = _____ b = _____ c = _____

or please explain why the program is incorrect:

Question 7:

Please examine the following Java program and then answer the question:

```
int a = 20;
int b = 40;
a = b;
```

Please describe in the box below what happens when the 1st statement "*int a = 20;*" is executed:

Please describe in the box below what happens when the 3rd statement " *a = b* " is executed:

B-3: Qualitative Questionnaire

Please answer the following questions:

1. **Has your understanding of "variable" and "assignment" concepts changed during this learning session? If "yes", please describe what the changes were:**

(If you answered 'No' for Question 1, please ignore questions 2 - 4 and directly jump to Question 5)

2. **What event or effects made you realize that your original understanding was incorrect?**

3. **Please state how strongly did you feel about the following statements:**

   a. When I found my original understanding was incorrect, I was surprised and found it difficult to believe.

   - ☐ Strongly disagree
   - ☐ Disagree
   - ☐ Neither agree nor disagree
   - ☐ Agree
   - ☐ Strongly agree

   b. When I found my original understanding was incorrect, I was curious about it and wanted to find the reason why.

   - ☉ Strongly disagree
   - ☐ Disagree
   - ☐ Neither agree nor disagree
   - ☐ Agree
   - ☐ Strongly agree

c. When I found my original understanding was incorrect, I felt upset.

- ◉ Strongly disagree
- ○ Disagree
- ○ Neither agree nor disagree
- ○ Agree
- ○ Strongly agree

4. At what stage during this learning session did you develop a new understanding of the programming concepts? Was there anything in particular that helped you construct this new understanding?

5. What features and content of the learning materials presented in this learning session did you find beneficial to your understandings of programming concepts? How did it (they) help your learning experience?

6. Do you still have any questions or still feel confused about the programming concepts "variable" and "assignment"? If so, please provide details.

Appendix C - The Questionnaire for Investigating the Effectiveness of the Proposed Learning Model for Value Assignment Concept

C-1: Questionnaire for Pre-Test

A *Staff* class has been defined that holds an *ID* field as an integer. Constructors like '*new Staff(1)* ' are used to create an object of *Staff* class with an ID of '*1* '.

Methods in the form '*changeID(5)* ' are used to change the value of the *ID* field of an object to be '*5* '.

The following questions require you to determine the value the *ID* field of the objects after the execution of the program fragment.

Question 1:

```
Staff a;
Staff b;
a = new Staff (1);
b = new Staff (2);
a = b;
b.changeID (5);
```

a.ID = _____; b.ID = _____;

Question 2:

```
Staff a;
Staff b;
a = new Staff (1);
b = new Staff (2);
a = b;
b = a;
b.changeID (5);
```

a.ID = _____; b.ID = _____;

Question 3:

```
Staff a;
Staff b;
Staff c;
a = new Staff (1);
b = new Staff (2);
c = new Staff (3);
a = b;
b = c;
b.changeID(5);
```

a.ID = _____; b.ID = _____; c.ID = _____;

Please describe what happens when the statement "a = new Staff(1);" was executed:

Please describe what happens when the statement "a = b;" was executed:

C-2: Questionnaire for Post-Test

An *Account* class has been defined that holds a *balance* field as an integer. Constructors like '*new Account(100)* ' are used to create an object of *Account* class with a balance of '*100* '.

Methods of the form '*changeBalance(750)* ' are used to change the value of the *balance* field of an object to be '*750* '.

The following questions require you to determine a value for the *balance* field of the objects after the execution of the program fragment.

Question 1:

```
Account a;
Account b;
a = new Account (100);
b = new Account (200);
a = b;
b.changeBalance(750);
```

a.balance = _____; b.balance = _____;

Question 2:

```
Account acc1;
Account acc2;
acc1 = new Account (100);
acc2 = new Account (200);
acc1 = acc2;
acc2 = acc1;
acc2.changeBalance(750);
```

acc1.balance = _____; acc2.balance = _____;

Question 3:

```
Account acc1;
Account acc2;
Account acc3;
acc1 = new Account (100);
acc2 = new Account (200);
acc3 = new Account (300);
acc1 = acc2;
acc2 = acc3;
acc2.changeBalance(750);
```

acc1.balance = _____; acc2.balance = _____; acc3.balance = _____;

Please describe what happens when the statement "acc1 = new Account (100);" is executed:

Please describe what happens when the statement "acc1 = acc2" is executed:

C-3: The Question to Trigger Cognitive Conflict

A*Student* class has been defined that holds an *name* field as a String. Constructors like '*new Student("Ben")* ' are used to create an object of *Student* class with a name of 'Ben'.

Methods in the form '*changeName("Tom")* ' are used to change the value of the *name* field of an object to be *"Tom"*.

The following questions require you to predict the value the *name* field of the objects after the execution of the program.

```
Student a;
Student b;
a = new Student (“Ben”);
b = new Student (“Ross”);
a = b;
b.changeName(“Tom”);
```

a.name = _____; b.name = _____;

C-4: Feedback Questionnaire

Please answer the following questions:

1. **Has this exercise resulted in any changes to your understanding of any programming concept ? If your answer is 'yes', please describe what these changes were.**

2. **When did you realize that your original understanding was incorrect?**

   ☐ When I was answering the questions before using the visualization tool

   ☐ When I was using the visualization tool

   ☐ When I was answering the question after using the visualization tool

   ☐ Other (Please explain)

3. **when you were viewing the visualization tool, how much attention did you pay to the accompanying textual explanations at the bottom of the window? Please choose the most suitable description from the following list.**

☐ I did not pay any attention to the textual explanations.

☐ I just had a quick look at the textual explanations but did not make any attempt to understand them.

☐ I paid attention to some of the textual explanations and spent mental effort to understand them.

☐ I paid attention to all of the information covered by the textual explanations and spent mental effort to understand it.

And how much did you understand the accompanying textual explanations?

☐ Not at all ☐ A little ☐ Half ☐ Most ☐ All

4. **Do you still have any questions or still feel confused about the programming concepts presented in this tutorial session? If so, please provide details.**

C-5: the Extra Questions for the CC+Viz Group

Extra Question 1 (appeared just after participants finished the cognitive conflict question):

You were just told your answer is incorrect. What was your reaction when you were told your answer is incorrect? Please choose one or more of the answers from the following list.

1. To be honest, I did not really care about whether or not my answer is correct.
2. I was surprised because I thought my answer was correct.
3. I was surprised and even wondered if there is something wrong with the system
4. I was not surprised because I was not confident with my answer.
5. I was interested to know why my answer was incorrect.
6. I felt upset

Extra Question 2 (appeared along with other questions in C-1):

Before you went to use the visualization tool, there was a question asking you to predict the result of the execution of the program used in the visualization tool, and then you were told whether or not your answer was correct. How useful do you feel it is to explicitly let you know that your understanding was incorrect before using the visualization tool? Please choose one or more of the answers from the following list.

- ☐ It helped me to concentrate on the learning materials when I was told beforehand that my understanding was incorrect.
- ☐ It helped increase my interest in the learning materials when I was told beforehand that my understanding was incorrect.
- ☐ It encouraged me to actively seek the reason why my understanding was mistaken when I was told beforehand that my understanding was incorrect.
- ☐ It did not make any difference when I was told beforehand that my understanding was incorrect.
- ☐ It decreased my interest in the learning materials when I was told beforehand that my understanding was incorrect.

In addition, there is one additional answer option for the second part of question 1 in C-1, i.e. **And when did you realize that your original understanding was incorrect?**

- ☐ When I was told that my answer was incorrect before using the visualization tool