This issue contains four research articles. The problem researched in the first article by Singh, et. al. is software testing in general and graphical user interface (GUI) testing in particular. The rationale is that GUI testing is one of the major challenges in the lifecycle of any software system. GUI testing is inherently more difficult than the traditional and command-line interface testing. Large numbers of objects, different look and feel of objects, many parameters associated with each object, progressive disclosure, complex inputs from multiple sources and graphical outputs are some of the factors which make GUI testing different from the traditional software testing and significantly more difficult. The existing testing techniques for the creation and management of test suites need to be adapted and enhanced for GUIs, and new testing techniques are desired to make the creation and management of test suites more efficient and effective.

Furthermore, this article addresses this problem by researching a proposed methodology to create test suites for a GUI. The proposed methodology organizes the testing activity into various levels. The tests created at a particular level can be reused at higher levels. This methodology extends the notion of modularity and reusability to the testing phase. The organization and management of the created test suites closely resembles the structure of the GUI under test. Once the User Interface Graph provides a framework to determine test coverage an evaluation is carried out using a proof of concept case study.

In the second article Gupta, et. al. researched the testing of object-oriented programs which consist of several different levels of abstraction, namely, the algorithmic level, class level, cluster level, and system level.

The article then proceeds to proposing a testing methodology to generate test cases at class level for object-oriented programs. The formal object oriented class specification is used to develop a test model. This test model is based on finite state machine specification. The class specification and the test model is analyzed to select a set of test data for each method of the class, and finally the test cases can be generated using other testing techniques like finite-state testing or data-flow testing.

The problem researched in the third article is how to develop and evolve Web Applications viewed from an engineering perspective that relies on and accommodates the knowledge inherent in patterns.

In this article, Kamthan addresses the problem by proposing a methodology for pattern-oriented Web Engineering, namely POWEM. POWEM consists of a sequence of steps that include selection of a suitable development process model, construction of a semiotic qual-
ity model, and selection and mapping of suitable patterns. To evaluate the methodology by supporting decision making and placing POWEM in context, a feasibility study of issues involved in each step are identified. In addition, the use of patterns during the design phase of a Web Application is described. Finally, some directions for evolution of POWEM are highlighted.

The problem research in the final article by Ahmed and Sundaram is that though traditional Decision Support Systems (DSS) provide strong data management, modeling and visualization capabilities for the decision maker, they do not explicitly support scenario management appropriately. Systems that purport to support scenario planning are complex and difficult to use and do not fully support all phases of scenario management.

This article addresses the problem by proposing a framework for Scenario-driven Decision Support Systems Generator (SDSSG). The framework is to align the DSS with the scenario management process that supports usage of scenario as a core component of decision making.

Furthermore, this research presents a life cycle process for scenario management. The proposed process helps the decision maker with idea generation, scenario planning, development, organisation, analysis, execution, and the use of scenarios for decision making. This article also proposes a generalized scenario evaluation process that allows homogeneous and heterogeneous scenario comparisons among multiple instances of similar and dissimilar scenarios respectively. This research introduces scenario as a DSS component and develops a domain independent, component-based, modular framework and architecture that supports the proposed scenario management process. The framework and architecture are evaluated through implementation and validation through a concrete prototype.