Software Engineering Tools

John Grundy
Department of Computer Science, University of Auckland
Private Bag 92019, Auckland, New Zealand
john-g@cs.auckland.ac.nz

Abstract
Software tools are a crucial part of successful software development. Good tool support allows developers to more effectively plan and co-ordinate their work, appropriately specify and design large software systems while maintaining control of their complexity, implement and test software with a range of technologies, and effectively and efficiently maintain software over time. The construction of software tools themselves is a challenge, a significant software engineering task in its own right. This mini-track's papers address many uses of software tools and techniques for tool development.

1. Introduction
The need for good software tools has never been greater. Developers are faced with increasing complexity of software under development, new methods and technologies with which to develop software, and ever-increasing demands from clients and users for better quality software delivered more cheaply and quickly. An essential part of a developer's arsenal to combat these conflicting pressures are quality software engineering tools. Different tools can be deployed during all phases of software development, some during many (e.g. project management, CASE and workflow tools), while others are very specific to a particular task (e.g. documentation, compilation) [1]. Software tools themselves are often very complex and one of the more challenging software projects in their own right [2]. Providing complex visual interfaces, managing complex data models, supporting persistency, and handling multiple, distributed users are all part of many tool-building projects. This mini-track includes six papers offering new and improved insights into software tool construction, design and usage. We hope they help tool developers and users improve their tool use, integration and construction.

2. Mini-track Overview
Many applications provide highly complex graphical user interfaces (GUIs). Subramanian and Chen describe a tool used to test GUIs via specification-based testing with finite-state machines. This improves GUI testing coverage and importantly makes it repeatable.
Monitoring complex distributed systems is very challenging. Rackl describes a tool and technique to monitor heterogeneous distributed system components. This allows developers to better understand their system performance when using diverse component technologies.
The UQ* document editor generator described by Toleman et al. provides a range of document editor specification techniques. These are used to realise highly flexible textual and graphical tools for software developers.
Software specifications and design based on UML-style models continually evolve over a project's life-span. Raush describes a XMI-based tool used to support UML model evolution, ensuring model representation is more readily evolved during a project.
Developers typically work with ever-larger numbers of reusable components. Ye describes a tool for retrieving these from a repository, essential for developers to effectively reuse large component libraries.
Complexity management is the theme of Schuemers's tool, allowing developers to organise their work into "software spaces". These facilitate better management of software and improved collaboration among developers.

3. Conclusions
The need for good software engineering tools is likely to increase rather than lessen [2]. With this in mind, new tools, improved tool construction techniques and improved use of tools are all very important to the future of software engineering. These mini-track papers demonstrate software tools continue to be a fruitful area of research and development.

References