Lectures on Formal Specification and Design

University of Shanghai for Science and Technology

June 2004

Tony Mullins

Introduction

- What do we want from computer programs?
  - Reliability
  - Extendability & Reusability
- Reliability

  Ability of systems to meet their contractual obligations
  - Correctness
  - Robustness

Introduction

- Correctness

  Ability of software to perform according to its required specification

- Robustness

  Software is fault tolerant

Introduction

Is it possible to make software systems that satisfy their requirements and do not fail?

Critical Role of Software Systems

- Monitoring Systems
- Aviation
- Nuclear War-Heads
- Medicine
- Weather Reporting
- Water Dams
- Nuclear Reactors

Professional Responsibility

- What are the consequences of system failure?
- Who is ultimately responsible?
  - Programmer
  - Analyst
  - Client

System Misbehaviour

- Defects in the stated requirements of the system
- Defects in the specification that fail to meet the stated requirements
- Defects in the program that fail to meet the given specification

System Misbehaviour

Other possible reasons for failure:

- hardware errors
- system software faulty
- incomplete specification
- failure to integrate with existing software

System Misbehaviour

It has been shown that the earlier an error is discovered in the development process and the system altered, the cheaper the alteration will be.

Too often errors are only found when testing or deployed!

System Requirements

- Provided by Client
  - Domain experts
  - Brain storming sessions
  - Research
- Documentation
  - Natural language
  - Diagrams

Specification

- Contract between client and system developer
- Developed system must meet requirements stated in specification
- Contract satisfied once system meets specification

Specification Notations

- Informal
  - Natural language + diagrams
- Semi-Formal
  - Graphical notations - JSD, UML
- Formal

Abstract Data Types

- An abstract data type is a set of objects capable only of particular kinds of behaviour, which correspond to a finite set of allowable operations on objects of the type. (Liskov & Berzins)

Abstract Data Types

- Its behaviour is completely characterized by the behaviour of the operations.
- Access to the representation of the object restricted to the operations of the type
- Representation may be changed as long as behaviour of operations preserved

Specification

- Defines contract with client
- Defines **what** system must do
- Abstracts from implementation details
- Provides basis for test cases
- Documentation for system

Program

- Description of **how** to implement the **what** of the contract
- Concerned with:
  - performance issues
  - use of system resources
  - robustness and fault tolerance

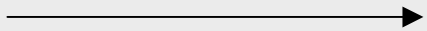
Specification

Correct if there are no errors, ommissions, contradictions or ambiguities in the contract.

Program

- Correct if it performs according to requirements defined in contract
- Robust if it is capable of handling situations not covered in the specification

Degrees of Formality

Specification  Program

How to derive program from specification:

- Formal specification only
- Formal specification & rigorous development
- Formal spec & formal development

Degrees of Formality

Formal Spec only:

- Gives precise unambiguous statement of what system is to do
- Use as guide for design and implementation with informal design techniques
- Valid spec against requirements to discover discrepancies and resolve them
- Used to formulate test cases after coding

Degrees of Formality

Formal spec with rigorous dev:

- Formality applied to both specification and development.
- This means writing both abstract and concrete specifications and recording the development relations between them.
- Mappings from abstract to concrete are not justified

Degrees of Formality

Formal Spec & Formal Dev:

- Formality applied to both specification and development.
- This means writing both abstract and concrete specifications and recording the development relations between them.
- Mappings from abstract to concrete are justified

Degrees of Formality

Formal Spec only:

- Provides solution to **error discovery**
- Fails to solve the **error avoidance** problem because there is no justification of the relation between specification and code.

Degrees of Formality

Formal Spec & Rigorous Dev:

- solve both:
  - error discovery
  - error avaoidance
- Also provide a record of how development was done that is invaluable for maintenance, extendibility and reuse
- Does not solve testing problem

Degrees of Formality

Formal Spec & Formal Dev:

- Solves error detection and avoidance.
- Provides necessary documentation for maintenance, extendibility and reuse
- Solves testing problem

Styles of Specification

- Algebraic
  - Algebras and abstract data types
  - Homomorphic mappings between algebras
- Model Oriented
  - Based on notional state
  - Sets, relations and functions

Algebraic

- A type in an algebra corresponds to a set of elements
- Functions describe the operations associated with the type
- Axioms define the semantics of the operations
- Preconditions define restrictions on the type

Algebraic

Implementations are treated as other algebras and correctness is achieved through showing the existence of homorphic mappings from one algebra to the other.

Stack

- **TYPE**
  - STACK [G]
- **FUNCTIONS**
  - put: STACK [G] X G $\rightarrow$ STACK [G]
  - remove: STACK [G] $\rightarrow$ STACK [G]
  - item: STACK [G] $\rightarrow$ G
  - empty: STACK [G] $\rightarrow$ BOOLEAN
  - new: $\rightarrow$ STACK [G]

Stack

- **AXIOMS**
  - For any x: G, s: STACK [G]
  - A1 item (put (s, x)) = x
  - A2 remove (put (s, x)) = s
  - A3 empty (new)
  - A4 not empty (put (s, x))

Stack

PRECONDITIONS

- remove (s: STACK [G]) require not empty (s)
- item (s: STACK [G]) require not empty (s)

Model Oriented Specifications

- Permit rigorous analysis of system functionality
- Provide proof of consistency and correctness
- Provide a model of **what** system must do
- Abstract from implementation issues

Model Oriented Specifications

- Based on sets, relations, functions and discrete mathematics
- Provide abstract data types through
  - abstract variables
  - state invariants
  - operations on the state
  - operations expressed in terms of a rule that relates pre-condition with post-condition

Model Oriented Specifications

  - Initialisation operations that must satisfy state invariant

- Provide a syntactic and semantic model of composition, i.e. a method to compose new specifications from existing ones.
- A proof theory

Model Oriented Specifications

- Potential for formal derivation of program code from initial specification
- Rules of refinement

Model Oriented Notations

- Z — J. Abrial & Oxford Group
- B-Method — J.Abrial & I. Sorenson
- Perfect — D. Crocker & Escher Technologies

Z

- Model oriented notation
- Used to specify IBM CICS development tools
- Adopted by IBM and used for research at Hursley Park
- Tool support
- Formal Specification only

Z Specification

- Specify given types
- Specify state space
- Specify initialisation
- Specify operations on the state

B-Method

- Model oriented notation
- Used to specify and implement Paris underground metro
- Tool support
- Theorem prover that can be used at all stages of development

B-Method

- Refinement model
- Code generation
  - C++

Perfect

- Model oriented specification language based on Object-Oriented Paradigm
- Used to specify and implement itself!
- Tool support
- Theorem prover that can be used at all stages of development process

Perfect

- Refinement model
- Code generation
  - C++
  - Java
  - Ada

Features of PD

	Teaching	Prof	Enterprise	Safety-critical
Automatic code generation	yes	yes	yes	yes
Automated verification	yes	yes	yes	yes
Unlimited project size		yes	yes	yes
Distributed verification			yes	yes
System-wide property checking			yes	yes
Human-readable proof output	yes	yes	yes	yes
Machine-readable proof output				yes
C++ code generation	yes	yes	yes	yes
Java code generation	yes	yes	yes	yes
Ada 95 code generation				yes

Formality

Both B and Perfect provide development environments that allow:

- formal specification
- proof of correctness
- proof of refinement
- code generation

Platforms

- B
  - Unix and Linux only
    - (B-Core U.K. Ltd, Atelier B, France)
- Perfect
  - Window
  - Unix and Linux
    - Escher Technologies U.K.

Platforms

Perfect developer provides:

- an object oriented specification and design tool;
- verified design by contract.

Perfect Developer

- Student Edition
  - € 120
- Professional Edition
  - € 9000
- Safety Critical Edition
  - € 21500

Course Content

- Lecture 1
  - Introduction
  - Crimson Editor
  - Perfect Developer
- Lecture 2
  - Types
    - nat, char, int, bool, real, string

Course Content

  - Functions
  - Properties and assert
  - Recursion and recursive functions
- Lecture 3
  - Classes in Perfect
    - Object-oriented model
    - Design by contract in Eiffel

Course Content

- Differences between Perfect and Eiffel
- Structure of a class

- Lecture 4
  - Set theory
  - Specifying with sets
- Lecture 5
  - Bag theory
  - Specifying with bags

Course Content

- Lecture 6
  - Theory of sequences
  - Specifying with sequences
- Lecture 7
  - Functions
  - Specifying with functions
- Lecture 8
  - Refinement

Course Content

- Lecture 9
  - System specification
- Lecture 10
  - System specification
- Lecture 11
  - Genericity
- Lecture 12
  - Inheritance

Assessment

- Tutorials & Worksheets
  - 5%
- Assignment
  - 15%
- Exam
  - 80%

Editor

- Crimson Editor
  - Available at:
    - www.crimsoneditor.com
- To configure editor for Perfect put
  - Extension.pd in link directory
  - Perfect.spc in spec directory
  - Perfect.key in spec directory

Editor

- Perfect.key contains
  - collection of perfect key words and their associated colours
- Perfect.spc
  - specification file for Perfect delimiters,comments, quoptes,etc

(Files written by Gareth Carter, Maynooth)

Editor

Run Crimson and Click on "Document -> Syntax Type -> Customize"

Scroll down to an **Empty type** and type

In Description "Perfect"
In Lang Spec "PERFECT.SPC"
In Keywords "PERFECT.KEY"

Perfect Developer

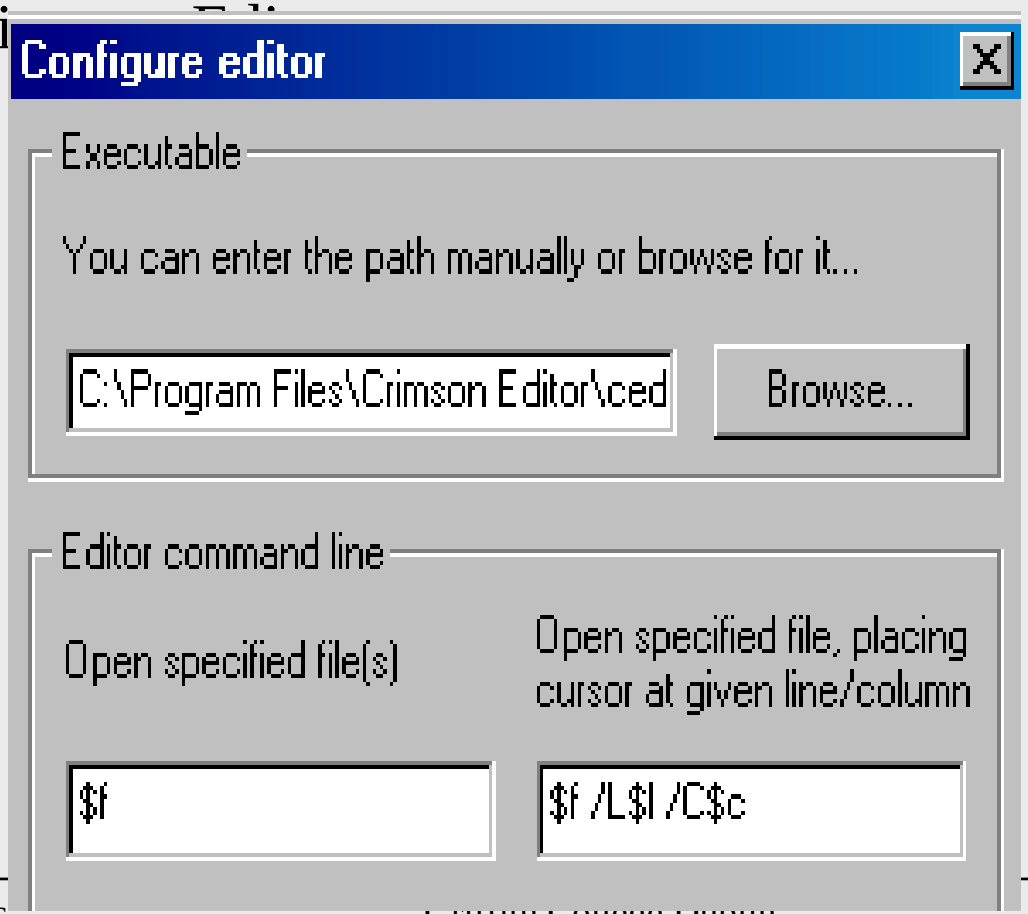
Run Perfect and click on

Options -> Editor

Perfect Developer

Run Perfect and click on

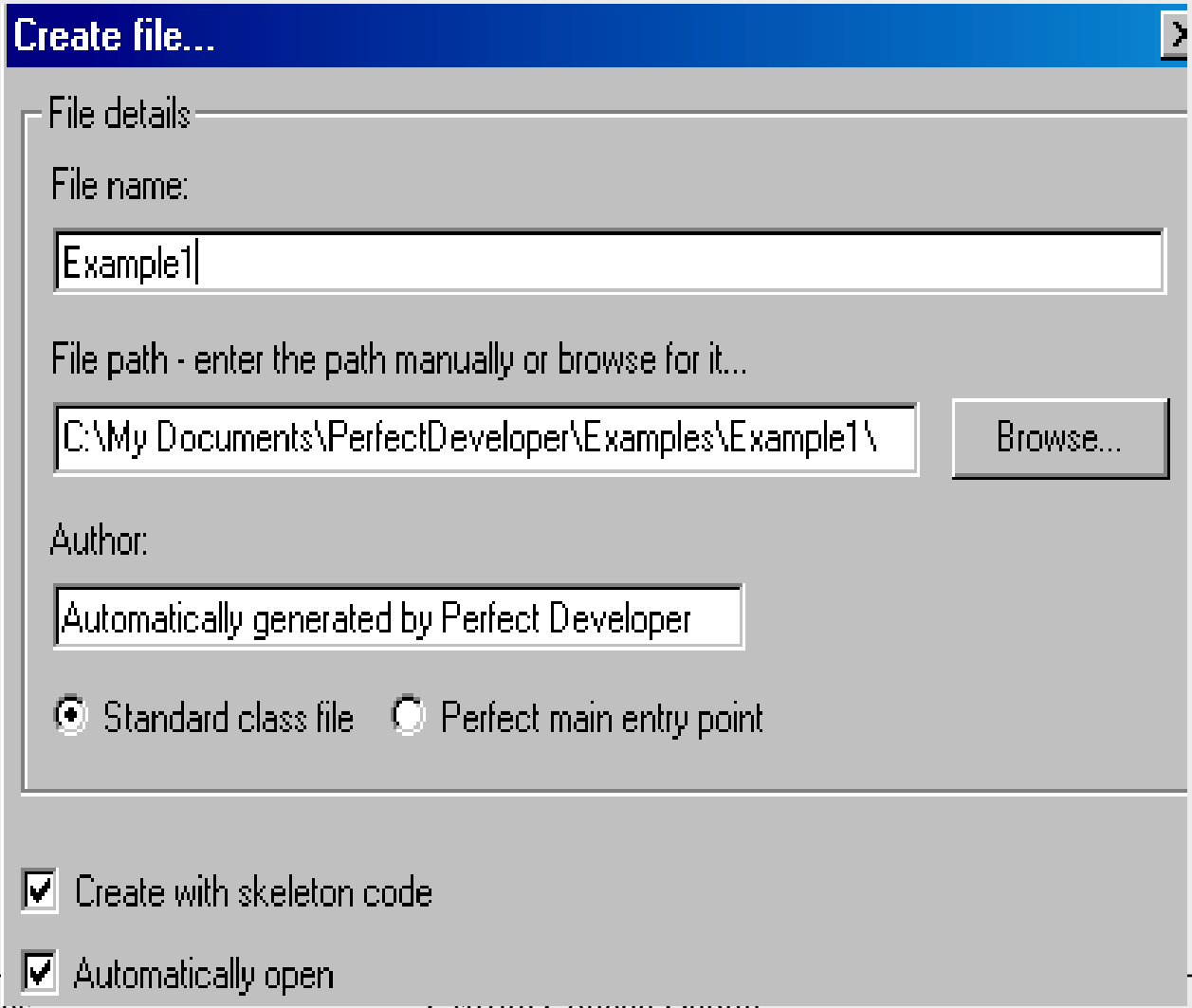
Opti



Perfect Developer

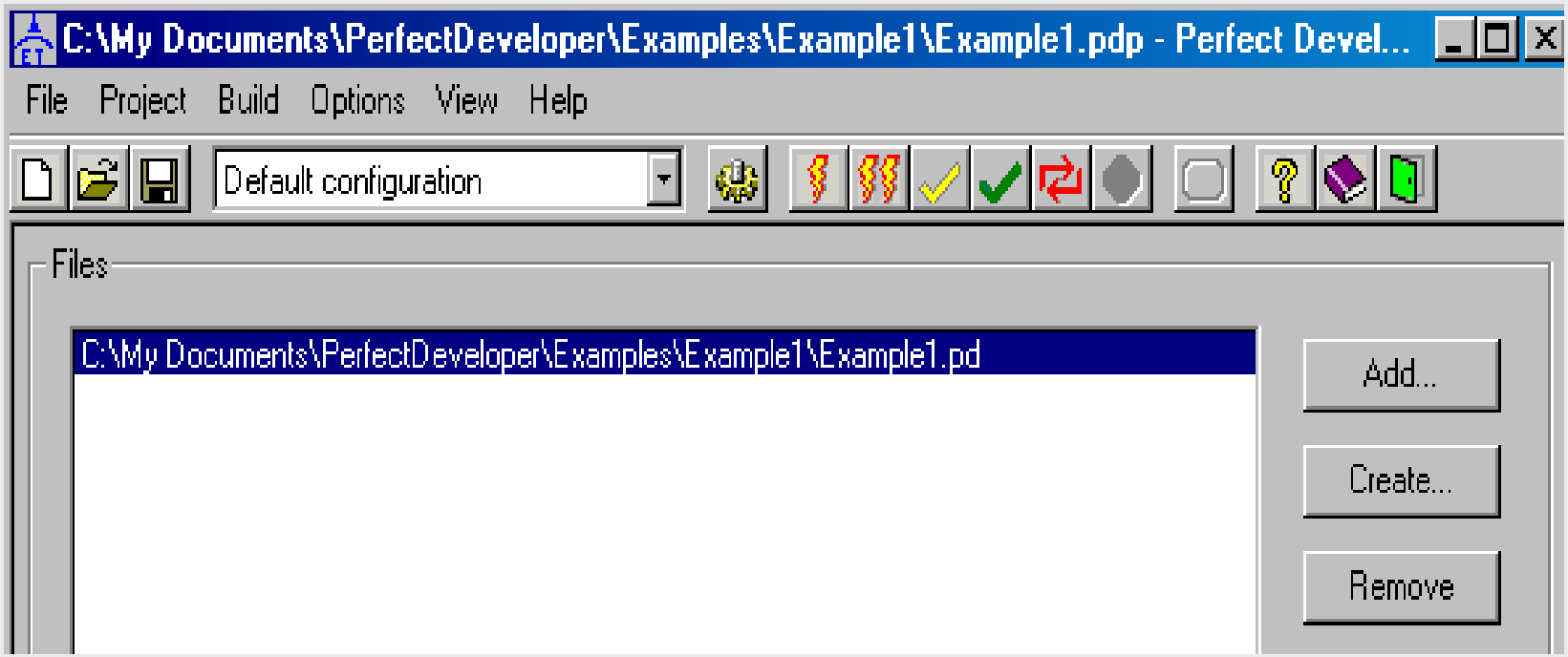
- To create a new project
- Create a directory with project name
- Click on File->New Project
- Select directory
- Click on Project ->Creat file

Perfect Developer



Perfect Developer

- A file called *Exampl1.pd* is created and displayed in the window below



```
/************************************************************
//* File:    C:\My Documents\PerfectDeveloper\Examples\Example1\Example1.pd
//* Author:  Tony Mullins
//* Created: 16:37:14 on Saturday January 17th 2004 UTC
//************************************************************

class Example1 ^=
abstract
        // Add variable, invariant and private method declarations here...
interface
        // Add public access function, selector and method declarations here...
        // ...
        build{}
                post ?;
end;

// End
```