CS6201 Software Reuse

Lecture Notes Set #1: Introduction

Outline of today's lecture:

1. Course overview
2. Software Product Line concepts and examples
3. Fundamental reuse problems
4. Common variation mechanisms and XVCL

Copyright (C) 2010 Stan Jarzabek

About this course

Copyright (C) 2010 Stan Jarzabek

What do we learn in this course?

1. Software Product Line (SPL) approach
   a) Domain analysis
   b) SPL core assets (all what we can reuse)
   c) Components and architectures
   d) Variation mechanisms and why we need them
2. XVCL: reuse technique, used in the project
3. SPL case studies
   – class libraries, Web Applications, others
4. Misc topics related to design for reuse

Copyright (C) 2010 Stan Jarzabek

Practical problems addressed in the course:

- day-to-day software maintenance
- long-term software evolution
- software reuse via product lines

Copyright (C) 2010 Stan Jarzabek

Course organization

- lectures
- project (30%): applying reuse techniques in practice
- presentations of research topics (10%):
  - select a topic for the presentation from the list (check course Web); or propose your own topic – must be approved
  - prepare and conduct 1-hour presentation, Q&A
- exam (60%) – open book, based on:
  - lectures, project, presentations

Copyright (C) 2010 Stan Jarzabek

Presentation and project teams

- there should be max 8 presentation teams
- project teams can be the same as presentation teams or not
  - you can choose to do a project individually

Copyright (C) 2010 Stan Jarzabek

Hands on: reuse with XVCL

- xvcl.comp.nus.edu.sg, open source software
- a generative technique for enhanced reusability and maintainability
- applied on top of conventional OO programs
- XVCL helps control software complexity:
  - avoid redundancy and repetition in software systems (reuse)
  - manage change during maintenance
  - increase software flexibility and adaptability

Copyright (C) 2010 Stan Jarzabek

Project types

- apply XVCL to enhance maintainability and/or reusability
  - ➢ build a new program or work with and existing program
  - ➢ propose your own topic for the project or select from the list
  - ➢ emulate one of the case studies discussed during the lectures

Copyright (C) 2010 Stan Jarzabek

First month of the course at glance

- first four weeks – lectures only, no presentations:
  - Reuse, software product line concepts, examples
  - XVCL briefing
- after that: 1h. presentation + 1h. lecture
- by **January 21**,
  - form presentations teams and let me know your presentation topic (see Web site)
- by **February 4**
  - form project teams (if different from presentation teams) and let me know the topic of your project

Copyright (C) 2010 Stan Jarzabek

Introduction to software reuse

Copyright (C) 2010 Stan Jarzabek

We can develop very complex software

- IBM OS (1960's)
- military software is huge, complex, must be reliable
- WINDOWS (close to 100 million LOC)

2000 BC

12th century

20th century

How can we develop software at lower cost, with higher success rate?

Copyright (C) 2010 Stan Jarzabek

Software engineering challenges

despite new technologies and many successes:

- Software projects are often unpredictable
  - many projects run out of schedule and budget, 25% of large projects are never completed
- Maintenance cost up to 80% of computing cost
  - change is hard, evolution is hard
- Reuse has not become a standard practice
- Outsourcing: a leading software development technique
  - US$ 100 billion, growing trend

Hard work $\neq$ productivity

Copyright (C) 2010 Stan Jarzabek

Some technical challenges

- software models (documentation) integrated with code
  - models developed, maintained and reused in sync with evolving code
  - external docu, UML and generators dilemma: disconnection from code
- traceability from requirements to design and to code
  - how various requirements are implemented?
- managing families of similar software systems (reuse)
  - multiple software releases (evolution) or software Product Lines
  - how to make benefit of commonalties among systems?
  - how to delineate differences among systems from commonalities?

⇨ explosion of similar component versions

⇨ already implemented functionalities are difficult to spot and reuse

Copyright (C) 2010 Stan Jarzabek

Reuse and productivity

- Many companies today:
  - Develop multiple product variants rather than single product
    - Similar products for different customers
  - Then, maintain all those product variants
- These companies can benefit from reuse via Product Lines

Similarities in products

Repetitions in processes

Where to look for productivity improvements?

- We can't cut the cost of creative development activities
- We can cut down the cost of routine, repetitive work
- Similarities: potentials for productivity improvements
- Reuse is suppose to realize these potentials

Copyright (C) 2010 Stan Jarzabek

Software Product Line (SPL)

explained by examples

Copyright (C) 2010 Stan Jarzabek

Project Collaboration Environment (PCE) Software Product Line

References to this study:

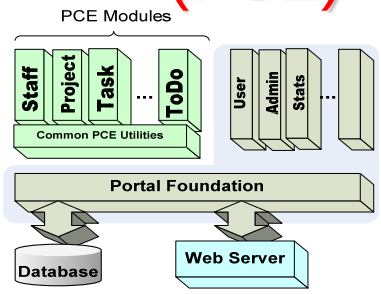
Patterson, U., and Jarzabek, S. "Industrial Experience with Building a Web Portal Product Line using a Lightweight, Reactive Approach," ESEC-FSE'05, Europ. Soft. Eng. Conf, and ACM SIGSOFT Symp. on the Foundations of Soft Eng, Sept. 2005, Lisbon, pp. 326-335;

Rajapakse, D. and Jarzabek, S. "Towards generic representation of web applications: solutions and trade-offs" *Software, Practice & Experience,* Volume 39 Issue 5, April 2009, pp. 501 – 530, Published Online: 27 Nov 2008

Rajapakse, D. and Jarzabek, S. "Using Server Pages to Unify Clones in Web Applications: A Trade-off Analysis," Int. Conf. Software Eng, ICSE'07, Minneapolis, USA, May 2007, pp. 116-125

Copyright (C) 2010 Stan Jarzabek

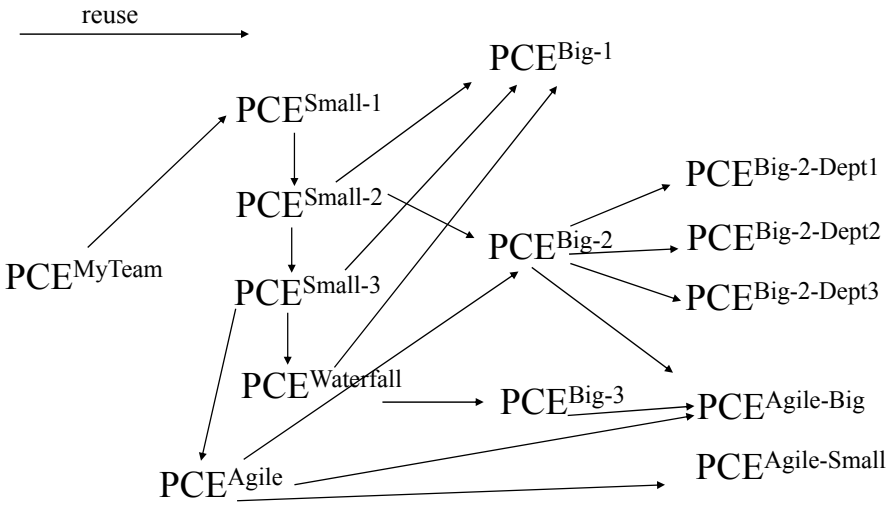
Project Collaboration Envir (PCE)



- PCE stores staff, project data, facilitates project progress monitoring, communication in the team, etc.
- e.g., Module Staff: allows the user to create, edit, and update data about staff members, assign staff members to projects, etc.

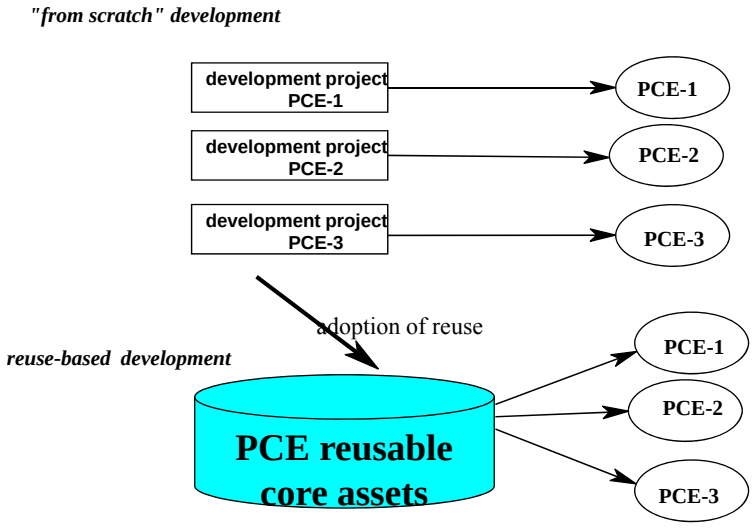
Copyright (C) 2010 Stan Jarzabek

PCE product variants



Copyright (C) 2010 Stan Jarzabek

Transition to reuse-based development



Copyright (C) 2010 Stan Jarzabek

Ad hoc reuse

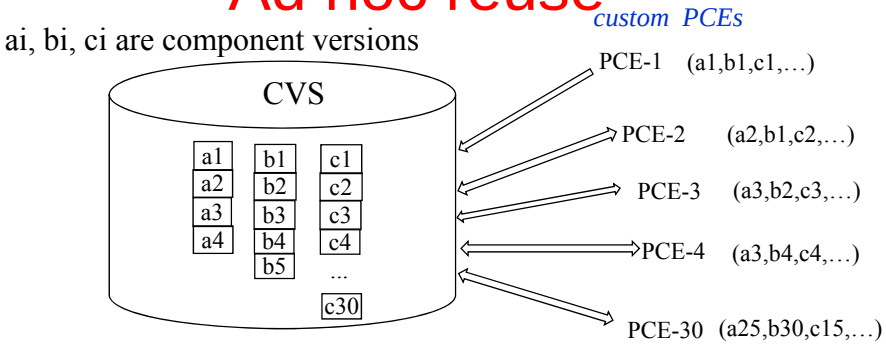
- Store PCE code under software configuration management tool
  - Such as CVS or SVN
- Implementation of a new product:
  - Reuse by **copy-paste-modify** relevant source files from existing products
  - Implement new features into a product

what problems?

Copyright (C) 2010 Stan Jarzabek

Ad hoc reuse

ai, bi, ci are component versions



- Development of new PCE^{New}:
  - Analyze requirements for PCE^{New}
  - Find component versions that "best match" PCE^{New} requirements
  - Customize components (copy-modify), integrate, test
- We maintain/evolve each custom products separately

Copyright (C) 2010 Stan Jarzabek

Problems of ad hoc reuse

- Many component versions stored in CVS
- Tracing features to components not easy
  - Which components implement which features?
  - Which component versions will fit new product?
  - How to find components for reuse?
  - Many errors during component version selection, customization, integration
- We may need to repeat component selection/customization cycle many times before we get it right!
- Many products need be maintained, ignoring much similarity

Copyright (C) 2010 Stan Jarzabek

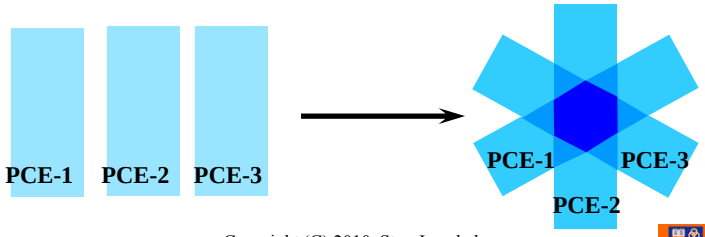
Software Product Line (SPL) definition:

- a family of similar software products that satisfy needs of a particular market segment or customer group,
- managed from a common, reusable base of core assets

Copyright (C) 2010 Stan Jarzabek

Towards PCE Product Line

- Each PCE variant implements:
  - Common features shared by all PCEs
  - Features shared with some of the PCEs
  - Some unique new features
- Implementation of the same feature varies across PCEs
- Solution: reuse! re-engineer into PCE PL



Copyright (C) 2010 Stan Jarzabek

SPL core assets and their reuse

Domain experts

domain knowledge

existing systems in a domain

domain engineering

design and evolution of core assets

Feedback

PCE PL core assets

user requirements

Product development team

reuse of core assets

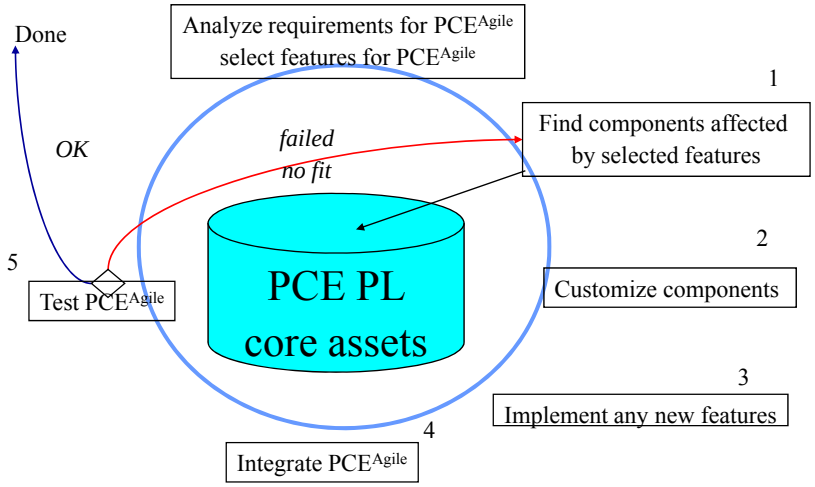
product development

PCE^{Agile}

Copyright (C) 2010 Stan Jarzabek

Reuse-based PCE development

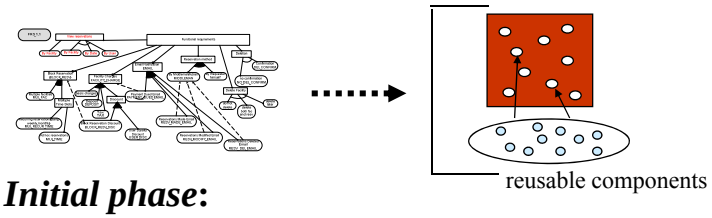
We wish to build PCE^{Agile}



Copyright (C) 2010 Stan Jarzabek

Reuse-based PCE development

1. **Analyze requirements** for PCE^{Agile}: select variant features



2. ***Initial phase*:**
   a) Understand the impact of variant features on components
   b) Find all the feature-related variation points
3. ***Iteration phase*:**
   a) Customize components at variation points
   b) Implement any new features and components
   c) Integrate components, test PCE^{Agile}

Copyright (C) 2010 Stan Jarzabek

Role-Playing Games

(RPG)

Software Product Line

Reference to this study:

Zhang, W. and Jarzabek, S. "Reuse without Compromising Performance: Experience from RPG Software Product Line for Mobile Devices," *9th Int. Software Product Line Conference, SPLC'05*, September 2005, Rennes, France, pp. 57-69

Copyright (C) 2010 Stan Jarzabek

RPGs on mobile phones

RPG: Role Playing Games

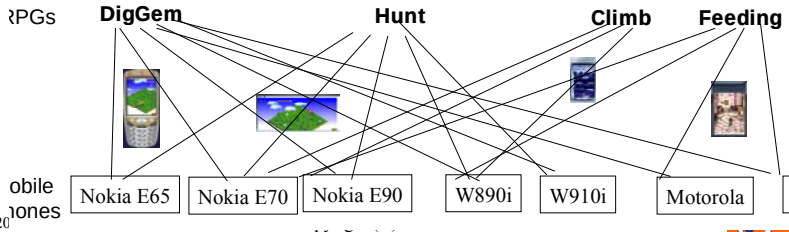
DigGem

Hunt

Jump

Feeding

An opportunity and nightmare for RPG vendors!

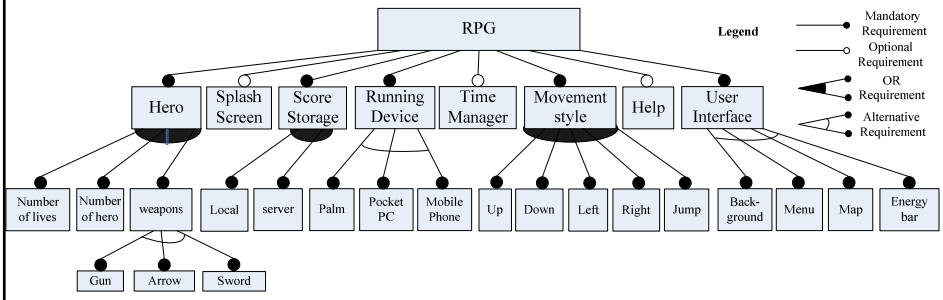


Market forces: good reasons to reuse

- Similar RPGs must run on many types of mobile devices and must perform well
- Many brands and models of mobile devices
  - differ in platforms, communication protocols, display units, memory size, etc.
  - 640 x 200 color screen vs. 100 x 80 mono display
  - 80M memory vs. less than 100kb memory
  - J2ME MIDP2.0 vs. MIDP1.0
- Development cost, time-to-market are important

Copyright (C) 2010 Stan Jarzabek

RPG features



- Two dimensions of variability
  - Game functionality variants
  - Mobile phone platform variants
- Scoping RPG PL (Product Line)

Copyright (C) 2010 Stan Jarzabek

So – what are features?

Feature: any system characteristics from use or developer view point

- User requirements (functionality)
- Quality requirements
- Platform characteristics
- Design alternatives

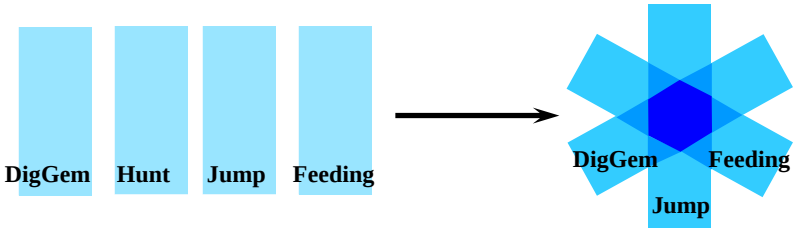
Features show how products are similar and different:

- Common features
- Variant features:
  - optional features, alternative features, OR-features

Copyright (C) 2010 Stan Jarzabek

Towards RPG PL

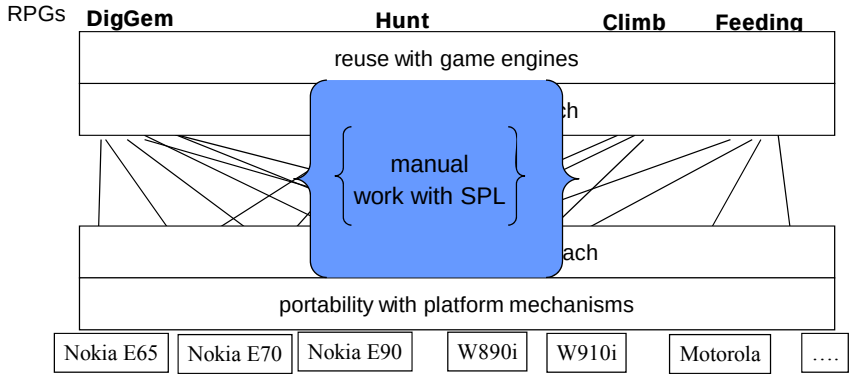
- Each RPG implements:
  - Common features shared by all RPGs
  - Features shared with some of the RPGs
  - Some unique new features
- Implementation of the same feature varies across RPGs
- Solution: reuse! re-engineer into RPG Product Line



Copyright (C) 2010 Stan Jarzabek

Vendor provided reuse solutions

Game engines and platform mechanisms



Copyright (C) 2010 Stan Jarzabek

SPL core assets and their reuse

Domain experts

domain knowledge

existing systems in a domain

domain engineering

design and evolution of core assets

Feedback

RPG PL core assets

user requirements

Product development team

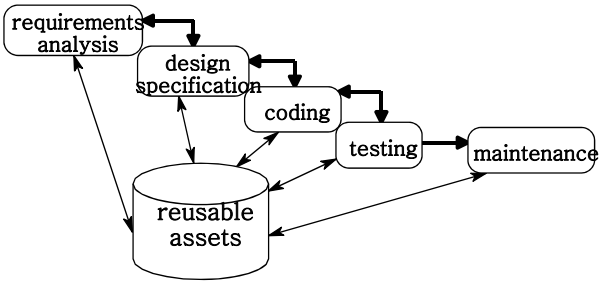
reuse of core assets

product development

DigGem for Nokia

Copyright (C) 2010 Stan Jarzabek

What software assets should we reuse?



- source code, code deployed to the customer
- design (component architecture) and requirement specifications
- documentation: technical (models), user (manuals)
- test cases
- software processes, best practices, company standards

Copyright (C) 2010 Stan Jarzabek

What are SPL core assets?

SPL core assets include all software assets that form a product and whose reuse is beneficial

- Common architecture shared by products
  - Core components and their organization
  - Component interfaces
  - All important design and technical decisions, strategies
- Common implementation (parameterized)
- Variation mechanisms to manage product variability
  - Conditional compilation, Ant, make, parameter files, …
- Product derivation methods, techniques and tools
  - Help developers build custom products with reuse of assets
- Models, technical documentation, user manuals
- Test cases

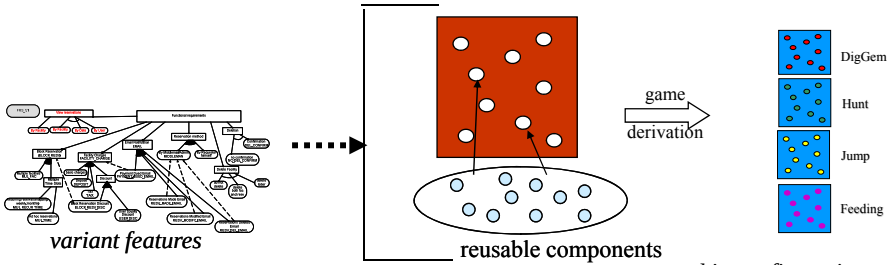
Product Line Architecture

PLA

Copyright (C) 2010 Stan Jarzabek

Steps towards reuse in RPG domain

- How are RPGs similar and different?
  - business-oriented analysis of variability in a domain (top-down)
  - observe repetitions across similar RPGs (bottom-up)
- Design architecture and reusable components for RPGs
  - Apply extra variation mechanisms for component adaptation



Copyright (C) 2010 Stan Jarzabek

Case study of reuse practice at

Fudan Wingsoft Ltd

Wingsoft Financial System

Software Product Line - WFS PL

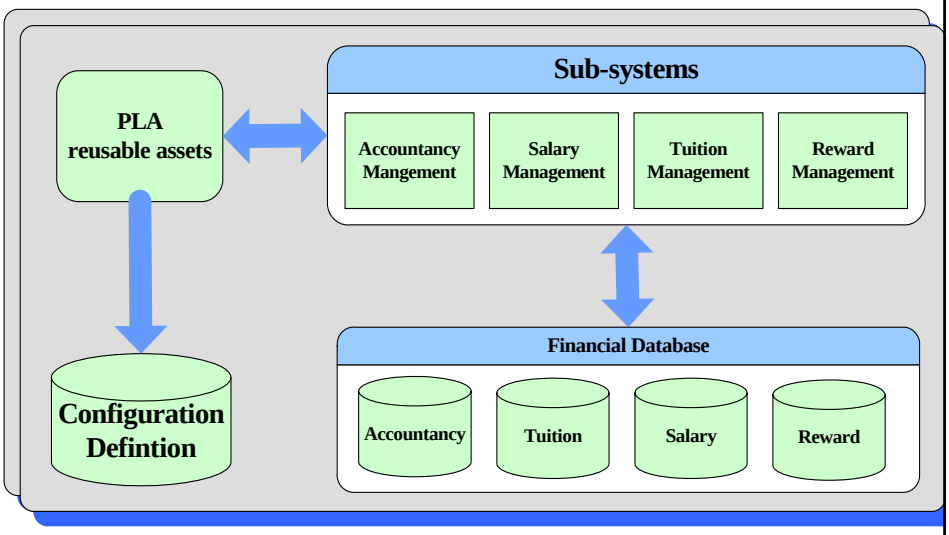
Copyright (C) 2010 Stan Jarzabek

Wingsoft company and Product Line

- Fudan Wingsoft Ltd: a small company in Shanghai (60 staff)
- Wingsoft Financial System (WFS):
  - supports financial operations at universities
  - first WFS developed in 2003
  - evolved to an SPL used at more than 100 universities
- Our case study: Tuition Management Subsystem (TMS)
  - A web-based portal for students to pay tuition fee
    - 58 Java source files
    - 99 other source files: JSP (HTML) files, configuration files (XML), DB schema (SQL Scripts)

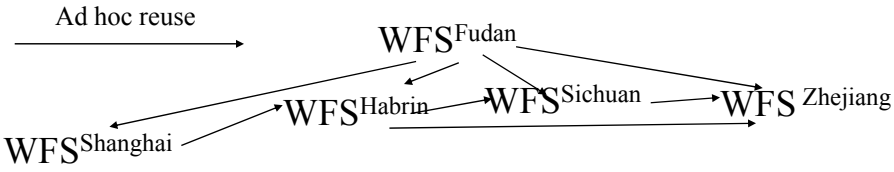
Copyright (C) 2010 Stan Jarzabek

WFS functions and architecture



Copyright (C) 2010 Stan Jarzabek

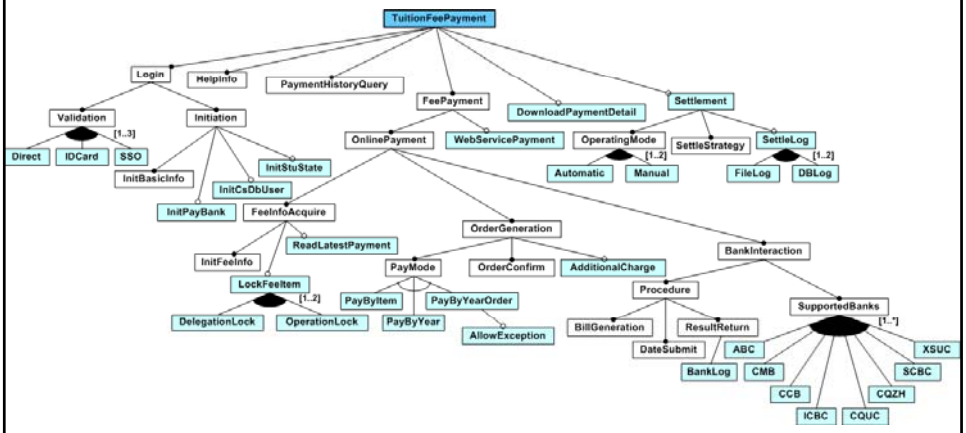
A journey towards Product Line



1. Started with WFS developed for Fudan University
2. Five WFS variants developed for other universities
   - Ineffective ruse by copy and modify files, maintenance problems
3. Re-engineered into WFS Product Line WFS PL
   - Refine and stabilize WFS architecture
   - Variant features handled with many simple variation mechanisms
4. WFS product variants used at over 100 universities

Copyright (C) 2010 Stan Jarzabek

WFS features (32)



Copyright (C) 2010 Stan Jarzabek

WFS PL core assets and their reuse

Domain experts

domain knowledge

existing systems in a domain

domain engineering

design and evolution of core assets

Feedback

WFS PL core assets

user requirements

Product development team

reuse of core assets

product development

WFS Habrin

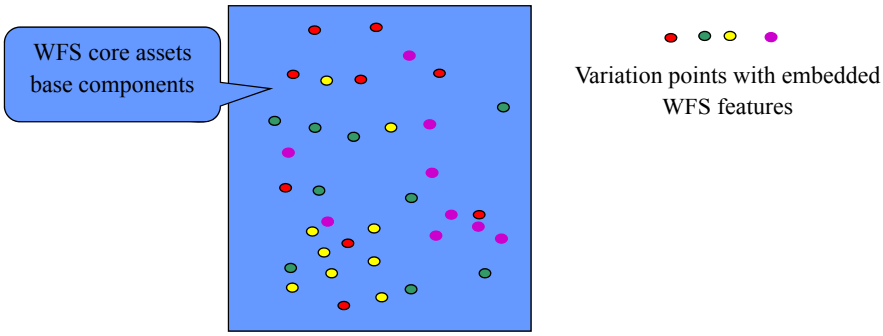
Copyright (C) 2010 Stan Jarzabek

How Wingsoft did WFS PL?

- WFS core assets: All-in-one, customizable product
- Set up WFS component architecture: base components
- Apply common variation mechanisms to embed features in base components
  - Conditional compilation & comments
  - Design patterns & reflection
  - Overloaded fields
  - Ant
  - Parameter configuration files

Copyright (C) 2010 Stan Jarzabek

All-in-one reusable product



- To derive a custom WFS, we enable required features at variation points

Copyright (C) 2010 Stan Jarzabek

Conditional Compilation

- To manage fine-grained features in Java code

```
1  public class FeatureConfiguration {
2    // Configuration items
3    public static final boolean DelegationLock = true;
4    public static final boolean OperationLock = true;
5  }
```

```
1  public class FeeInfo {
2    . . .
3    public void initInfo(FeeUser user, boolean isPaidFeeInfo)
4      throws Exception {
5      //get each year's fee items
6      for( int i=0; i < yearTemp.size(); i++ ) {
7        if ( FeatureConfiguration.DelegationLock
8          && FeatureConfiguration.OperationLock )
9          // Code when both features are selected
10       else if ( FeatureConfiguration.DelegationLock )
11         // Code when delegationLock is selected
12       else if ( FeatureConfiguration.OperationLock )
13         // Code when operationLock is selected
14
```


Copyright (C) 2010 Stan Jarzabek

Commenting out feature code

- To manage fine grained features in non-Java files
  - DB schema definitions, JSP files, etc.

```
1  create table userInfo(
2     uniNo char(21),
3     name char(30),
4     password char(21),
5     id_card  char(20),
6     inYm  char(6),
7     banks  char(50),
8  // If feature InitPayMode is selected, use the following field
9  // to record pay mode for each student payMode char(1) default 'F'
10    feeDBUser  char(50),
11
12    primary key(unino)
13 );
```


Copyright (C) 2010 Stan Jarzabek

Design patterns

- AbstractFactory with FactoryMethod
- Strategy pattern
- Patterns used with other variation mechanisms such as reflection and Ant)

```
1  public class FeeOrder {
2      private Initializer initializer;
3      public init(FeeUser user, FeeInfo info, HttpServletRequest request) {
4          Class c;
5          try{
6              c = Class.forName( user.getPayMode() );
7              initializer = (Initializer) c.newInstance();
8              initializer.init ( . . .);
9          } catch(Exception e ) {
10             e.printStackTrace();
11         }
12     }
13 }
```

Example:
Strategy Pattern
with Reflection

Copyright (C) 2010 Stan Jarzabek

Overloaded Fields

- Used for the customizing DB schema
- The same field used for different purposed in different WFS variants
  - E.g. the same table field may be used to store bank card number in one product variant and ID card number in another one
- Pros: several product share the same DB schema, not need to configure for specific product
- Cons: hard to understand when many product variants share the same field, error-prone

Copyright (C) 2010 Stan Jarzabek

Ant

- For customizing coarse-grained features
- The build tool configures product components

```
1  <project name="webfee"  basedir="." default="main">
2   <target name="copy-src" depends="create-folders">
3    <!-- Copy java classes of Feature DownloadPaymentDetail -->
4    <copy todir="${src.dir}">
5     <fileset dir="${core-src.dir}/${DownloadPaymentDetail}"/>
6    </copy>
7   </target>
8   <target name="copy-webpage"
9    depends="create-folders">
10   <!-- Copy webpages of Feature DownloadPaymentDetail -->
11   <copy todir="${web-root.dir}">
12    <fileset dir="${core-webpage.dir}/${DownloadPaymentDetail}" />
13   </copy>
14  </target>
15 <project>
```

Example: configure *DownloadPaymentDetail* with Ant

Parameter configuration files

- Contain both data and control parameters (XML)
- A tool reads the file and does configuration
  - e.g., generates the Ant configuration file

```
1  <webFee>
2    <paymode>PayByItem</paymode>
3    <bank-info>
4     <supportedBank>
5      <bank>ICBC</bank>
6      <bank>CCB</bank>
7      <bank>CMB</bank>
8     </supportedBank>
9     <ICBC>
10     <bankUrl>http: //mybank.icbc.com.cn/servlet/co...</bankUrl>
11     <keyPath>C: /apache-tomcat-/webapps/…</keyPath>
12     <keyPass>12345678</keyPass>
13     <merchantid>440220500001</merchantid>
14    </ICBC>
15   </bank-info>
16   <DownloadDetail>true</DownloadDetail>
17 </webFee>
```

Variation mechanisms in WFS-PLA

# Techniques	# Features
Conditional compilation & comment	31
Ant	19
Overloading fields	13
configuration items	12
Design Pattern & reflection	3

Copyright (C) 2010 Stan Jarzabek

Multiple variation mechanism : Pros

- Expressive power: any unique feature of WFS can be handled by conditional compilation, comments, Ant, etc.
- Common variation mechanisms are easy to learn, apply
  - No need for training, third party tools, etc.

On overall: Multiple variation mechanism strategy worked fine for WFS-PLA (over 100 custom products maintained by 1-2 developers)

Recommendation: It is the right strategy for small- to medium-size SPLs (<10K LOC, <100 features)

Copyright (C) 2010 Stan Jarzabek

Multiple variation mechanisms: Cons

- Poorly compatible variation mechanisms used together create problems
- Using many mechanism together becomes complicated:
  - Manual, error-prone customizations (reuse)
  - Manual evolution of core assets (PLA)

Copyright (C) 2010 Stan Jarzabek

A key reuse problem:

How features affect reusable components?

- If I select feature f – which components are affected and how?
- If I select features f1, …, f2 – which components are affected and how?

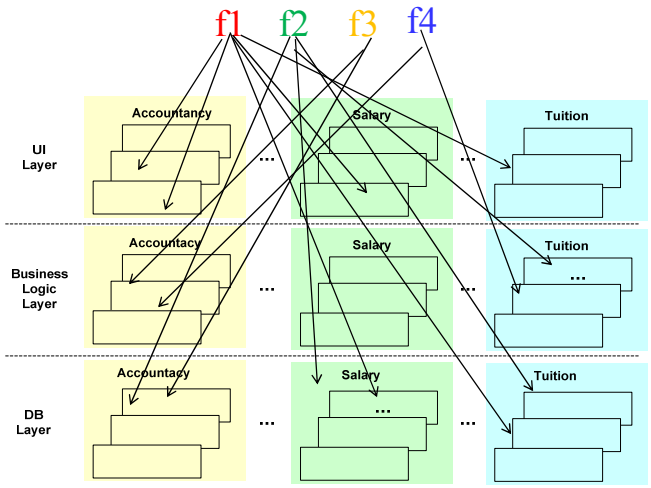
Copyright (C) 2010 Stan Jarzabek

Components with embedded features

```
1  function editable_form($form, $id=0, $relModID=null,
                          $Container=null, $ContainerID=null)
2  {
   global $module_name, $attributes;
   If($form=="Edit"){
       $msg=" Please modify the $module_name data";
       $title="Edit $module_name";
       $nextAction="saveChanges";
   } elseif ($form=="createInsideContainer"){
       $msg=" Please enter the $module_name data";
       $title="New Composed $module_name ";
       $nextAction="saveInstanceInsideContainer";
   } elseif ($form=="copy"){
       $msg="Please enter the ".$module_name." data";
       $title="Copy ".$module_name;
       $nextAction="saveCopy";
   }else{
       $msg=" Please enter the $module_name data";
       $title="New $module_name ";
       $nextAction="saveInstance";
   }
3  If($form=="Edit"){
       //check user has rights to edit instance…
   } elseif ($form=="createInsideContainer"){
       //check user has rights to edit container…
   }elseif($form=="copy"){
      //retrieve data to be copied
   }
4  Title($title);
   <form><table>
   <tr><td> echo($msg); </td></tr>
   <tr><td>Title</td>
   <td><input name="Data[Title]" type="text"
       value=" if(($form=="Edit")||($form=="copy")) echo
   $result['Title'];"></td></tr>
   If($form=="Edit"){
       // retrieve data, show link to container, if exists…
   } elseif ($form=="createInsideContainer"){
       //show link to container
   }
5  foreach($attributes as $attribute)
       <tr><td>echo $attribute; </td>
       <td><input name="Data[echo $attribute;]"
            type="echo ($module_name=="File" &&
                        $attribute=="FileUpload")? "file":"text";"
            value="if(($form=="Edit")||($form=="copy")) echo
   $result[$attribute];" >
       </td></tr>
   }
6  If($form=="Edit"){
       <tr><td>Change Remark</td>
       <td><textarea name="Remark" ></textarea></td></tr>
   }
   <input type="hidden" name="cmd" value="$nextAction">
   }
```


Copyright (C) 2010 Stan Jarzabek

Impact of features on components



Copyright (C) 2010 Stan Jarzabek

Feature management in SPL

- Feature may mean any product characteristic
- One feature may affect many product components

Features interactions:

- Functionally interdependent features:
  - If I select one feature I must also select some other features
- One feature may affect implementation of other features

Copyright (C) 2010 Stan Jarzabek

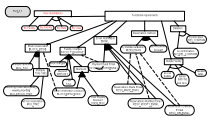
Types of features

- Coarse-grained feature: implemented in source files that are included into a customized product when feature is selected
- Fine-grained feature: affects many product components, at many variation points
- Mixed-grained feature: involves both fine- and coarse-grained impacts on components

Copyright (C) 2010 Stan Jarzabek

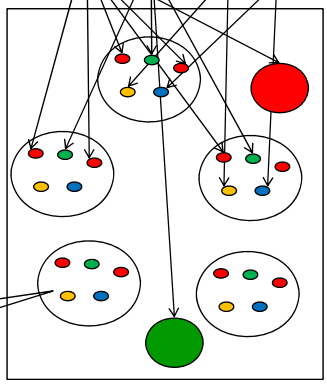
Reuse challenge:

Tracing the impact of features on components



Select features for WFS^{Habrin}

f1 f2 f3 f4



WFS^{Habrin}

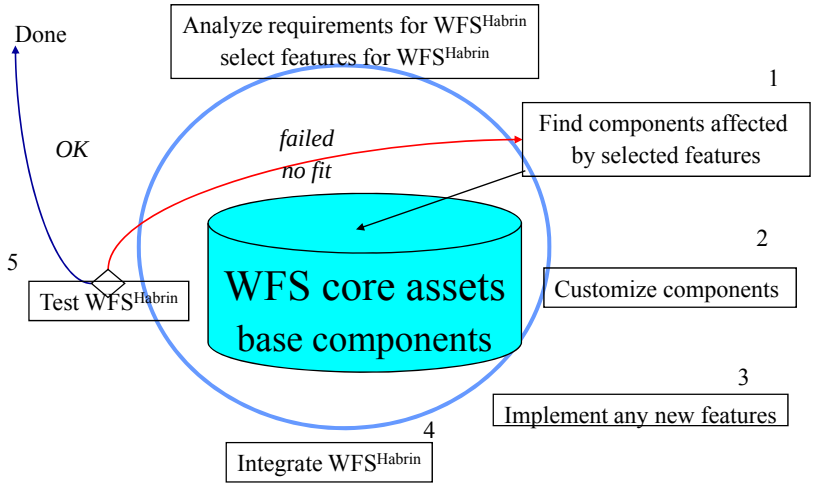
Manual customization

reusable components with variation points

Copyright (C) 2010 Stan Jarzabek

Reuse-based WFS development

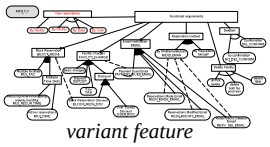
We wish to build WFS^{Habrin}



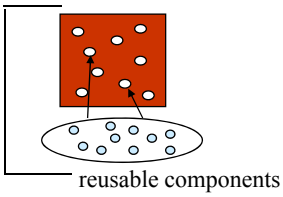
Copyright (C) 2010 Stan Jarzabek

Reuse-based WFS development

1. **Analyze requirements** for WFS^{Habrin} : select variant features



variant feature



reusable components

2. ***Initial phase*:**
   a) Understand the impact of variant features on components
   b) Find all the feature-related variation points
3. ***Iteration phase*:**
   a) Customize components at variation points
   b) Implement any new features and components
   c) Integrate components, test WFS^{Habrin}

Copyright (C) 2010 Stan Jarzabek

In the course we study better ways to manage features

Copyright (C) 2010 Stan Jarzabek

WFS PL in subset of XVCL

- XML-based Variant Configuration Language
- A construction time variation mechanism for SPL
- All-in-one solution to managing features in SPL
- Used in sync with conventional programming technologies:
  - Java/XVCL, ASP/XVCL, PHP/XVCL, J2EE/XVCL, .NET/XVCL
- Public domain, available at sourceforge

http://xvcl.comp.nus.edu.sg

Copyright (C) 2010 Stan Jarzabek

SPC // specifies feature selection for WFS^{NEW}
<**set** a = "**A**" />
<**set** d = "**D**" />
<**set** b = ""/> <**set** c = ""/>
<**adapt** DBSchema/>
<**adapt** FeeUser/>

Suppose A, B, C, D are all WFS features and we selected A and D for WFS^{NEW}

DBSchema
<**set** v = **@a** />
<**select** v > // feature A affects DBSchema here
<**option A**> *if feature **A is** selected*
<**otherwise**> *if **A is not** selected*
</**select**>
…some code for DBSchema
<**set** v = ..>
<**select** v > // another variation point in DBSchema

FeeUser
public class FeeUser *{*
…some code for FeeUser
<**set** v = **@a @c @d** />
<**select** v > // feature s A, C and D affect FeeUser here
<**option A**> *code for feature A*
<**option A D**> *code for feature interaction A and D*
<**option A** C D> *code for feature interaction A, C and D*
<**otherwise**>
</**select**>
… some code for FeeUser
<**set** v = .. >
<**select** v > // another variation point in FeeUser

Copyright (C) 2010 Stan Jarzabek

SPC

```
// Login feature group
<set IDCard = "IDCard"/>
<set SSO = "SSO"/>
<set Direct = ""/>
// Paymode feature group
<set PayByItem = "PayByItem"/>
<set PayByYear = ""/>
<set PayByYearOrder = ""/>
<set InitPayMode = "InitPayMode"/>
<adapt FeeOrder />
<adapt DBSchema/>
<adapt FeeUser/>
...
```

<set PayMode = "PayByYear"/>

FeeOrder

```
<set PayMode = "PayByItem"/>
public class FeeOrder {
  public init(FeeUser user, FeeInfo info,
  HttpServletRequest request) {
...
try
  @PayMode c = new @PayMode();
    c.init ( . . .);
  } catch(Exception e ) {
    e.printStackTrace();
  }
 }.. .
}
```

DBSchema

```
create table userInfo(
  uniNo char(21),
  name char(30),
  password char(21),
  id_card char(20),
  inYm char(6),
  banks char(50),
<select InitPayMode >
   <option InitPayMode >
       payMode char(1) default 'F',
 </select>
  feeDBUser char(50),
primary key(unino)
);
```

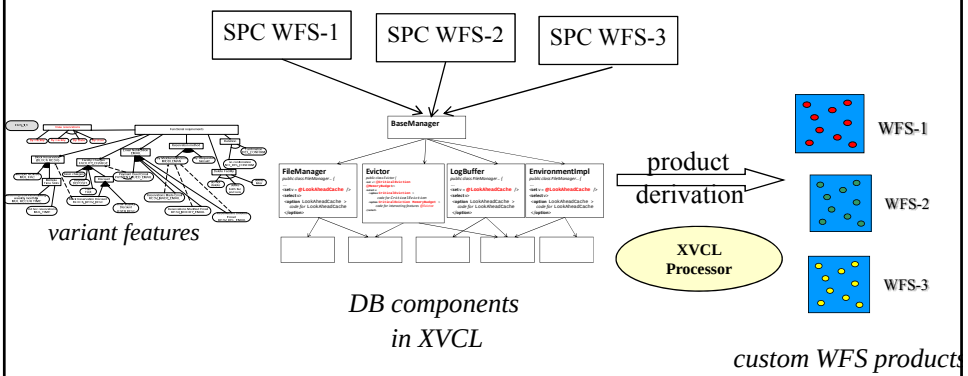
FeeUser

```
public class FeeUser {
  public FeeUser() throws Exception {
  <set Login = @IDCard @SSO @Direct />
// feature s IDCard, SSO and Direct affect FeeUser
     <select Login >
       <option IDCard>
       …
       <option IDCard SSO>

     </select>
}
 public boolean login() throws Exception{
        …
         <select InitPayMode >
         <option "InitPayMode">
            payMode = Global.nTrim
           (rs.getString( "payMode"
)).charAt( 0 );
         </select>
   ....}  }
```


Copyright (C) 2010 Stan Jarzabek

Generation of custom WFS products

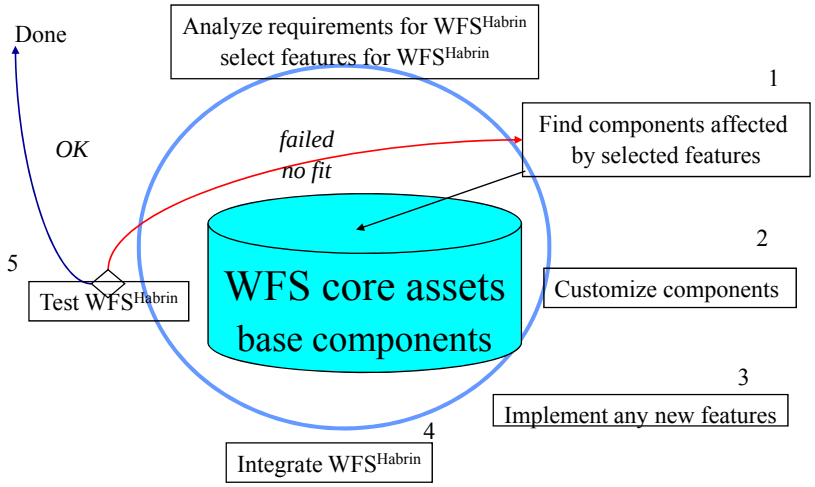


Reuse is automated by XVCL Processor

Copyright (C) 2010 Stan Jarzabek

Reuse-based WFS development

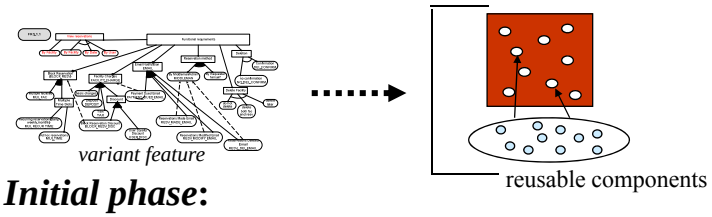
We wish to build WFS^{Habrin}



Copyright (C) 2010 Stan Jarzabek

Reuse-based DB development

1. **Analyze requirements** for DB^{New} : select variant features



2. ***Initial phase*:**
   a) Understand the impact of variant features on components
   b) Find all the feature-related variation points
3. ***Iteration phase*:**
   a) Customize components at variation points
   b) Implement any new features and components
   c) Integrate components, test DB^{New}

Copyright (C) 2010 Stan Jarzabek

Problems still remain:

- Things work fine as long as we reuse features "as is"
- Base components with embedded features may be complex to work with
- To modify feature, we must:
  - find all variation points relevant to a given feature
  - understand feature
  - understand feature interactions
- Adding new feature can be also complex

Copyright (C) 2010 Stan Jarzabek

Feature queries

- FQL: Feature Query Language
- A tool locates and shows all variation points relevant to a given feature
- Show all variation points where feature "f" affects components

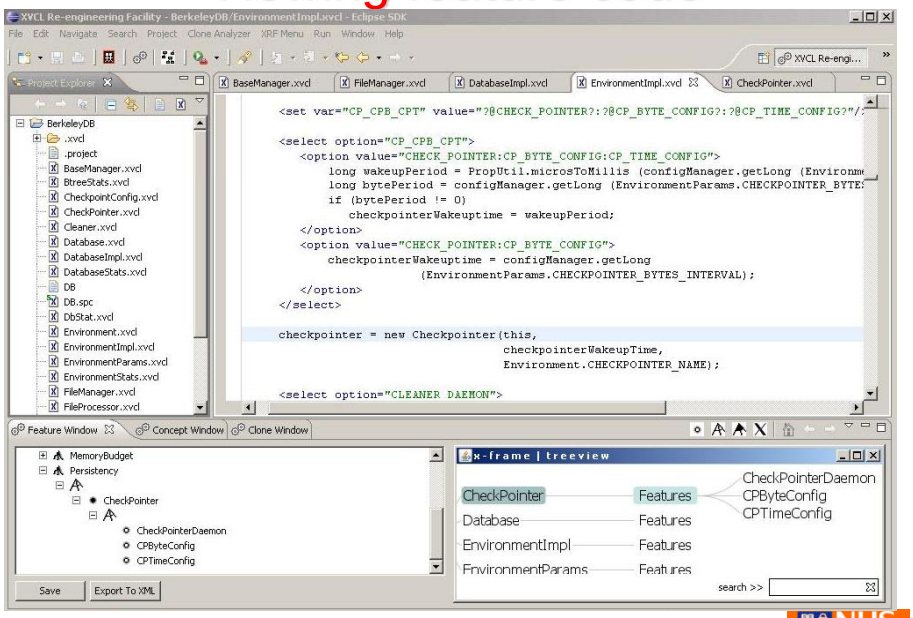
```
Declare option o
Select o
   where o.feature="f"
```

- Show all variation points where feature "f" interacts with other features

```
Declare option o
Select o
  where o.feature="*f*"
```


Copyright (C) 2010 Stan Jarzabek

Viewing feature code



Copyright (C) 2010 Stan Jarzabek

Summary of approach

- Embed features in reusable components
- A mechanism to *compose* required features into the product
- Mark each variation point with names of interacting features
- Formally inter-link all variation points affected by a given feature
- Query-based visualization of features and their interactions

Copyright (C) 2010 Stan Jarzabek

Evaluation: problems solved

- Legality of feature selection
  - Validation done prior to feature processing (Zhang, H)
- Automation of product derivation
  - feature composition into base done by XVCL Processor
- Feature comprehension
  - each variation point marked with names of interacting features
  - inter-linking all variation points affected by a given feature
  - query-based visualization of features and their interactions

Copyright (C) 2010 Stan Jarzabek

Problems still remain

- Solution gets complicated as the size of product increases, and the number of features and feature dependencies grows
  - True, we can find feature code – but how to understand, maintain and reuse features if their code spreads though many variation points, in many base components?
- Assumption of "base components" is limiting
  - we can't contain the impact of features at the implementation level only – use design!
- Direction for possible improvements:
  - Reduce the number of variation points
  - Relax the assumption of a "base components"
  - Represent products in generic form (full XVCL)

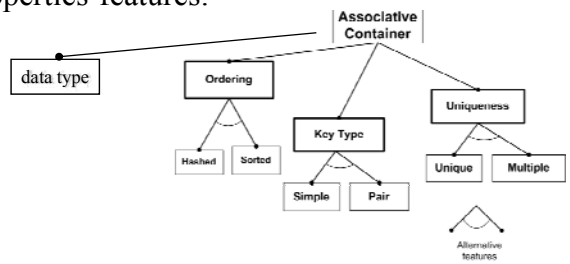
Copyright (C) 2010 Stan Jarzabek

An example of perfect reuse

Copyright (C) 2010 Stan Jarzabek

STL

- STL: a library of C++ classes and functions:
  - containers: stack, queue, set, …
  - operations: sort, search, …
- similar data structures and operations are differentiated by all kinds of properties-features:



- for each legal combination of features (container, operation, data type, etc.) – we need a class

Copyright (C) 2010 Stan Jarzabek

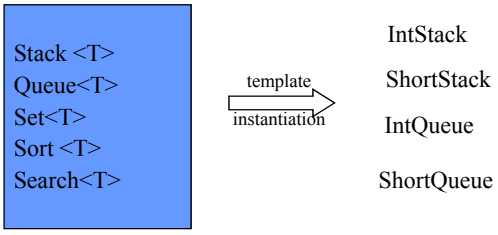
STL classes

- we need a lot of classes and functions:
- Stacks of int, float, double, char, …
  - IntStack, ShortStack, LongStack, FloatStack …
- Queues of int, float, double, char, …
  - IntQueue, ShortQueue, LongQueue, FloatQueue …
- Sets:
  - IntSet, ShortSet, LongSet, FloatSet …
  - IntSet (*Hashed, Single, Unique)*
  - IntSet (*Sorted, Pair, Unique)*
- search, sort functions for different Containers
  - *SortStack, SortQueue, SearchStack, SearchSet, …*
- etc.

Copyright (C) 2010 Stan Jarzabek

Perfect reuse

- STL is a perfect example of effective reuse
  - A template (generics) represents a group of similar classes in generic, adaptable form:



Question: Can we scale templates to a general reuse paradigm?

Copyright (C) 2010 Stan Jarzabek

STL uses the

principle of generic design

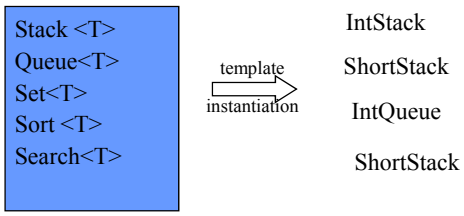
to tackle repetitions, and

to achieve reuse

Copyright (C) 2010 Stan Jarzabek

STL as a Product Line

- each concrete class we consider as a member of STL Product Line
  - IntStack, ShortStack, LongStack, FloatStack …
- templates form a PLA for STL



Copyright (C) 2010 Stan Jarzabek

STL Product Line

STL designers

domain knowledge

existing systems in a domain

STL design and evolution

Feedback

STL

user requirements

programmers

instantiate and reuse STL templates

program

Copyright (C) 2010 Stan Jarzabek

Interesting questions

1. What's the essence of STL's way of reuse?
2. How would STL look like if we designed it using architecture/component approach?
3. Which one is more effective in tacking repetitions?
4. Can we apply STL-like solution to reuse in other application domains?
   - Can we built parameterized PLAs in other domains?
   - When we can and when we cannot?
5. Can we enhance architecture/component approach to reuse with the STL's ability to tackle repetitions?

In this course we try to answer these questions

Copyright (C) 2010 Stan Jarzabek

XVCL

XML-based Variant Configuration Language

- A simple mechanism for unrestricted generic design
- Automated by XVCL Processor
- Used in sync with conventional OO/component technologies:
  - C, C++, Java, ASP, PHP, JEE, .NET, etc.
- Public domain, available at http://xvcl.comp.nus.edu.sg
- XVCL method supported by XVCL Workbench
- Based on Bassett's frames, Frame Technology™, Netron, Inc

Copyright (C) 2010 Stan Jarzabek

Generic components with XVCL

Copyright (C) 2010 Stan Jarzabek

Toy example: similar Account classes

```
class SavingsAccount {
  public static void main(String[] args) {
    System.out.println("This is a bank account");
    System.out.println("Savings Account");
  }
}
```

```
class CurrentAccount {
  public static void main(String[] args) {
    System.out.println("This is a bank account");
    System.out.println("Current Account");
  }
}
```

```
class LoanAccount {
  public static void main(String[] args) {
    System.out.println("This is a bank account");
    System.out.println("Loan Account");
    System.out.println(" Interest");
  }
}
```


Copyright (C) 2010 Stan Jarzabek

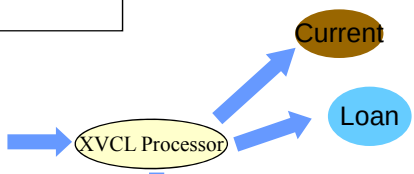
Generic class Account

SPC

```
<set className = SavingsAccount />
<set messages = This is a bank account, Savings Account/>
<adapt Account />
```

Account

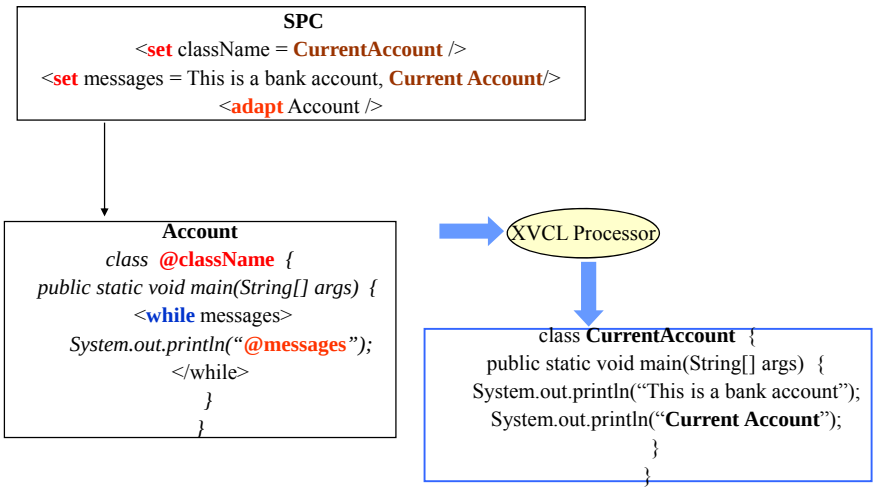
```
class @className {
public static void main(String[] args) {
  <while messages>
  System.out.println("@messages");
  </while>
}
}
```




```
class SavingsAccount {
  public static void main(String[] args) {
    System.out.println("This is a bank account");
    System.out.println("Savings Account");
  }
}
```

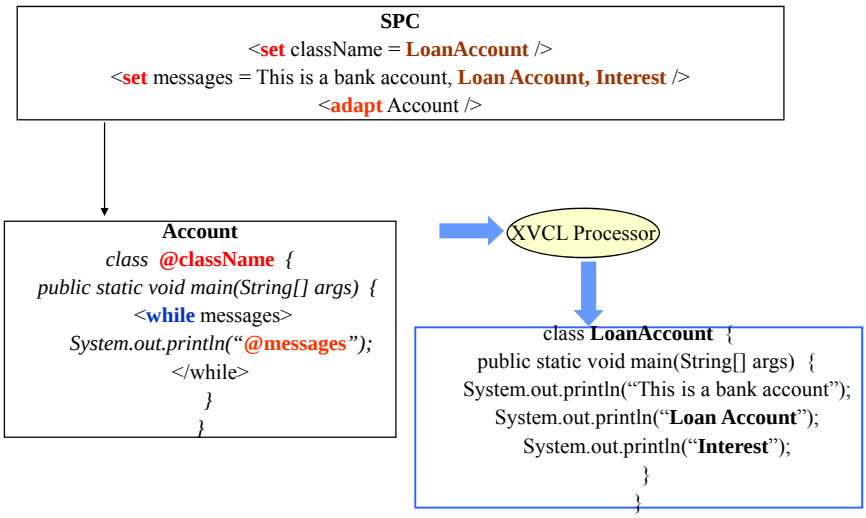

Copyright (C) 2010 Stan Jarzabek

Deriving Current Account



Copyright (C) 2010 Stan Jarzabek

Deriving Loan Account



Copyright (C) 2010 Stan Jarzabek

Deriving three Account classes

SPC

```
<set className = SavingsAccount, CurrentAccount,
                 LoanAccount />
<set common = This is a bank account/>
<while className>
<select option = className>
<option SavingsAccount >
<set messages = @common, Savings Account />
<adapt Account />
<option CurrentAccount >
<set messages = @common, Current Account />
<adapt Account />
<option LoanAccount >
<set messages = @common, Loan Account, Interest />
<adapt Account />
</select>
</while>
```

Account

```
class @className {
public static void main(String[] args) {
<while messages>
System.out.println("@messages");
</while>
}
}
```

```
class SavingsAccount {
public static void main(String[] args) {
System.out.println("This is a bank account");
System.out.println("Savings Account");
}
}
```

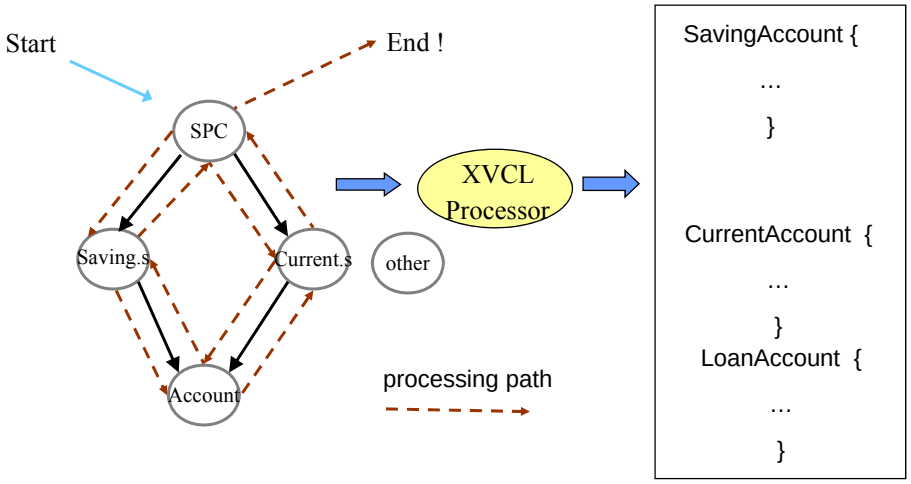
```
class CurrentAccount {
public static void main(String[] args) {
System.out.println("This is a bank account");
System.out.println("Current Account");
}
}
```

```
class LoanAccount {
public static void main(String[] args) {
System.out.println("This is a bank account");
System.out.println("Loan Account");
System.out.println("Interest");
}
}
```

XVCL Processor

Copyright (C) 2010 Stan Jarzabek

<adapt>-driven composition of x-frames



how does this compare to generics/templates so far?

generated classes

Copyright (C) 2010 Stan Jarzabek

Evolution of classes

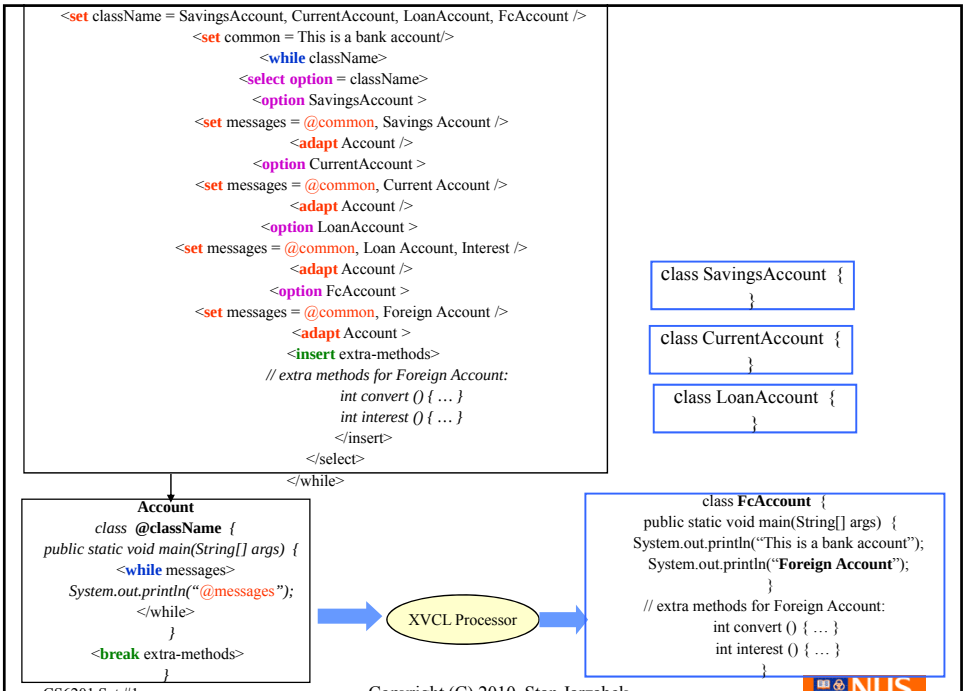
Copyright (C) 2010 Stan Jarzabek

Foreign currency account

- we need class FcAccount for foreign currency
- class FcAccount needs some extra methods as compared to other account classes

```
class FcAccount  {
public static void main(String[] args)  {
System.out.println("This is a bank account");
System.out.println("Foreign Account");
}
// extra methods for FcAccount
int convert () { … }
int interest () { … }
}
```

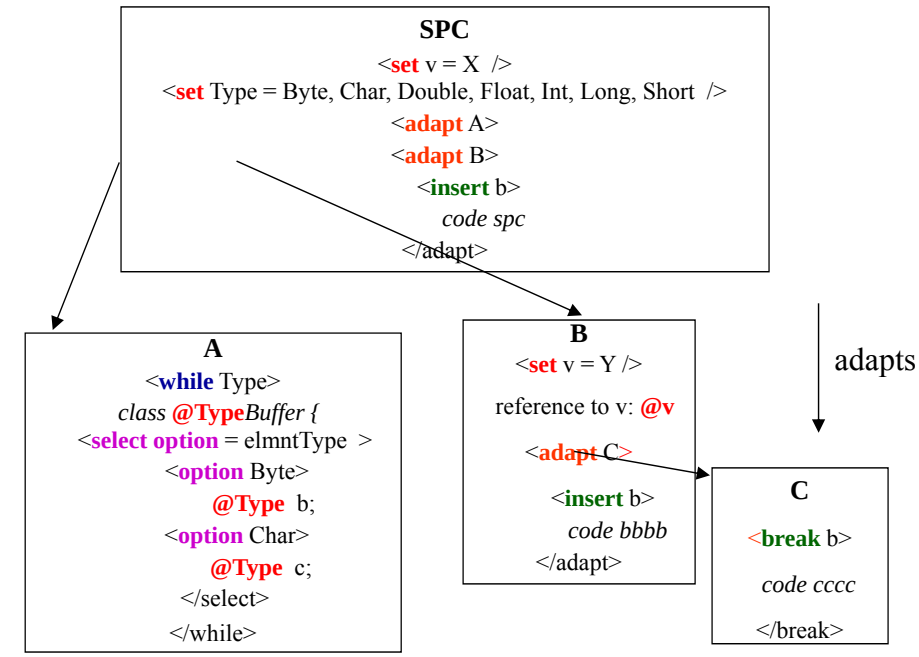

Copyright (C) 2010 Stan Jarzabek



Copyright (C) 2010 Stan Jarzabek

Summary of XVCL mechanisms

Copyright (C) 2010 Stan Jarzabek



Copyright (C) 2010 Stan Jarzabek

Processing rules

- the processor traverses x-framework in depth-first order, as dictated by <**adapt**>s embedded in x-frames
- the processor interprets XVCL commands embedded in visited x-frames and emits a custom program into one or more files
- x-frames are read-only. The processor creates and modifies a copy of the adapted x-frame and never changes the original x-frame
- customization commands are specified for each <**adapt** A>
- recursive adaptations are not allowed

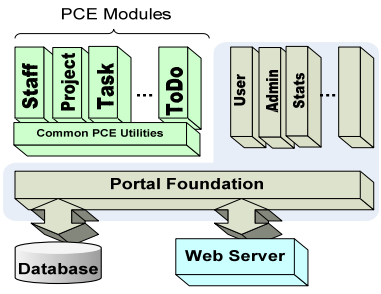
Copyright (C) 2010 Stan Jarzabek

Project Collaboration Environment (PCE)

Software Product Line

Copyright (C) 2010 Stan Jarzabek

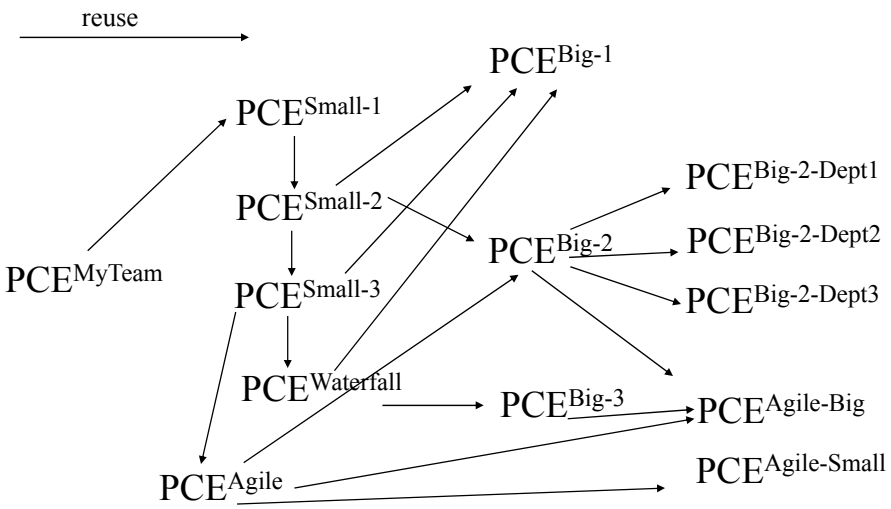
Project Collaboration Envir (PCE)



- PCE stores staff, project data, facilitates project progress monitoring, communication in the team, etc.
- e.g., Module Staff: allows the user to create, edit, and update data about staff members, assign staff members to projects, etc.

Copyright (C) 2010 Stan Jarzabek

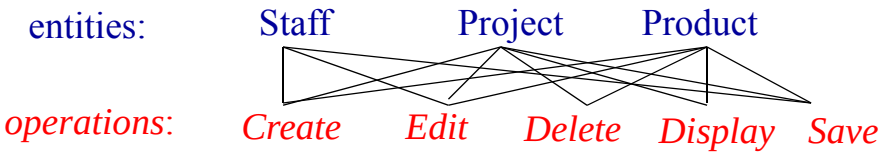
PCE product variants



Copyright (C) 2010 Stan Jarzabek

PCE domain analysis

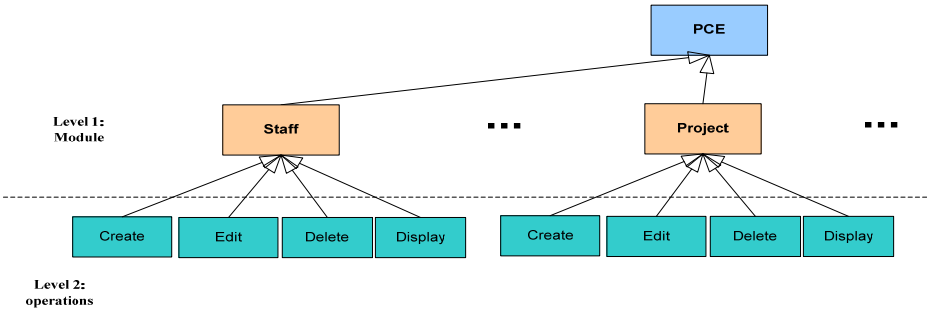
- analysis of requirements for many PCEs
- each PCE involves entities and operations



- PCE modules implement operations for various entities

Copyright (C) 2010 Stan Jarzabek

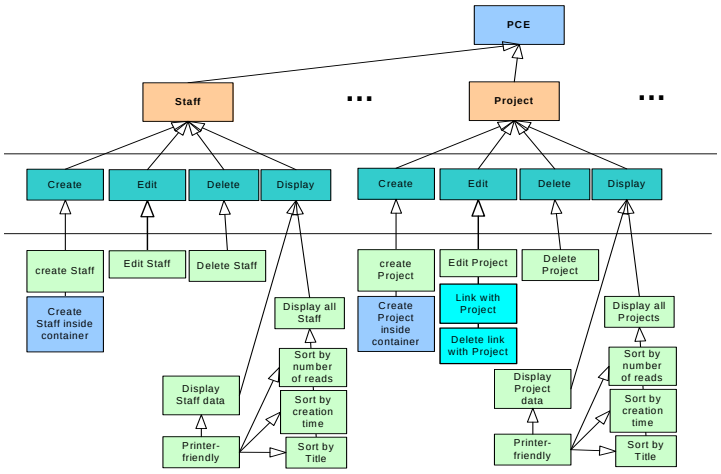
Modules for Staff, Project, …



More detailed analysis of PCE domain and design, next slide

Copyright (C) 2010 Stan Jarzabek

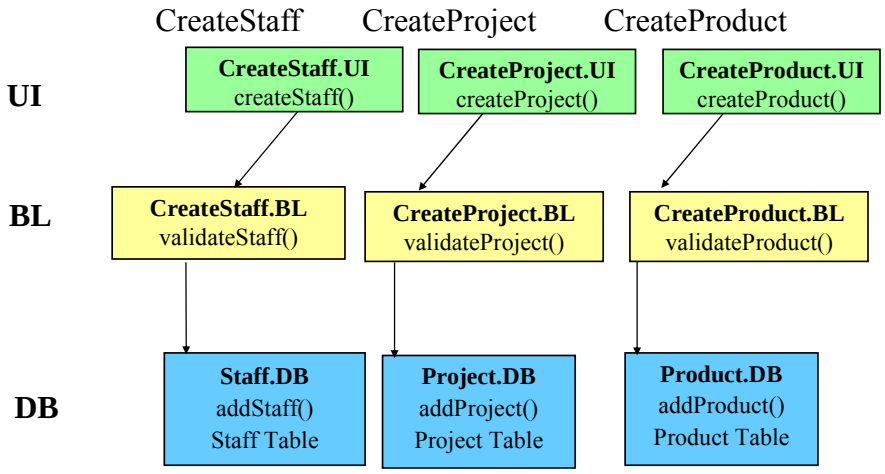
Modules for Staff, Project, …



- Operations *Create* for Staff, Project, Product, etc. are similar but also different

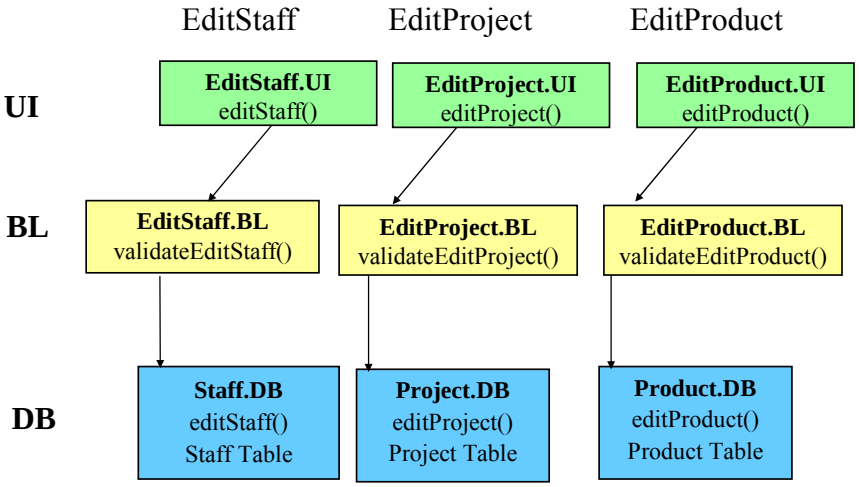
Copyright (C) 2010 Stan Jarzabek

Group of similar operations: Create[E]



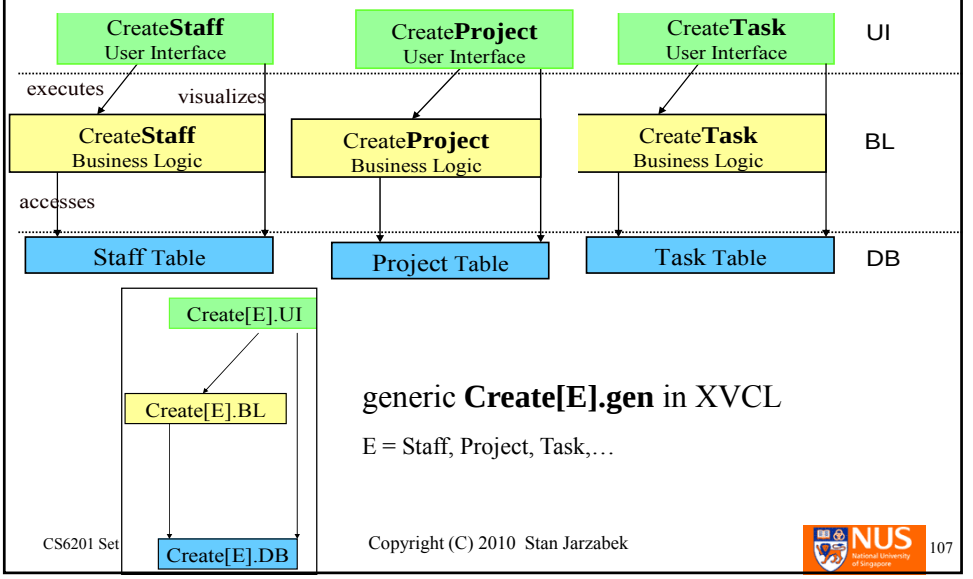
Copyright (C) 2010 Stan Jarzabek

Group of similar operations: Edit[E]



Copyright (C) 2010 Stan Jarzabek

Generic representation of component patterns in XVCL

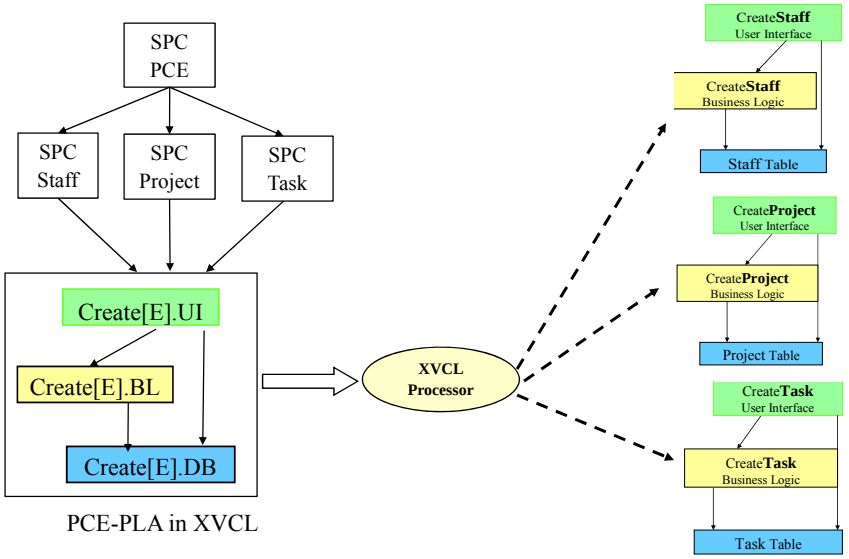


generic **Create[E].gen** in XVCL

E = Staff, Project, Task,…

Copyright (C) 2010 Stan Jarzabek

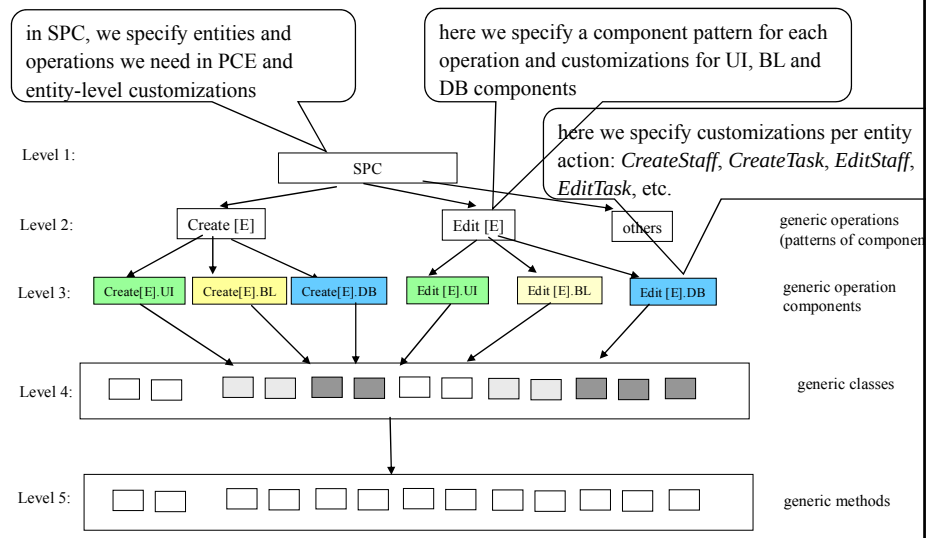
Generating Create[E] modules



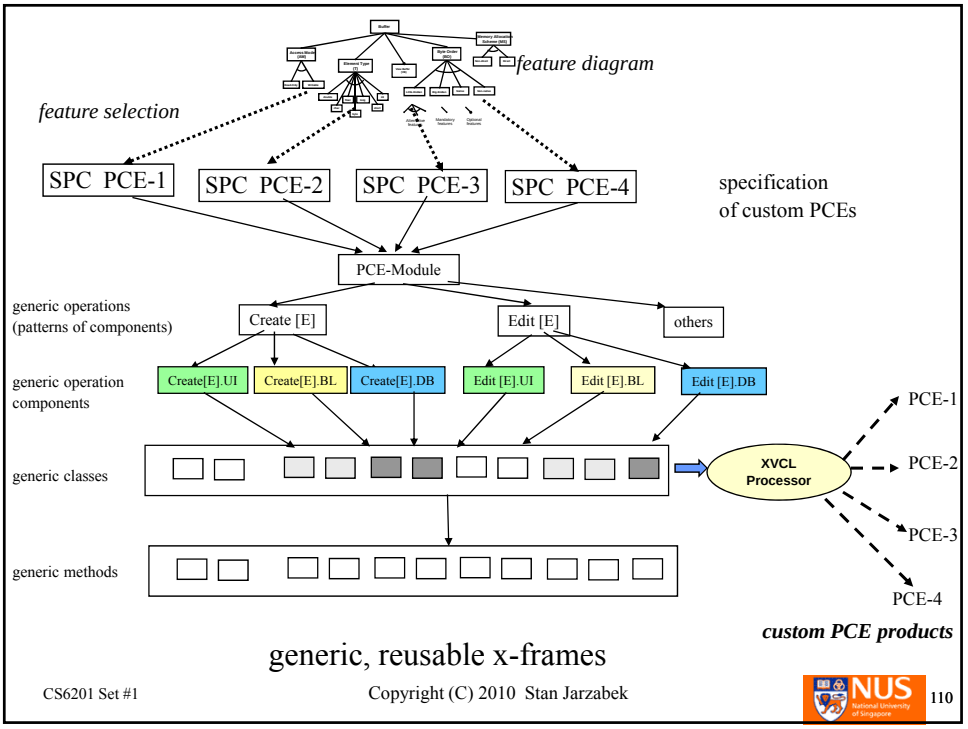
PCE-PLA in XVCL

Copyright (C) 2010 Stan Jarzabek

Generic PCE with XVCL



Copyright (C) 2010 Stan Jarzabek



Copyright (C) 2010 Stan Jarzabek

What can we achieve in XVCL way?

Experiences, Evaluation

Copyright (C) 2010 Stan Jarzabek

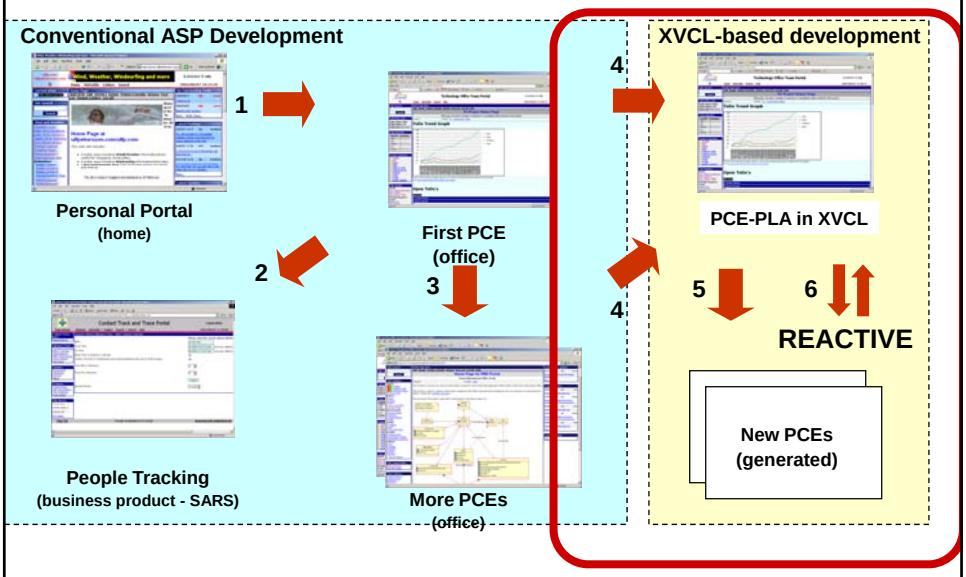
Web Portal in ASP/XVCL

by ST Electronics (Info-Software Systems) Pte Ltd

Details in : Pettersson, U., and Jarzabek, S. "Industrial Experience with Building a Web Portal Product Line using a Lightweight, Reactive Approach," *ESEC-FSE'05, European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Sept. 2005, Lisbon, pp. 326-335

Copyright (C) 2010 Stan Jarzabek

From PCE to PCE Product Line



Copyright (C) 2010 Stan Jarzabek

Experiences from ASP/XVCL project:

- STEE has built and maintains over 20 different portals
  - based on XVCL-enabled Product Line architecture
- Short time (less than 2 weeks) and small effort (2 persons) to start seeing the benefits
- High productivity in building new portals with XVCL
  - **60%** - 90% **reduction of code** needed to build a new portal
  - estimated **eight-fold** reduction of effort
- Reduced maintenance effort for released portals
  - for the for first nine portals, managed code lines was 22% less than the original single portal

Copyright (C) 2010 Stan Jarzabek

XVCL reuse capabilities

PLA design:

- Handle any product-specific customizations (like in case of common variation mechanisms)
- XVCL captures knowledge of product customization
  - no need to store component versions in CVS as they can be re-generated

Product derivation (reuse)

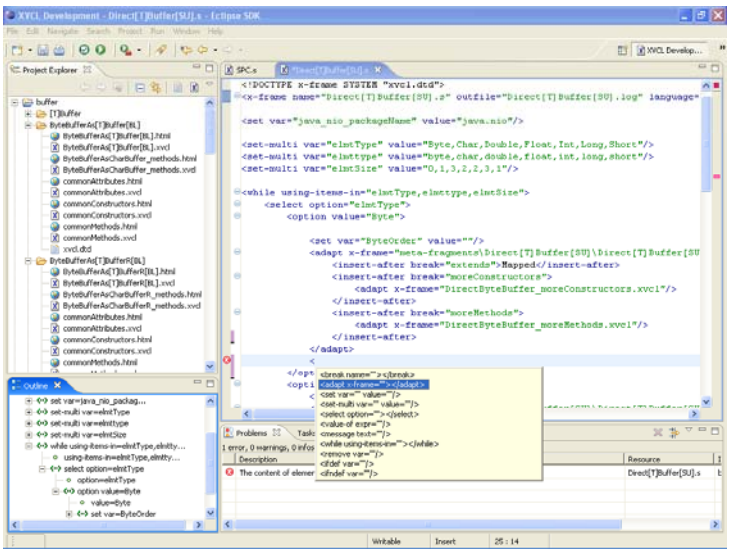
- Specify unique properties of a product separately from core components
- System-wide propagation of parameters, customizations for reuse
- Automation by XVCL Processor

PLA and product evolution

- Propagate changes of core components selectively to products
- Modify products without disconnecting them from core components

Copyright (C) 2010 Stan Jarzabek

XVCL Workbench



Copyright (C) 2010 Stan Jarzabek

Summary

- Use one variation mechanism instead of many
- Unrestricted customizations
- Automation of reuse
- Evolution of products and reusable components

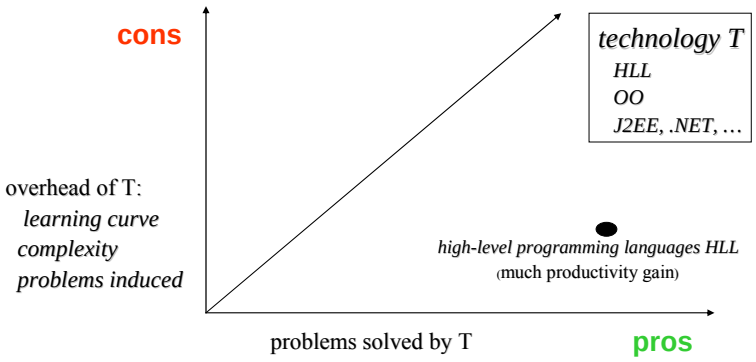
Copyright (C) 2010 Stan Jarzabek

Trade offs

- XVCL applied with good results:
  - only in small- to medium-size projects
  - agile development methods
- Integration with standard processes is a challenge
  - XVCL Workbench
  - technology transfer and methodological guidelines

Copyright (C) 2010 Stan Jarzabek

Impact of technology on productivity



- technology impact – benefit: balancing cons and pros
  - impact on maintenance – a critical factor in technology evaluation

where do we place technologies on this scale?

Copyright (C) 2010 Stan Jarzabek

Q & A

--- The End set #1---

Copyright (C) 2010 Stan Jarzabek