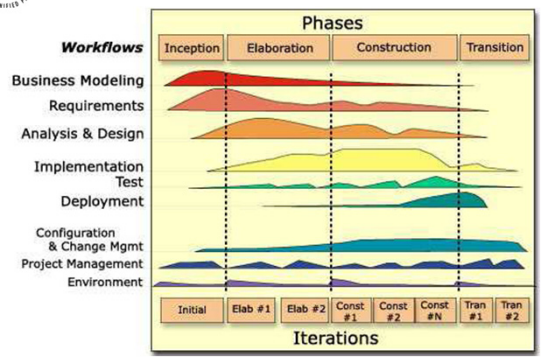
Experiencing RUP

1



2

A naïve view of RUP

- Rational Unified Process (RUP) is a process model that defines:

  Who

  needs to do

  What

  and

  When *it must be done by*

  in order to achieve

  Goals defined in Project Vision

3

RUP

- Extremely detailed guidelines
  - 30 different "roles" for team members (e.g. "developers" sub-classified as architects, designers, user-interface designers, capsule designers, database designers, implementors, integrators)
  - Each role characterized by
    - List of activities
    - List of artefacts that must be produced (e.g. "user-interface designer" must produce a "navigation map" and a "user interface prototype")
  - Each "discipline" (Requirements, Deployment, …) characterized by a detailed workflow model

4

Guidelines!

- Project groups aren't expected to use all the RUP stuff in any particular project.
- Groups follow those RUP guidelines that are appropriate given the scope and nature of the project that they are working on.¶

¶One of the first tasks is development of the "Development Case" which identifies those aspects of RUP that will be used.

5

A RUP example

Experience RUP vicariously by reliving the experience of the PSP-tools team!

6

"Textbook"

- "Software Development for Small Teams: A RUP-Centric Approach", G. Pollice, L. Augustine, C. Lowe, J. Madhur. *(PSP-tools team)*
- ***Not a textbook!***
- An experience report.
- As an experience report – quite useful.

7

Why did Pollice et al write it?

- It is a challenge for project managers (and, more generally, software developers) to *find the development process that works best for their projects.*
- Ideally
  - Try each of them on equivalent projects
  - Pick the best
- Impossible!
- So, instead, take guidance from published experience reports.

8

"Software Development for Small Teams"

- Small team
- Real, though relatively simple, application
  - Essentially a system for recording work records – how much time, what task, etc, etc.
  - Has a few reporting functions that generate summary statistics.
- Team using (slightly simplified) version of Rational Unified Process¶
  - RUP more typically used with somewhat larger projects
- Team's records (the book) detail an experience of RUP, *hence you can share their experience and so decide if RUP is good for you*

¶ (Also slightly "contaminated" RUP, they drop into lighter weight XP practices in places.)

9

Not a student project in style of CSCI321!

- Team:
  - Diverse
  - All experienced
  - All familiar (to varying degrees) with development language and development tools.
  - Some having limited experience in problem domain.
  - All have participated in a number of projects
  - All had been employees of Rational and so knew about RUP and Rational's tools (UML tool, project management tool, etc)

10

Not a student project in style of CSCI321!

- Team:
  - Each individual has specializations wherein they are competent.
    - Gary – "project lead" - business systems degree, software engineering grad school, familiar with problem domain (PSP), 30 years experience, different methodologies, limited RUP experience
    - Liz – "technical writer (also tester, tool engineer)" – 20 years experience, comp. sci. degree, programmer before becoming tech writer/tools engineer/tester, interested in "light weight" process
    - Chris – "? Developer/tester" – 15 years, C/C++, Windows, Windows GUI, wants more Java, has heard of RUP
    - Jas – "? tester" – had some experience with RUP in a larger scale project, knows a little about problem domain (PSP)
    - Russell – "? Customer/Business manager" – person who wants a usable PSP-tools product

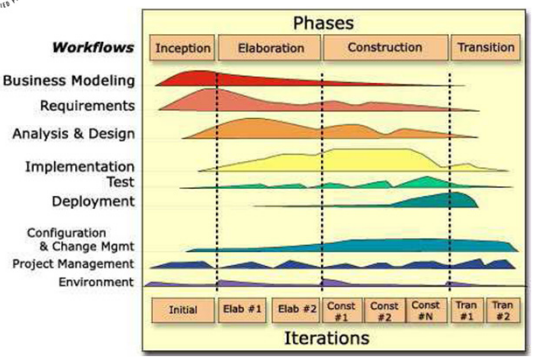
11

"Project Guideline 1"

- *Violated in all student projects!*
- Get the right mix of people on your project
  - Senior members who lead and teach junior members (Gary/Chris)
  - Senior members have experience on similar projects (Gary)
  - There are team members with existing technical expertise in all critical areas

    (actually they had some problems here, no real database guru).

12



13

Four phases

- PSP-tools group describe their activities for four phases of RUP
  - What they did.
  - What parts of RUP got used.
  - *What they should have done.*
- Inception (1 week)
- Elaboration (3 weeks)
- Construction (8 weeks)
- Transition (3 weeks)

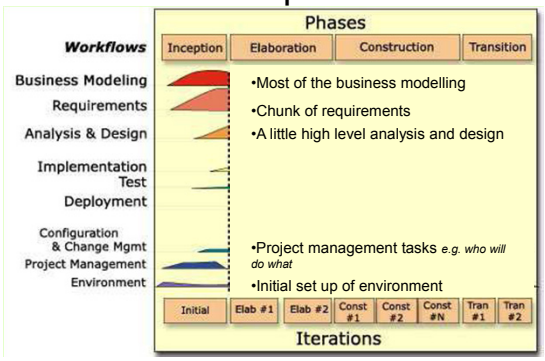
14

Actually PSP-tools group had a pre-inception phase

- Meet as group,
  - Introduce selves,
  - Explain what you view as your strengths and weaknesses
  - Discuss roles that you might take on later
  - Begin to notice possible weaknesses of group (hence "risks" that will be identified in the inception stage)
- Get some idea of project
  - PSP-tools group all did
    - A little reading about PSP
    - A couple of exercises on manually recording times on tasks as if they were using PSP
- This "pre-inception" phase is not standard
  - In real world, most of group will already have worked as colleagues on other projects in a company and so introductions and identification of potential roles already accomplished
- GOOD IDEA to try something like this when starting your CSCI321 project or any large group project component in another subject.

15

Inception



16

Inception

Inception – iterations?

- Usually Inception phase has just one "iteration"
  - Might get second iteration if "risk" analysis of first proposal suggests it will fail, could then try again with a changed vision for proposed product
- Style of RUP project here is similar to old waterfall model – business model, requirements, analysis and design are largely "completed"
- Some agile methodologies would already have code being generated and its behaviour being evaluated by the "customer", final functionality of product being left incompletely specified

17

Inception

PSP-tools : Inception

- "Development case"
  - Really part of the "project management" discipline
  - Decide how to use RUP
    - "A development case is a brief description of how you should apply the RUP, what artefacts to produce when and with what formality, how to map roles to people in your project, and so on. You typically produce a development case for a project or a type of project. The development case should be very short, ideally only a couple of pages long. Rather than duplicating information in the RUP, you can link to activities, artefacts, and roles in the RUP from the development case."

18

Inception

PSP-tools development case: artefacts

Artifacts	How to use				Review details	Tools used	Responsible
	Inception	Elaboration	Construction	Transition			
Supplementary Specification	C	M			Formal	ReqPro & MS Word	System Analyst
Use-Case Model	C	M			Formal	ReqPro, Rose & MS Word	System Analyst
User-Interface Prototype		C					User-Interface Designer
Vision	C	M			Formal	ReqPro & MS Word	System Analyst

We will create the use-case models in the inception phase modify them during elaboration; naturally, the "Vision" thing is created at inception. We allocate responsibility for these artefacts to roles as shown.

PSP-tools : development case

	Liz	Chris	Jas	Gary
System Analyst			X	
User-Interface Designer		X		
Data Designer				X
Software Architect				X
Integrator				X
Implementer		X		X
Test Designer		X	X	X
Tester	X	X	X	X
Deployment Manager				X
Technical Writer	X			
Configuration Manager				X
Project Manager				X
Process Engineer				X
Tool Specialist	X	X	X	X

Initial "who does what" map; define the roles that group members will fill.

PSP-tools : Inception

- Start developing environment (environment discipline)
  - PSP-tools group picked up some "collaborative workspace" product
    - Think of it as structured bulletin board

PSP-tools : Inception

- "Iteration plan" for rest of inception phase (management discipline)
  - Artefact, who responsible, when due

Artifact	Due	Responsible	Comment
Initial project plan	10/11/2002	Gary	Through Elaboration, reviewed and agreed upon
Vision and product feature requirements	10/13/2002	Jas, Gary	Reviewed
Supplementary requirements	10/13/2002	Jas	Deferred until needed
Tools environment	10/15/2002		
Clear Case	10/12/2002	Gary, Liz	VOBs
Test environment		Chris	Not completed
Requirements		Jas	Deferred
Project Web site		Gary	Deferred
Initial use-case model	10/15/2002	Jas	Actors and use cases with brief descriptions
Initial risk list	10/15/2002	Gary	Reviewed and understood
Test plan	10/15/2002	Chris	Draft reviewed and agreed upon
Elaboration Iteration Plan	10/15/2002	Gary	Reviewed and agreed upon

PSP-tools : Inception : the Vision

- What problem are we trying to solve? (Problem Statement)
- Who are the stakeholders?
  Who are the users?
  What are their respective needs?
- What are the product features?
- What are the functional requirements? (Use Cases)
- What are the non-functional requirements?
- What are the design constraints?

PSP-tools: Problem statement

The problem of	developing software in a predictable and reliable manner
affects	the management of software projects. Specifically, developers are not able to predict reliably how long it takes them to perform development tasks with acceptable quality, which makes it impossible to effectively plan a project;
the impact of which is	users and managers are never sure whether the produced software will meet its requirements, whether the software will be error-ridden, and whether the software will be delivered on time.
A successful solution would be	for developers to become more self-aware of what they do (i.e., the process they follow), how they spend their time, and the kinds of defects they find in their work. Through this awareness, developers would become better estimators.

"describe the problem, who it affects, how it affects them, and what type of solution would ease the pain."

PSP-tools: “Position statement”

For	software development teams
who	need to better understand how and when defects are introduced into their products.
PSP Tools is a	performance metrics gathering and reporting tool
that	helps developers gather and analyze software development metrics.
Unlike	the alternative of failing to gather the data or trying to track it manually.
our product	helps you gather data unobtrusively and provides objective feedback that allows you to improve your individual and team performance.

If you could develop software that would solve the problem described in the Problem statement:
who would buy it
and what would make it unique

Getting the money …

- Sometimes, “Vision” thing is created by entrepreneurial developers who must then sell it to venture capitalists.

- PSP-tools team had been commissioned to develop the software
- Most of projects you will be working on in future will similarly have been commissioned either by marketing seeing some product niche or by another part of company having some perceived need for software.
  - In these cases, “Vision” thing’s role is to establish some common understanding of project for development team and commissioning customer;
    Vision will be fleshed out as get into requirements

Identify stakeholders

- Customers
- Management
- Team
- Others
  - Some projects will need to satisfy government regulations etc (e.g. privacy of data), then someone in organization will have to act as a proxy for government, checking project’s compliance
- PSP-tools stakeholders were the team members and the customer Russell

Identify some features

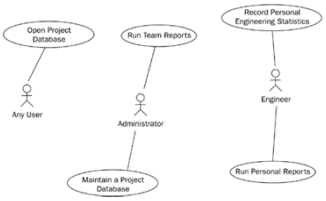
- PSP-tools:
  - “record personal statistics”
    - Time
    - Software defects
    - Source code size
  - “reporting”
    - Personal reports
    - Team reports
  - Viewing
  - Statistics
  - …

Sketch some high level use cases

- Software Engineer will use PSP-tools to:
  - Create project
  - Enter data
  - Count items
  - Use on line help when needed
  - Create report
  - …
- Process administrator will use PSP-tools to:
  - Configure system
  - Configure project
  - Install PSP-tools
  - Use on line help when needed
  - …

Iterate a bit with your initial use cases!

- PSP-tools team reduced that initial use case diagram down to this:



Identify non-functional requirements

- Typical things:
  - Must be implemented using *AAA* and *BBB* running on the *CCC* operating system (because those are standards for the company)
  - Must run reasonably¶ on a machine with following configuration …
  - Must be built using a process that complies with standard *xxx*
  - Must have support for internationalization
  - Must meet following security requirements limiting access to personal data …
  - Must have 99.999% availability 24/365
  - Must accommodate visually impaired users
- PSP-tools guys don't appear to have identified any non-functional requirements during the inception phase.

¶Take care when you see a requirement like that. What seems "reasonable" to you might seem "pathetically bad" to me! You must get stakeholder to define a quantifiable measure of "reasonableness"

James Gibson, the only real software engineer to have worked in this department, says I should use "shall" rather than "must"! So quaintly archaic!

31

Inception

Non-functional requirements

- Generally aspects such as
  - Usability
    - Accessibility, aesthetics, consistency, …
  - Reliability
    - Scheduled downtimes, recoverability after various disasters
  - Performance
    - Throughput, response times
  - Supportability
    - Ease of configuration, deployment, …
  - Design constraints
    - *E.g. must work with hierarchical database*
  - Implementation constraints
    - *E.g. Use C++*
  - Interface constraints
    - *E.g. Must upload data from a Blackberry*
  - Physical constraints
    - Usually only for military and embedded systems – must run on hardware that fits in this box and uses this much power

32

Initial Project Plan

- Prioritize requirements
  - Remember two of the "Spirit of RUP" rules
    - Baseline an executable architecture early on
    - Stay focussed on executable software
  - High priority on getting some minimal subset of overall project running.
  - Another high priority is getting some preliminary work done in area of high risk (areas where difficulties can be expected or where team knows it lacks significant expertise); starting these early helps quantify risks (and in worst case means project can be abandoned before too much effort expended)
  - Customer ranking of functionality
  - Other priority constraints – e.g. feature X depends on feature Y, therefore feature Y must have higher priority so it gets done first
- PSP-tools: priorities for use cases
  - 1) create database; 2) add data; 3) report data

33

Inception

Initial Project Plan

- Once you have prioritized use-cases, assign them to iterations in elaboration and construction phase
  - A minimal set of use cases, just enough to start an "executable architecture", will get at least partially implemented in elaboration phase
    - Some beginnings of user interface
    - Some minimal functionality
    - For use in subsequent iterations of negotiation with "customer"/"users" needed to clarify requirements and accommodate changed perceptions
  - Other use cases
    - Assign them to iterations in construction according to priority order; an iteration will deal with some number of use cases

34

Inception

"Time boxing" iterations

- This is typical.
  - Iteration-x, must complete in 3 weeks
- If, when engaged in Iteration-x, you find that it isn't going to be completed, you get approval for lowest priority feature assigned to that iteration to be pushed into a later iteration.
- As PSP-tools was a more "amateur" project (they were working in their spare time), they didn't do time-boxing (to be honest, they really didn't plan the iterations for the construction phase). They still did feature culling.

35

Initial Project Plan

- Other artefacts
  - Initial plan must also specify phase and iteration in which other artefacts will be delivered
  - Initial plan just defines order, as plans revised dates get added.

36

Inception

Initial Project Plan

Milestone	Artifact	Completion Dates		
		Planned	Revised	Actual
Iteration 1 (Inception)		10/15/2002		10/15/2002
	Initial project plans	10/11/2002		10/12/2002
	Inception iteration plan	10/1/2002	10/3/2002	10/5/2002
	Working environment set up, including the project repository and project Web site	10/15/2002	Taken out of Inception. Only partial environment by end of Inception.	
	Vision and product feature requirements	10/13/2002		10/14/2002
	Supplementary requirements	10/13/2002	N/A Removed from project until needed.	
	Initial use-case model	10/15/2002		10/15/2002
	Initial risk list	10/15/2002		10/15/2002
	Test plan	10/15/2002	Draft	10/15/2002
	Iteration Assessment	10/15/2002		10/15/2002
Iteration 2 (Elaboration)		12/15/2002		
	Updated project plan			

- PSP tools initial plan
  1. Plan done by 10th Oct
  2. Environment set up …
  3. Vision complete by …
  4. Supplementary requirements
  5. Initial use cases by ..
  6. Risk list
  7. Test plan
  8. Finish "Inception Iteration"

Identify Risks

- This is the task that is impossible for students who usually end up relying on undergraduate "humour" (*"struck by lightning", "thrown out of apartment", "break with girl-friend/boy-friend/both", "hard assignments in other core subjects", …*)
- In real-world projects:
  - Business manager and project team leader list things that have, in their past experience, resulted in disruptions to similar projects.
- Typical things
  - Project requires arcane knowledge or obscure technical experience (e.g. "must utilize the IMS hierarchical database") – risk is that you won't find anyone with the requisite knowledge
  - Project requires development environment, or methodology, that is new to most members
  - Constraints on team members ("no recruitment, must use existing employees") may constitute risk ("difficult people" etc)

Identify risks – and propose countermeasures

- No good just identifying risks, have to outline how they will be reduced.
- Examples:
  - Use IMS? !
    - Convince customer to allow use of relational database?
    - Separate out a distinct sub-process that cruds¶ with IMS, employ a consultant to implement it
  - New Environment, New Methodology?
    - Pay for training courses
  - Difficult team member?
    - Invoke help from the gods

¶crud – create, read, update, delete

PSP-tools

- They did a little more on setting up environment
  - Chose an IDE
  - Chose a code-versioning tool
  - Chose testing tools
  - Chose tools to record management information such as the plans (could just be a spreadsheet or even a word-processor document)

PSP-tools : end of inception

- Artefacts delivered
  - An initial project plan
  - Vision endorsed by the stakeholders
  - Programming guidelines ¶
  - Initial requirements
  - The development environment set up and ready for the remainder of the project
  - A test plan ¶
  - An initial Risk List
  - The iteration plan for the Elaboration iteration ¶

¶Book doesn't contain any details of these in its "Inception" section

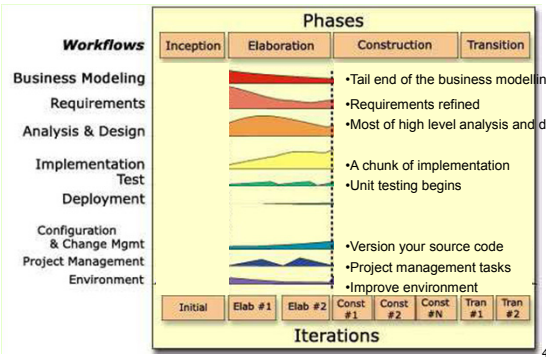
End of RUP inception – the Lifecycle Objective Milestone

- Conventional RUP requires an evaluation point – the Lifecycle Objective Milestone – at end of inception.
- This should verify:
  - Stakeholder concurrence on scope definition and cost/schedule estimates.
  - Requirements understanding as evidenced by the fidelity of the primary use cases.
  - Credibility of the cost/schedule estimates, priorities, risks, and development process.
  - Depth and breadth of any architectural prototype that was developed.
  - Actual expenditures versus planned expenditures.
- Some of these elements not relevant to PSP-tools as there were no budgets, costs, etc (it really is more like an open source collaborative project than a real costed industry project)

Inception

- You can vary the formality of RUP for inception
- You can make it like Waterfall
  - Complete business modelling
  - Work through all requirements
  - Hand over to next phase
- Or more like XP
  - Minimal requirements
  - No "analysis=paralysis"
  - Start implementing with "user" on hand
    - *"Is this what you want?"*
    - *"Is it better now?"*
    - *"What do you want next?"*

Elaboration



Elaboration - aims

- Not yet the product!
- Aims –
  - A "stable architecture"
    - No more radical changes (*"Hey, I've just had a great idea, lets make it web-based 3-tier client-server instead of a Unix shell-script!"*)
    - Lesser changes are permitted in later phases
  - Use cases elaborated with key scenarios worked through
  - Detailed plan for iterations in subsequent construction phase
  - Development environment effective and stable
  - The beginnings of executable software
  - The start of automated re-testing system for subsequent additions and changes
- Success of elaboration to be evaluated at end when reach the Lifecycle Architecture Milestone

Elaboration

- Artefacts may include
  - Updated versions of
    - Vision Document.
    - Risk List.
    - Software Development Plan
    - Iteration Plan.
    - Use-Case Model.
  - Supplementary Specifications
    - Finalized documented version of the non-functional requirements
  - Prototypes
    - explore software ideas
    - demonstrate specific behaviour.
  - Software Architecture Document—
    - Logical View
    - Use-Case View
    - Process View
    - Deployment View
  - Design Model.
  - Implementation Model
    - A collection of components, data, and subsystems that express the product design.
  - Project Specific Templates and Tools

PSP-tools: Elaboration Iteration Plan

Artifact	Due	Responsible	Comment
Updated project plan		Gary	Through first construction iteration, reviewed and agreed upon
Unit test harness		Chris	Installed and team members trained
Acceptance tests		Jas, Liz	Defined with initial scripts and run for implemented functionality
User interface prototype		Chris	Ready to be enhanced for the product, approved by customer
Software Architecture Document		John	With preliminary design model
Builds and components		John, Chris, Gary	
Test Plan		Chris	One page, describing the general approach
Unit tests		John, Chris, Gary	As per test plan
Documentation and support plan		Liz	Initial draft
Data model		John	Schema defined

Who is John?
Oh, he withdrew from CSCI321 about week-7 of session-1.
We had to finish without him.

Note defining tests – some members started thinking about acceptance tests (use the system to perform tasks – "black box" testing), others thinking about unit tests ("white box" Testing)

PSP-tools : elaborate the use cases

- Fill in some detail
  - HOW?
    - Based on experience with similar things in other projects
    - Guessing
- They initially created text descriptions

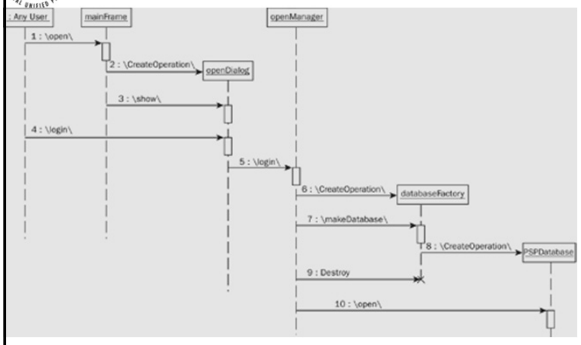
Any User	PSP Tools
1. Select **Open…**.	
2. Specify a project database and select **OK**.	3. Open the project database. If an error occurs, display an appropriate error message and end this use case.
4. Enter the user name and optional password and select **Login**. Select **Cancel** to end this use case with no state change.	5. Ensure that the user name and password (if present) are for an existing member of the database. If not, display an appropriate message and allow the user to be added to the database. If the PSP User decides not to add the new user, end the use case; otherwise, proceed with **Alternate Flow of Events: Add New User to Project Database**. If there is an existing user, proceed with the next step.
	6. Make the database the current database for the user. Close any open database. Display the appropriate view of the data in the project database for this user.

"Open project use case"

This is their version of "scenarios"

And MAGICALLY classes appear

- Gary
  - project lead,
  - lots of experience on similar projects
- Draws up an initial UML sequence diagram



Classes! ?!

- "mainFrame"
- "openDialog"
- "openManager"
- "databaseFactory"
- "PSPDatabase"

Now how did we get to those classes?

- Magic.
- The textbooks **NEVER** explain this.
- It is probably the hardest part of this stage of development.

They aren't classes anyway.

- What Gary has done is introduce a number of "Categories"
  - category: *a priori* conceptions applied by the mind to sense impression, or relatively fundamental philosophical concepts
- Gary has worked on similar applications.
- Gary knows that
  - There will be a control element that picks up a "Do create database" command from some form of GUI, so he infers that there is a something that he declares as a "mainFrame" object
  - This will result in display of a dialog where user enters details of database that is to be created – hence Gary's openDialog
  - Input data will be used
    - Firstly to confirm authorization (name/password checks maybe)
    - Then to parameterize the code to create the tables.
    - Garry is guessing here and picks
      - openManager, databaseFactory, and PSPdatabase

Lots of use of previous experience

- What is this "databaseFactory"?
- There is nothing in use case's text description that suggests the presence of any such class!
- Gary is using a "design pattern" with which he is familiar
  - *"Factory Method" : define an interface for creating an object but let subclases decide which class to instantiate*
- Gary has used something similar in another project and has just assumed it will be relevant here.

Use previous experience

- There is nothing in use case, or the text description that suggests the presence of this factory class!
- In fact, its occurrence will probably turn out later to be wrong;
  - at this stage they imagined that they might be creating different kinds of database structures – data for PSP-level-0 differs from data needed for PSP-level-1 so maybe they needed to created different kinds of PSPDatabase object.
    So, "Factory" – makes it possible to create different kinds of database object given parameter data
  - They will probably simplify that later!

55
Elaboration

Don't expect these "classes" (categories) to survive through many iterations

- These won't be your actual classes.
- A few more iterations are necessary to get your initial class definitions in terms of "owns" and "does"
- Caution:
  - Working from use cases often leads to bad designs where you have
    - a "main" class that has essentially all the program's business logic
    - Some "entity" classes that are really just structs representing rows in a relational database
    - A few minor wrapper classes, e.g. a "wrapper" around the datastore code that allows switching between file-based and relational-table style data repositories.
  - Good designs
    - business logic assigned appropriately to meaningful application specific business classes
    - Use of design patterns like "Command", "Observer", "Flyweight", "Singleton" which suggest groups of classes useful when handling tasks such as those that emerge in use cases

56
Elaboration

No previous experience?

- So how do you get your classes (categories) if you don't have Gary's experience?
- Try the "Classes Responsibilities Collaborations" game
  - Group members play act roles of things in program
  - Try exploring how user requests might be handled
  - Note
    - the things you need to know for your role,
    - The requests that you receive
    - The requests that you make to others

57
Elaboration

Classes Responsibilities Collaborations Game – pick your roles

- Group members play act roles of things in program
  - Member-1 "I'm the database; I can crud a table; don't ask me to do anything else. I am unique. I am a singleton."
  - Member-2 "I'll manage the database; creating new PSP-project tables, accessing existing tables. So I am THE DATABASEMANAGER. I think I'm a singleton, but there might be more than one of me.";
    "I will also pretend to be another object – something that verifies 'log-ins' to make sure that only authorized people create of use tables. In this role, I'm definitely a singleton."
  - Member-3 "I like being schizophrenic. I'll pretend to be each of the GUI data entry forms and responses. So I represent several objects from several different classes at different times"
  - Member-4 "I'll take the role of listening for inputs on GUI forms – from my friend member-3. I am probably several objects – listeners of different classes for different forms. My role as listener is to get the work done via Member-2 and/or member-1. I'll tell member -3 what results to display"
  - Member-5 "I'm main(). I create all you other guys and set links so you can talk to one another. Then my thread sleeps. Don't ask me to do anything."
  - Member-6 "I suppose I'm the user"

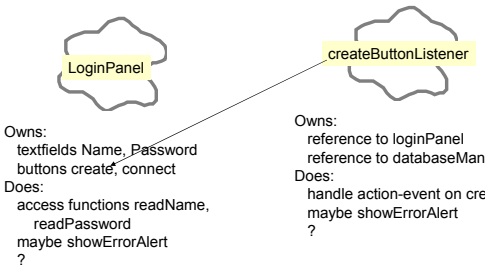
58
Elaboration

Classes Responsibilities Collaborations Game – *play your roles*

- Member-6 "I want to create a database"
- Member-3 "I suppose I'd better be a "LoginPanel" object; somehow #5 had better arrange that I'm displayed first when the program starts. I show name and password fields and create-database, connect-to-database buttons"
- Member-6 "I fill in my name in and hit #3's create button"
- Membe-4 "I'm a createButtonListener object, #5 arranged that I was waiting for this button press; I ask #3=LoginPanel for the name and password; there is no password; shucks; I think I can probably put up an error alert dialog; I'm not going to do anymore work now; if I can't put up the error alert, then it will have to be a responsibility of #3"

59
Elaboration

Guessing classes



Record the things you own, the things you do, who you talk to, what you ask them to do

60

Classes Responsibilities Collaborations Game – play your roles

- Member-6 “I close that stupid error alert!
  I fill in my name in and my password.
  I hit #3’s create button”
- Member-4 “I’m a createButtonListener object,
  I ask #3=LoginPanel for the name and password; I ask #2, in her guise of loginChecker to check the name password combination”
- Member-2 “In my role of loginChecker, I ask #1 database to retrieve the encrypted password for user with name=… from the users’ table”
- Member-1 “I’m the database. I crud. I return data”
- Member-2 “I encrypt supplied password, compared with crudded data, they match, I return OK”
- Member-4 “I’m still the createButtonListener, I resume my work on the login, I now ask #2 DATABASEMANAGER to create database”
- Member-2 “Er, problem here, you haven’t told me anything about the new database table, should there be some additional input data? Should login be a separate step from create, using different GUI panels maybe?”

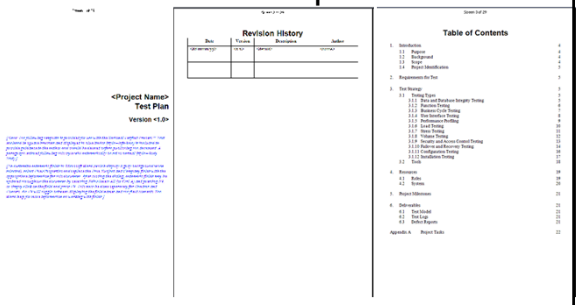
Somehow, magic occurs

- Based on previous experience, or CRC role-play, you do get
  - Some initial classes
    - Owns …
    - Does …
    - Talks to …
  - And some guess at an interaction diagram for the use case that you are working on

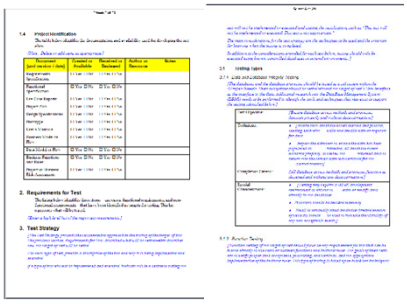
Elaboration - testing

- Gary + Chris to do most of programming
  - Decided to do it xP style with test driven development and unit testing
- First write the test plan
  - “*Let’s follow the RUP guidelines*”

RUP test plan



RUP test plan



29 pages like this!

RUP test plan template

Data and Database Integrity Testing

- Test Objective:
  - *Ensure database access methods and processes function properly and without data corruption.*
- Technique:
  - *Invoke each database access method and process, seeding each with valid and invalid data or requests for data.*
  - *Inspect the database to ensure the data has been populated as intended, all database events occurred properly, or review the returned data to ensure that the correct data was retrieved for the correct reasons*
- Completion Criteria:
  - *All database access methods and processes function as designed and without any data corruption.*
- Special Considerations:
  - *Testing may require a DBMS development environment or drivers to enter or modify data directly in the databases.*
  - *Processes should be invoked manually.*
  - *Small or minimally sized databases (limited number of records) should be used to increase the visibility of any non-acceptable events.]*

RUP test plan

- Similar detailed templates
  - Data and Database Integrity Testing
  - Function Testing
  - Business Cycle Testing
  - User Interface Testing
  - Performance Profiling
  - Load Testing
  - Stress Testing
  - Volume Testing
  - Security and Access Control Testing
  - Failover and Recovery Testing
  - Configuration Testing
  - Installation Testing

PSP-tools

- Er – maybe not, that stuffs for bigger projects!
- PSP-tools Test Plan
  - *Each class will have corresponding unit tests. No code will be checked into version control unless all unit tests pass.*
  - *Acceptance tests will be run and will pass before any software is delivered to the customer.*

PSP-tools : acceptance tests

- They started sketching these
  - Really just a variant on the existing use-case description with a few annotations.
  - A guess as to how eventual user might interact with some interface element

Actions:
1. Select **Open Project Database**.
2. At file dialog prompt, specify a project database.
   a. Already a member — you're in.
   b. Not a member — add yourself.
   c. Not a member — change your username.

Preconditions:
At least two databases exist.
App is started.

Tests:

Test ID	Select database	Member?	Expected Result	Next step
2.0.1	Select database: Acc/Test1	Yes	Display view of data for DB1.	Test 2.0.2
2.0.2	Select database: Acc/Test2	Yes	Display view of data for DB2.	Leave app open
2.0.3	Select database: Acc/Test1	No	Prompt for next step	Opt to add self. Leave app open
2.0.4	Select database: Acc/Test2	No	Prompt for next step	Opt to switch user id. leave app open
2.0.5	Select database: Acc/Test2	No	Prompt for next step	Opt to cancel, exit app.

Architecture

- PSP-tools persons admit they cannot give a good definition of architecture (page 83).
- "Highest level concept of a system in its environment"
- Actually, multiple levels
  - Process and inter-process communication level
    - What processes make up system?
    - What machines do they run on?
    - How do they intercommunicate?
    - What messages are exchanged?
  - Process
    - Subparts
      - GUI interface, Database interface, Control code, Business logic parts, …
  - "Subpart"
    - Principle classes and packages

PSP-tools is a simple, single process application; architectural effort reduced!

Architecture – PSP-tools

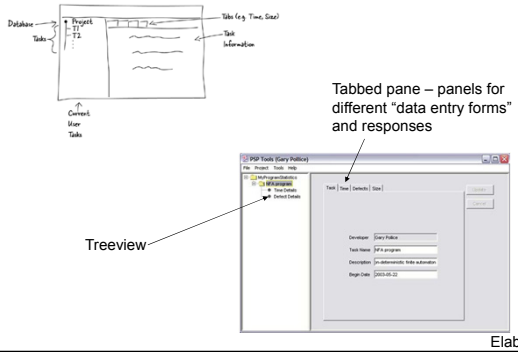
- Slightly odd here
- Under heading "architecture" they discuss
  - their first ideas as to user interface
  - The "packages" – GUI, managers (control), database, data-objects and "utilities"
- ?
- The GUI design didn't really seem to fit the architecture heading;
  RUP puts architecture responsibilities with "System Architect"
  RUP assigns prototype GUI design to another role – "User Interface Designer"

Use previous experience

- "A good analyst is a biased analyst"¶
- The GUI design proposed reflected lots of experience with Java swing framework.
  - Gary + Chris knew swing classes most often used to create GUIs, and common GUI configurations
    - Classes : JFrame, JPanel, JTree, JTabbedPane
    - Configuration:
      - Tabbed Pane interface with "tabs" for different input forms, tabular responses, and message;
      - Hierarchical tree-view for selecting subsets of data

¶If you know that "tree views" are easy to implement, this influences your analysis and you end up suggesting to the user that some data should be displayed in a tree view

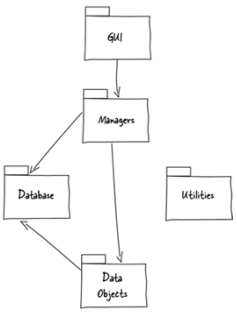
PSP-tools : GUI



Menus and dialogs

- Design model 1
  - Tabbed Pane has "tabs" for data input forms and for tabular presentations of selected data
- Design model 2 (the one used by PSP-tools)
  - Tabbed Pane only has "tabs" for displays of selected data
  - Data entry to be achieved by using menu to get dialog displayed, data entered in dialog – action button submits data and closes dialog
  - Data entry updates database and also tree view in left panel
  - Tree view is "active" – click on entry to place selected data in a structure that can then be viewed in a tabbed pane panel

PSP-tools : "package architecture"



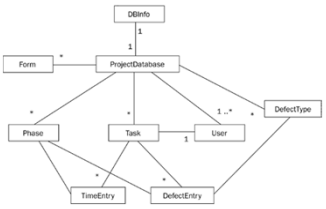
Architecture?

- Haven't really advanced much.
- The "architecture" derived so far:
  - Describes essentially every
    - Java-GUI/back-end data-store application via JDBC
    - C#/Visual-Basic "Winforms" using data access objects and a back-end data-store
- Developing this architecture was easy for PSP-tools guys
  - *We've done dozens of applications like this, roll out the standard elements*

Not always so easy in CSCI321

- May be required to construct an interface using a technology with which no group member is familiar
  - TCL/TK for GUI interfaces maybe?
    - Does it support "Tree Views", "Tabbed Panes"?
    - Are the GUI elements instances of classes or are they something else.
- Application may require an architecture more elaborate than those that you have met:
  - Single process
  - "Two-tier" (process that talks to database)
  - Simple WWW (browser-client, middleware, database)

Database design



Database design

- Tables ("entities")
  - The different types of data item you wish to store;
  - For PSP-tools
    - Projects
    - Users (many users participate in a project)
    - Phases (several phases in a project)
    - Tasks (many tasks in a project, each task the responsibility of one user)
    - Time entry (time entry associated with user, phase & hence project, and task)
    - DefectEntry (similar)

Database design

- As you refine your ideas, you get to
  - Table declarations
    - Primary keys (often auto-allocated, e.g. defect item's primary key to be allocated by database)
    - Foreign keys (defect introduced by user – so defect record will have a foreign key that is primary key in user table)
    - Other fields
      - User
        - » Name, Initials, encrypted password, …

Database

- Effectively defining "entity classes" at same time.
- You read a user-record from database
  Either
  you only want one field (select password from users where username=? and initials =?); if this is always the case, then can take data from result set,
  Or
  you want to use more of the data – in which case you probably want a "class User" that you can instantiate and fill in details

Database classes

- Consider defect:
  - In table: unique id, date, type, foreign-keys for phase, task
  - In memory
    - Unique id, date, type are ok
    - Do you want foreign keys as longs
      class Defect { private: long id, task, phase; date d; string type; … }
    - Or do you also load in phase and task data and build
      class Defect { private: long id; Task* tsk; Phase* phs; date d; string type; …
- You probably don't know which way to go at this stage; it is going to depend on the most typical patterns of processing;
  pick simpler style now, but remember you may have to change later.

Database – consistency, etc

- **Enumerated type** e.g. Defect type –
  - how about defining a constraint on values of field in table so database validates info on entry.
- **Foreign keys** –
  - you cannot expect PSP-tools user to know the unique id for project and for their identity when recording a defect or time entry;
  - implicit requirement that your processing code will have to get this information from data-tables once user has logged in and identified task for which they are entering data (in PSP-tools, they load the data into the tree view);
  - how are they going to identify task – your code will have to present a list showing possible tasks
  - ***Look how just thinking about data-table has given you a whole series of subtasks that you should record so that they can be scheduled and allocated***
- **Deletes** – "cascading the changes"
  - Remove a task?
    Not appropriate in this application, but similar applications will have deletes.
    All time logs, and all defects relating to that task should be deleted.

PSP-tools

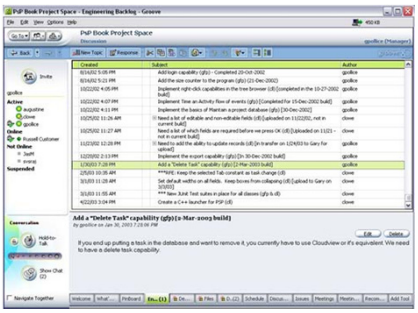
- Gary devised some SQL create table statements given their first database design diagram
- Tables created by running scripts – departure from earlier plan of having application create tables dynamically.
- Later they considerably simplify their data model and change all the tables.
  - Such a change allowed in "lightweight" approaches.
  - More formal (Waterfall) methods would have required them to do more careful analysis of data model at this stage and then have had the database design frozen.
  - That would have made the implementation problems they had much more serious.

Management activities

- "Engineering backlog"
  - Note examples
    - CRC role play, participants noted that their conceived user-interaction hadn't provided the data needed to create a tables
    - Decisions about data stored for a time record or defect record identified need to retrieve keys identifying user and a list of keys and descriptions for tasks
  - Add these to "project engineering backlog"

85
Elaboration

Engineering backlog



86
Elaboration

Engineering backlog

- This is an "agile" feature.
- Less agile approaches require that you have the foresight to identify all the tasks before you start the phase, so there won't be anything to add.
- Agile is more realistic

87
Elaboration

Management – feature cull and re-prioritize

- PSP-tools
  - Originally, had some idea of allowing users of PSP-tools to customize it so that they could select the data to be recorded etc;
  - Requires customization of data-entry forms, data-tables, entity classes etc
  - Ooops – too hard
  - Defer to "release 2"

88
Elaboration

Management

- Keep records of features
  - Their revised priorities
  - Iterations for implementation
  - "Release"

89
Elaboration

Incidentals

- Things you didn't ever think of that mess you up and divert you to work on support tasks
  - Cost time
  - Time that was never budgeted
  - Will make your project run late and cost too much
- PSP-tools
  - Installation
    - Technically less sophisticated group members (representing customers) couldn't install and run the prototypes
    - Have to divert effort into obtaining and learning how to use one of those systems that lets you create "self-installing" applications.
  - How to use the GUI?
    - Found it necessary to get a Windows screen recording thingy that let them record "movies" of sessions so that potential users could in effect watch demonstrations.

90
Elaboration

Elaboration – steps toward the first executable

- PSP-tools Starting
  - Install Java
  - Install a database product (Cloudscape)
    - Often people choose things like MySQL or Apache Derby for the database used in development phase
    - The "Developer" (or "Express") editions of DB2, SQLServer, Oracle etc might be more appropriate
  - Picked a code versioning system
- Defined their project in terms of layers – user-interface, control and business logic, persistent data;
  used this to define an initial package structure

Build a GUI

- No functionality behind it
- Just a GUI
- See book for their arguments as to why they made this their starting point
  - More the "agile" style
  - They have a tame user, they want to keep Russell constantly involved, showing him bits and asking "is this right?"
  - They argue that changes to GUI have major ripple effects causing changes to all other parts, hence they argue that do as much as possible to get the GUI right early on

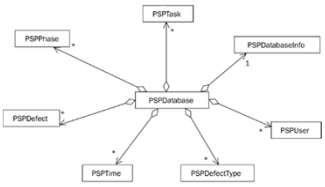
PSP-tools GUI

Faked! Data shown in tree defined by constants in early code.
Data "added" via form-panels isn't added to anything.

Early GUI

- Important to explain to quality testers and customer representatives that functionality is limited or non-existent.
- Otherwise
  - "Defect report log" gets filled with bug reports about things not working (when engineers knew they weren't implemented yet)

They simplified their database model

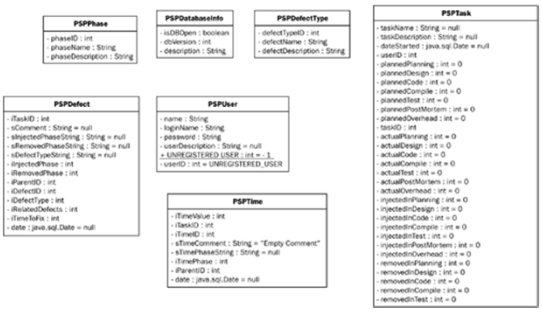


Main thing is that now the database represents a single project where participants are using PSP; so don't have to keep track of which project users and tasks belong to etc; a lot of earlier relations between entities have gone so removing need for foreign keys etc.

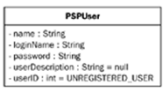
Simple entity classes

- They defined a number of simple entity classes – each corresponding to a table.
- Entity objects represent rows.
- "Documented" them with UML class diagrams – but these are simply lists of the fields (table-columns)
  - No real functionality defined for these classes
  - Presumably (though not shown) there are getX (accessor) and setX (mutator) functions for each field
- UML class diagram maps easily to SQL createtable

PSP-tools Entity classes



PSP-tools Entity classes




```
– Users table. These are the users of this project database
CREATE TABLE Users (
        userName        VARCHAR(50)         NOT NULL,
        loginName       VARCHAR(20)         UNIQUE NOT NULL,
        password        VARCHAR(20),
        userDescription VARCHAR(250),
        userID          INTEGER DEFAULT AUTOINCREMENT INITIAL 1000 INCREMENT 1
);
INSERT INTO Users VALUES
        ('Gary Pollice', 'gpollice', NULL, 'Chief curmudgeon'),
        ('Chris Lowe', 'clowe', NULL, 'The smart one'),
        ('Liz Augustine', 'laugustine', NULL, 'The efficient one'),
        ('Jas Madjur', 'jmadhur', NULL, 'The cerebral one');
```


Entity classes

- They really were thinking in terms of the database tables.
- They could simply have used SQL create-table statements and created the tables
- Many systems can import meta-data from database and use this to create entity-classes or entity class diagrams.

I have feeling that they were hacking a bit at this stage; a little too much agility, not enough thinking things out.

Classes other than entity classes

- They've built their GUI
  - Used a GUI-builder to assist the process
    - See their comments about problems that this can cause if you switch development environments
- They've got their entity classes
- Now they need the bits with the control logic and the database managers – the bits that really provide functionality

PSP-tools group adopts "Test Driven Development" strategy

- Really an XP practice.
- You are planning to implement class X which (in this phase) has functionality f1,f2, and f3
- You start by thinking up tests
  - Create an X object
  - Initialize it
  - Ask for property value
  - Invoke f1 operation that should change that property's value
  - Ask for (changed) property value
  - Assert difference between original and new values is the difference that was expected

JUnit testing

- They created their unit test cases
- For example
  - They plan a class PSPUser
    - It has name, and loginname fields, these to be set by the constructor
    - It will have an "equals" operation
    - Two PSPUsers are "equal" if they contain same strings in their name fields, and in their loginname fields
  - Hence test

```
public void testEquals() {
    PSPUser u = new PSPUser("Gary Pollice", "gpollice");
    Assert.assertTrue(u.equals(new PSPUser("Gary Pollice",
                "gpollice");
}
```

Uhm – I'm still not convince of the value of such micro tests.

End of elaboration phase

- Build an executable architecture?
  - Er no
    - But they have a GUI toy that does nothing, but does that in an attractive way that pleases the customer
    - They have some faked data inserted into tables so they will soon be able to start testing data management code and entity classes
    - They have their unit test framework set up; the tests are scripted; they will be able to automatically retest after every change from now on.

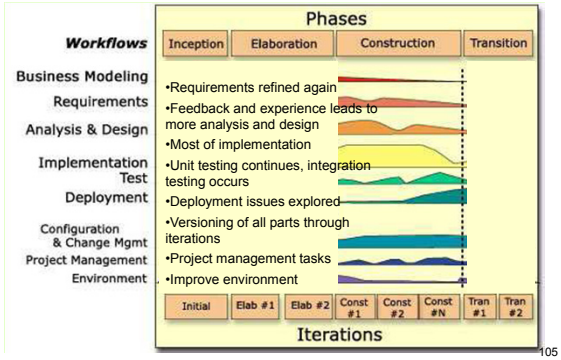
103
Elaboration

PSP-tools view of state at end of elaboration

- We have an executable architecture!
- We have refined the scope
  - Taken out some things that would have been too hard
- We have a sketch of an iteration plan for construction phase (they don't appear to include it in the text)

104
Elaboration

Construction



105
construction

A bit light on the planning!

- The account in the book suggests that there was significantly less work done on planning (managerial, analysis+design, …) then might be expected for a professional RUP project
- Examples
  - Iterations
    - Not shown the iteration plan devised for construction phase
    - 12 iterations reported
      - Twelve iterations in 8 weeks?
      - Really these are "after the fact iterations", they've added something and decided that the extension justifies a new "internal release" to the customer representative and the quality assurance part of team
  - Classes
    - Evidence suggests no UML class designs

106
construction

No UML class design?

- Supposedly we
  - Create designs
  - Diagram them in UML
  - Review
  - Work out interaction patterns, document these
  - Then code
- They dived into coding after only limited design
- Appear to have "Reverse engineered" the code to get UML diagrams
  - Reverse engineering is a feature available in "professional" versions of things like RationalRose
  - Use is meant to be:
    - Create UML design overviews for imported or legacy code
    - Reduce work involved in updating UML diagrams to maintain consistency with code after "refactoring"

They don't put reverse engineering into the versions of RationalRose used when learning UML – because they don't want to encourage such hacking!

107
construction

"Agile"

- As noted in text, this wasn't a pure RUP model.
- They also wanted to get some experience with agile methods; XP (eXtreme Programming gets a couple of mentions), unit test strategies were XP motivated.
- Agile methods do tend to go more lightly on the design documentation – so no UML class diagrams, code first, reverse engineer afterwards to get documentation

Waterfall---RUP---Agile---Hacker---Student

You can position your work style anywhere along this axis. Whatever you choose, you will encounter problems – the particular difficulties you meet vary with your chosen style.

108
construction

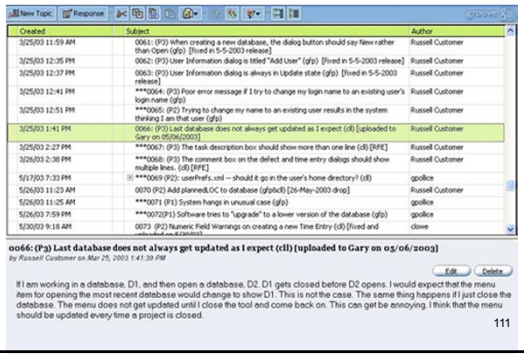
Environment

- RUP diagram shows “hump” for environment discipline at the start of each phase
  - You extend your environment
  - Different support tools required in the different phases, you must install, make available to team members, and possibly train some team members in their use
- PSP-tools
  - A few extra problems
  - They switched existing tools as well (Sun Forte Java IDE replaced by Eclipse etc)
  - *When doing your CSCI321 project, try to avoid following the example of this team in regard to environment changes!*
- Typically, at start of construction phase
  - Start using code versioning system seriously
  - Additional testing tools (code coverage, scripted GUI-testers, batch scripts for regression testing) come into use
  - Code generators may be used (possibly necessitating training in their use)

Management

- Put in place system for “defect tracking”
  - PSP-tools illustration shows most of “defect notices” being raised by Russell (Customer) who continues to be supplied with “executable releases”
  - More typically defects logged by
    - Your group’s “Quality assurance” person performing “black box” testing of the system
    - Developer’s flagging problems with code that has been done in an earlier iteration

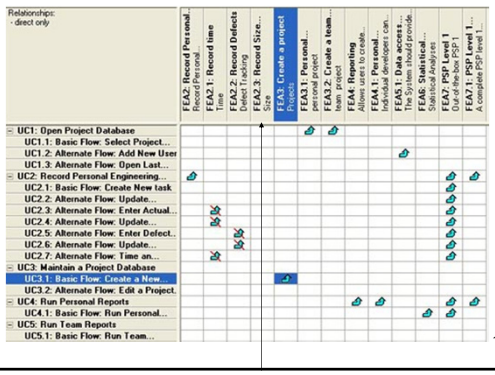
Defect logging



Management

- Requirements traceability
  - Get list of agreed requirements (from updated Vision produced at end of elaboration phase)
  - Check that each requirement has a use case associated with it
  - Check that each use case is being developed into code

Traceability matrix



Construction

- Executable architecture from “elaboration”
  - Nothing relating to
    - Generating reports
    - Viewing data from time records or defect records
    - Recording size data for tasks
    - …
  - Faked GUI support (but no underlying processing of data or persistence) for:
    - Recording time on tasks
    - Recording defects
    - …

Construction - iterations

- First couple of iterations look as if there may have been some planning prior to commencement of construction; subsequent iterations are "serendipitous"
  - "*Chris and Gary worked at a steady pace to implement the needed functionality. Every few weeks, they produced a build that was ready for testing. When the build was released, anyone could, and everyone was expected to test it, and report any defects found. Russell or Gary often added requests for enhancements; after considering priority and effort, the became new requirements. Then Russell and Gary would reallocate work based on the priorities. We did this all without a formal planning cycle.*"

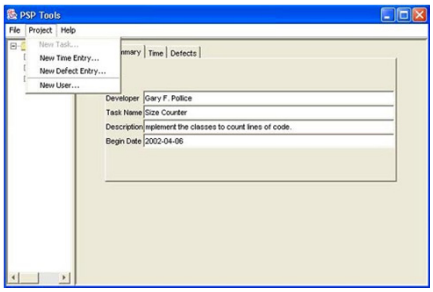
115
construction

Construction – a first iteration

- An executable architecture that actually does something
  - Details are not clear from the account but it appears that a user of the PSPtool program construction-version-1 could
    - Open an existing database
    - Add time records and defect records
      - still only stored in memory (no persistence to database!)
      - Appear in tree view, can be processed when generating summary reports
    - View totals (simple summary reports) in tabbed pane panels

116
construction

Construction – a first iteration



117
construction

Construction – a first iteration

- Time for first iteration not specified.
- Book discusses "delighters" –
  - *Something that costs nothing to implement and will delight users*
  - In this case, it lead to a User Preference's file
    - Why a file? Why not a table in database?
      If you make it a file, it isn't the same when user working own main desktop machine or on own laptop.
      If you make it a file, there will be issues with multiple users needing preferences files – do you understand enough about home directories on different operating systems so that you can put the file there?
  - At this stage, preferences file records the "database" last used, the "delighter" feature makes it a one-click step to open this database (otherwise dialog needed, name must be entered etc)

Cloudscape database = a file; you can have many different "databases"; for things like DB2 or Oracle you would have to be defining "schema" to achieve same thing

118
construction

Delighters?

- More hacking?
- A more thoroughly planned RUP project would have identified the need for "user preferences" and factored these in to the construction iterations (and also resolved issues of where to store preferences which apparently became a problem in a later iteration)
  - Basic preferences support would have been scheduled for a construction iteration (probably the iteration after that where data persistence achieved)
  - Different preference features would have been scheduled into later iterations

119
construction

Construction – Iteration 2

- Features
  - Help menu (actually no help at this stage, simply the "About PSP-Tools" dialog!)
  - Improved Project-menu/New User dialog
  - "Prettier" dialogs
    - (Greater experience with javax.swing classes)
  - FINALLY – the data go into the database tables
- Well, OK, now do have a kind of executable architecture.
- Two weeks (*remember they are working part time, so that probably means about 16 hours work*)

120
construction

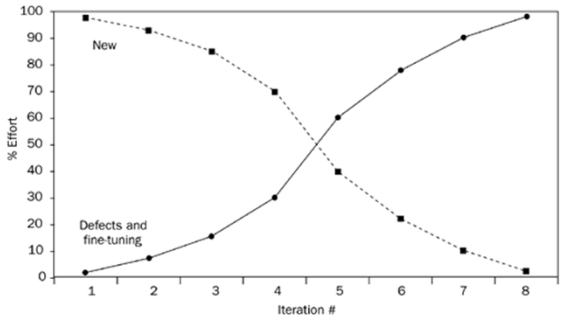
Construction – Iteration 3

- Features
  - Create a database from within PSP-tools
    - As noted earlier, this is only meaningful if using small "file-based" databases like Cloudscape where each database, with all its tables, is created in a separate file.
    - Creating a new schema in a real database would have been more involved (and usually wouldn't be possible as only the database administrator can create schema)
  - Right-click pop-up "context sensitive" menus associated with tree view

Construction – Iterations 4-12

Iteration[7]	Functionality Added
C4	Incorporated activity time and defect entries. Also implemented activity timer that updates the database directly. Implemented ability to update task summary information directly from the task summary panel.
C5	Added line counter tool to the program. Improved login dialog. Removed need to run with a Cloudscape database server.
C6	Installed database schema changes and automatic database upgrade mechanism. Added the Database Properties dialog box.
C7	Added basic export function. Made user information editable.
C8	Added program size and estimation tab.
C9	Added ability to delete a task.
C10	Delivered a self-extracting archive that unpacked the required files into a target directory and removed dependencies on the user's environment.
C11	Fixed defects. Released an initial User's Guide.
C12	Added an executable program for Windows that launched PSP Tools. Users no longer needed to run a batch file, and extra windows no longer appeared on the user's task bar.

Balance of effort : new & "improve"



Database "versioning"

- They were still hacking re-factoring their entity classes and the way their program interacted with the database.
- Tables and entity classes being changed
- Consequently "databases" created by QA and customer are incompatible with later releases of the software.
- Their approach meant that they had to devote effort to mechanisms for identifying incompatibilities and for migrating existing data.

- You will probably create similar problems for yourselves in CSCI321.

"Single quotes in fields"

- Curious
  - Highlighted as a problem
  - Everyone knows that issue easily resolved if use PreparedStatement rather than Statement
    - Varchar field for comment
    - SQL of form
      - "update datatable1 set description='" . Info . "'"
    - Get's unhappy when you put in Gary's Info
- Well they did note that they didn't have a database guru on their team.

Testing

- JUnit style tests continued.
- Also made use of a Java code-coverage tool
- But how do you test a GUI?

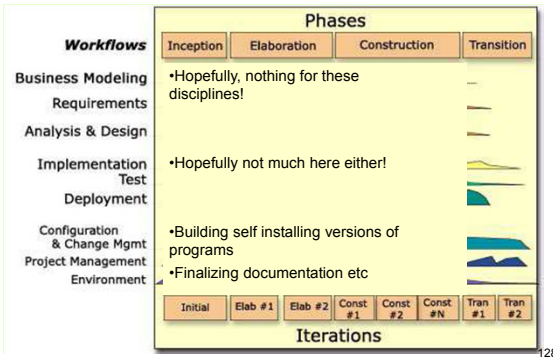
- They made use of a specialist tool
  - Records actions
  - Generates a script (their tool generated Java code)
  - Can then rerun this script anytime and tool with drive operations on the program under-test.

Testing script code - fragment

```
// Frame: PSP Tools
PSPMainFrame().click();
PSPMainFrame().click();
menubar().click(atPath("File"));
menubar().click(atPath("File->Open..."));
Filename().click(); JDialog().inputChars("d:\\pspproject\\project1");
Open().click();
PSPLoginDialog().click();
passwordtext().click();
PSPLoginDialog().inputChars("psp");
Login().click();
tree().click(atPath("project1"));
menubar().click(atPath("Project"));
menubar().click(atPath("Project->New Task..."));
…
```

127
construction

Transition



128
transition

Transition

- RUP suggestions
  - Users validate product – does it really do what we specified
    - With more agile styles, this requirement should have already been satisfied by having the users involved in testing all the incremental releases
  - User training
  - Product packaging
    - Installable code
    - Documentation
    - Help files (multi-lingual?)
  - Support engineering

129
transition

Transition

- PSP-tools
  - 3 weeks
  - Report – well doesn't clarify much; it mentions:
    - Fix a bug
    - Finalize user manual (on-line help had disappeared somewhere on route)
    - Packaging for distribution (they had really resolved this earlier because of problems they had had making thing work for their testers)
    - Training of users

130
transition

PostMortem

131

PostMortem phase of a project

- Group meets to discuss project given benefit of hindsight
- Task
  - Identify where problems occurred and why
  - Identify what worked well
- Aim
  - Consolidate your experience
    - Build on what you did well
    - Take avoiding action early if you see similar problems arising in another project!
    - Remember the final versions of the classes that you developed and the patterns of interactions among instances of these classes; next time, start with these as your pre-conceived categories for analyzing use cases (instead of the rubbish categories that you used when you started this project).

132

PSP-tools PostMortem

- Some of the problems
  - No database guru (5th member John dropped the project early on, had been designated as database guru)
  - Should have spent more time team building early on
  - Should have been more serious about testing
  - Should have used a better defect tracking system
  - …

133

PSP-tools

assessment

134

What mark did Pollice's team get for their CSCI321 project "PSP-tools"?

- Designers of CSCI222 2005/2006
  - 95 HD, great, perfect model for CSCI222, base all assignments on this work
  - 88 HD, yes
  - 90 HD, concur (*well, to be honest, I didn't actually read it*)
- NABG
  - 70 Cr, they hacked

135

Yes – it is more like a CSCI321 project than was intended

- Typical CSCI321 problems
  - Group member drops out
  - Team members inexperienced in some aspects of technology
  - Most of the tools used were new to team members
  - Most of coding work done by a subset of team members, contribution of other members is limited
  - Lack of planning, resort to hacker style development
  - Inappropriate assumptions about technology

136

It had an advantage not shared by CSCI321 projects

- An involved customer – Russell
- His presence made a continuous use testing approach feasible. "Executable architecture" did develop fairly early and subsequent iterations did build on this.

137

An example of RUP?

- Phases and iterations were followed
- Importance of "executable architecture" was recognized
- Attention was paid to different RUP disciplines – they did explicitly consider some of management and other tasks associated with each phase
- RUP milestones and their deliverables did help guide what the team did in each phase.

138

Weaknesses

- From a RUP perspective – too little effort in analysis, design, and planning.
- It has too much the flavor of an XP or hacker style project.

139