An introduction to methodologies for the Software Hut

Professor Mike Holcombe

More details in the Book of Genesys Solutions

Department of Computer Science, University of Sheffield

The needs of the Software Hut

- We need to deliver high quality in a short time – the deadline is fixed
- We need to meet the clients' requirements as far as we can
- Requirements capture is vital – if we get this wrong then everything else is wrong!
- Quality is vital – people are going to be using your software – it may be business critical!

The consequences

- We have to use an approach that is appropriate for this situation
- Our method has been developed and proved over a number of years
- It is able to deliver high quality systems quickly
- It has been shown to be more enjoyable for the developers than traditional methods

Agile software engineering

- Many software projects – large and small FAIL
- Often nothing is delivered
- What is delivered is often wrong
- It is often buggy and unreliable
- It often does not deliver the functionality needed by the business – which may have changed since the start of the project
- Established design approaches are often part of the problem

Plan

- **Why are lightweight methodologies emerging?**
- **Problems with design-based approaches.**
- **Removing unnecessary bureaucracy.**
- **The promise of eXtreme Programming, (XP).**

Lightweight methodologies are approaches which reduce the impact of expensive parts of the software engineering life cycle.

Typically this means reducing the complexity of the process and making it less bureaucratic.

Also, trying to make it more human friendly.

Lightweight methodologies focus on building quality software -

rather than on creating vast amounts of documentation that is often:

incomplete,

inconsistent

and unintelligible!

XP tries to address issues where current software engineering fails i.e.:

the dynamic nature of modern business

by the time the design is done it is out of date;

as business needs change the software has to evolve – this is hard to do.

The four basic principles of XP

Communication

Feedback

Simplicity

Courage

The twelve *sacred* tenets of XP

1. **Test first programming.**

Before writing any code programmers build a set of tests.

These tests are run – of course they will fail as no code has been written!

Why would one do this?

To get used to testing *continuously* –

At the end of a session, at the end of the day, whenever a small piece of code has been built -

ALL the test sets are run, this means -

all the relevant unit tests,

all the functional tests.

The test sets are the most important resource and are continually enhanced.

The customer helps to supply tests.

Functional tests are derived from the planning game (see below).

The test sets replace the specification.

If any tests fail the code must be fixed.

2. The planning game

The customer provides business stories and estimates are made about the time to build software to implement the stories.

Programmers use story cards to define the stories

The customer decides which stories provide the most business value.

The programmers then implement them.

3. Small, frequent releases

Release early and release often.

This is like incremental delivery or RAD.

4. Always use the SIMPLEST design that adds business value.

Over-complicated systems are usually bad systems which are difficult to test and maintain.

5. System metaphor.

Programmers define a handful of classes and patterns that shape the core business problem and solution.

A kind of top level architecture.

This might be written on a whiteboard. Some aspects may change as we learn more about what we are building.

6. On-site customer.

Encourages intense face-to-face dialogue.

Not always practical.

We must have very good communication with ALL the key players in our client's business.

7. Refactoring.

Restructuring code without changing its functionality.

Used mainly to SIMPLIFY code – make it more understandable, more maintainable.

Avoid distributed methods, large complex classes etc.

8. Pair programming.

Two people - One machine.

All code must be written in this way.

This is *continuous* review and gives a much greater understanding of what is being done.

Pairs swap around frequently.

Different pairs form up regularly.

9. Collective code ownership.

ALL the code belongs to ALL the programmers.

Anyone can change anything.

There are house rules for writing and documenting code and for communicating between teams.

10. Coding standards.

Defines rules for shared code ownership and for communication between different team's code.

Consistent class and method naming.

Everyone should use the same coding styles.

11. Continuous integration.

Code is integrated into the system at least a few times every day.

All unit tests must pass prior to integration.

All relevant functional tests must pass afterwards.

12. Forty hour week.

Tired programmers write poor code and make more mistakes.

Programmers don't have to be heroes!

Planning our time is vital to save pain later.

It is quite hard to stick with ALL these rules - XP requires *discipline*.

Some teams need a "coach" to ensure that they do stick to XP!

There have been successes as well as failures with XP –

More research is needed.

It is demanding

- For the programmers – they need to develop all round skills
- For the clients – they need to give up more time
- For managers – they need to trust their teams more
- But it raises the profile of testing!

Does it work?

- **Some comparative empirical evidence that it does.**
- **Different teams using XP and traditional methods building the same systems for the same client.**
- **XP delivered better quality, quicker with less stress.**

What are the problems with XP?

The biggest problem is with the functional test sets.

Study the approach in the *Genesys Solutions* book .

The system metaphor development also needs care.

And can it work for big projects?

Proper support is needed

- Support for managing XP projects
- Support for XP testing
- Support for training and reinforcing the XP principles
- Support for code conventions

XP in Genesys Solutions

- **Introduced XP in 2000.**
- **Both new projects and maintenance projects**
- **Popular with most programmers but not all.**
- **It's easy to degenerate into bad habits.**
- **Regular reinforcement of the philosophy is needed.**

Non-functional requirements

- **Increasing these need testing in a sophisticated way.**
- **Specialist testers will have to oversee this.**
- **Usability, reliability, accessibility, etc.**
- **All these issues require specialists.**

Summary.

The whole point about XP is that it is an *agile* process.

It will develop and adapt to changing needs.

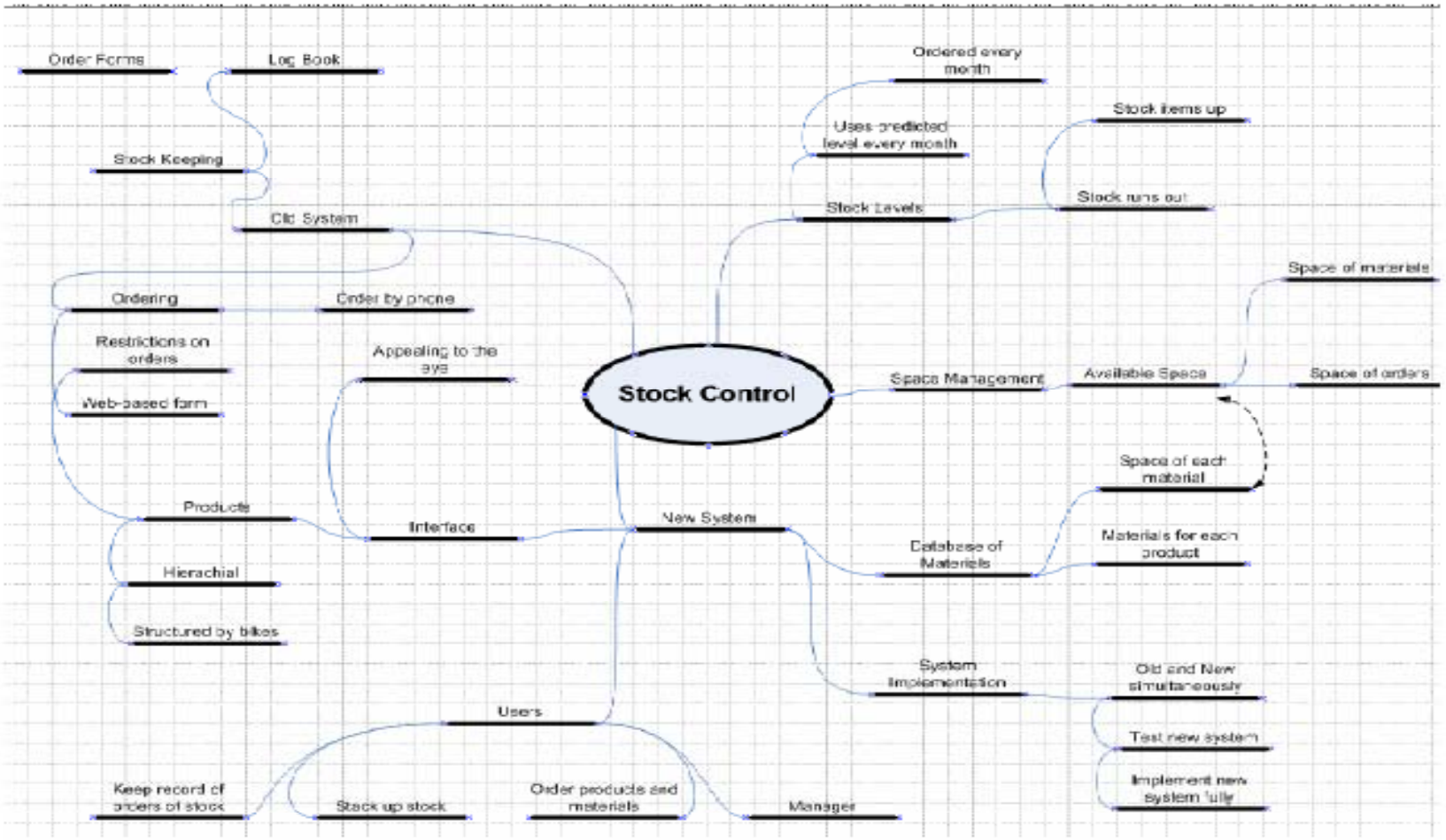
The Software Hut approach

- It is based on story cards and rapid incremental builds
- We use Extreme X-machines (XXM) as the system metaphor
- Unit tests are done with JUnit – or equivalent
- Systems tests are based on XXMs
- Coding standards are strictly enforced

Basic iterative cycle

1. Carry out a brainstorm and use Mind Maps to identify the key aspects of the requirements
2. Define some key stories, estimate their cost and prioritise them with the client
3. Write the unit tests for the stories first and start to build the story code
4. Test the stories
5. Integrate the stories into a single system
6. Test the system
7. Go to 2 and repeat until finished

Mind Maps


Stock Control
Order Forms
Log Book
Stock Keeping
Old System
Ordering
Order by phone
Restrictions on orders
Web-based form
Appealing to the eye
Products
Interface
Hierachial
Structured by bikes
Ordered every month
Uses predicted level every month
Stock items up
Stock runs out
Stock Levels
Space Management
Available Space
Space of materials
Space of orders
Space of each material
New System
Database of Materials
Materials for each product
System Implementation
Old and New simultaneously
Test new system
Implement new system fully
Users
Keep record of orders of stock
Stock up stock
Order products and materials
Manager

Story cards

Customer story card | **Project title** Quizmaster

Date March 15th 2001 **Project phase** Initial

Requirements number 3 **Story name** Lecturers can edit details of a topic.

Task description (English)

Lecturers may inspect the list of topics and papers and change the detail of the topic for a paper

Initiating event Lecturer requests edit option

Memory context Current list of papers and topics is updated to new list

Observable result Updated memory changes confirmed

Risk factor 1 **Change factor** 1

Related stories Lecturers can add topics.

Notes

Story name _____ **customer approval** ..... **date**.....

Resource estimates actual

input	☐	**simple**	■
output	☐	**average**	☐
enquiry	☐	**complex**	☐
reference file	☐		
database	■		

Function/object point total 1 **Man-hours total** 12

Functional tests

1. Carry out an update on an existing record
2. Carry out an update for a non-existant record [error]
3. Define a n illegal topic type [error]
4. Do nothing and exit the function [cancel]

Quality attributes

The process of updating the topic and receiving confirmation of correct operation should be instantaneous

The operation process should be clear from the interface design

Trials of this function amomgst representative users should be 99% successful

Tabular approach

story	function	input	current memory	output	updated memory	change risk
1	click(customer)	customer button click	-	new customer screen	-	low
1	enter(customer)	customer details entered	current customer database	confirmation details screen	-	medium (nature of details liable to change)
1	confirm(customer)	customer confirm button clicked	(current customer database)	OK message and start screen button	updated customer database	low
3	click(order)	orders button clicked	-	new orders screen	-	low
3	enter(order)	new order details entered	current orders database	confirmation orders screen	-	high (nature of details of orders liable to change)
3	confirm(order)	orders confirm button clicked	(current orders database)	Ok message and start screen button	updated orders database	low

Software Hut Management Tool

Now viewing team: 0, switch to: (1-14) Switch

Main Menu

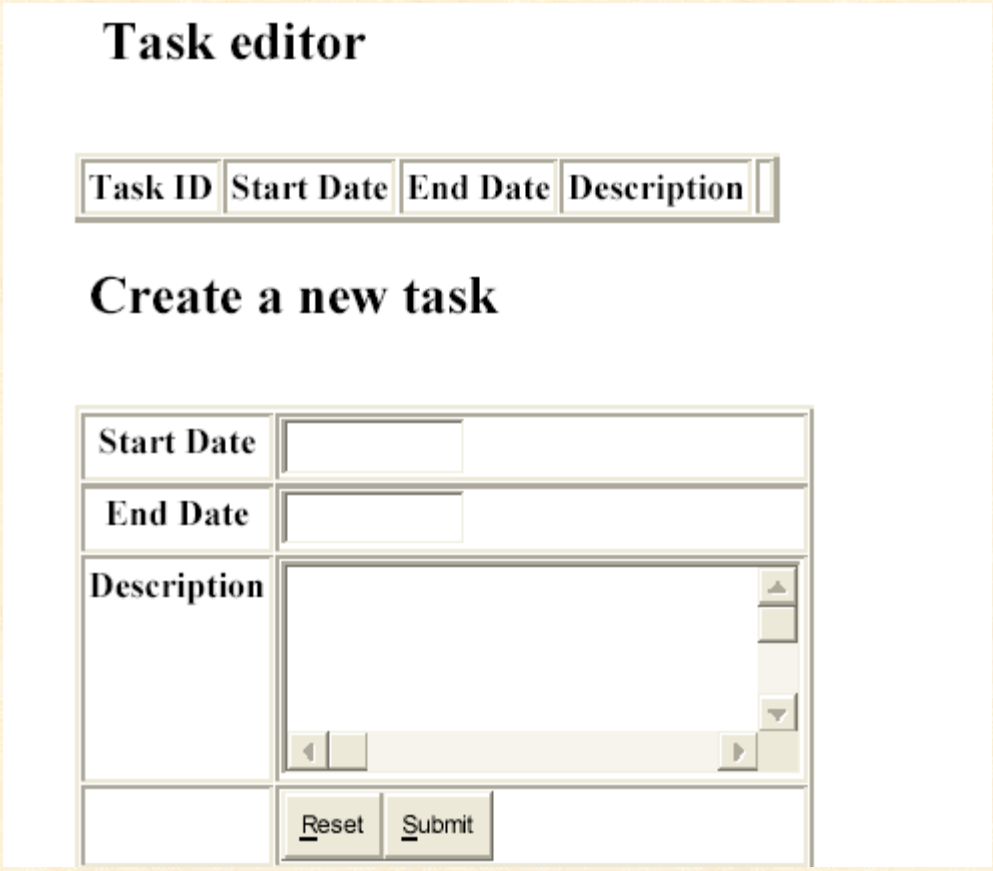
Document	Create	Edit	Reports
Meetings		Edit	
Requirements		Edit	Final Report
Story Cards		Edit | Print	Final Report
Time Sheet		Edit	
Files	Ready next week!	Edit	This week
Actions	Search for actions	My Actions	My Accomplishments
Tasks		Edit	Project plan

Meetings Menu

Date/Time	Location	Reason		Agenda Items	Minutes	Present

Create a new meeting

Date	
Time	
Location	
Comments	
	Reset Create New Meeting


Task editor
Task ID
Start Date
End Date
Description
Create a new task
Start Date
End Date
Description
Reset
Submit

Action viewer (view restricted on: minute(), task(), and status(in progress))

Minute	Action Id	Task	User	Due Date	Time Estimate	Actual	State	Task	

Add new action

Minute ID	
Task ID	
User ID	
Due Date	
Time Estimate (hours)	
Actual Time (hours)	
State	
Description	
	Reset Create

Timesheet for the week beginning: Mon 06 Feb

	Mon 06 Feb	Tue 07 Feb	Wed 08 Feb	Thu 09 Feb	Fri 10 Feb	Sat 11 Feb	Sun 12 Feb
Lectures							
Team Meetings							
Preparing documents							
Drawing UML diagrams							
Desiging Screens							
Writing sepcification							
Desiging							
Writing Code							
Testing							
Preparing Presentation							
XXM							
Story Cards							
Unit Testing							
Meeting the Client							

Update

Previous Week | Next Week | Print | Print (timesheet for whole team)

Back to: menu

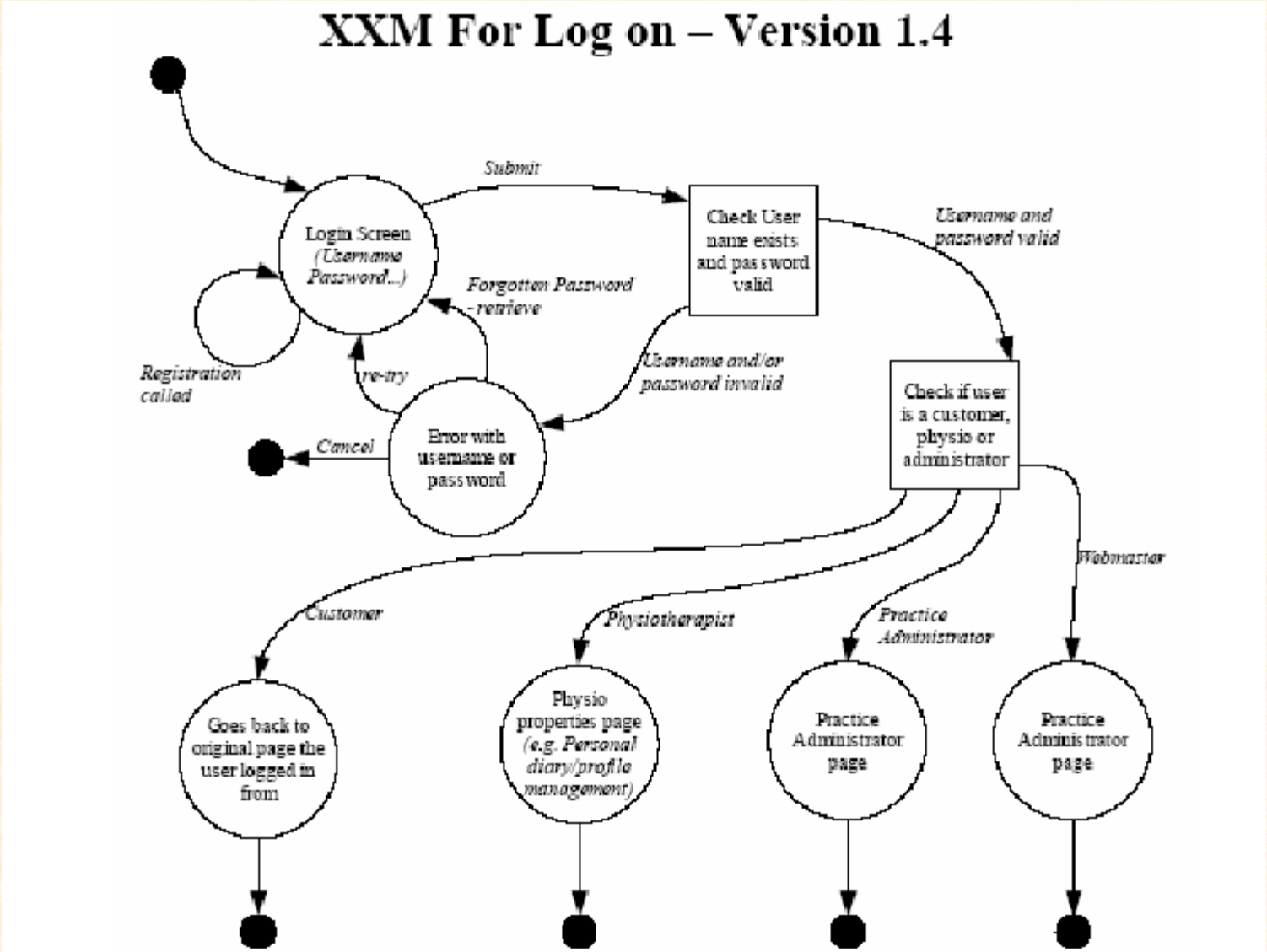
Story Card Editor

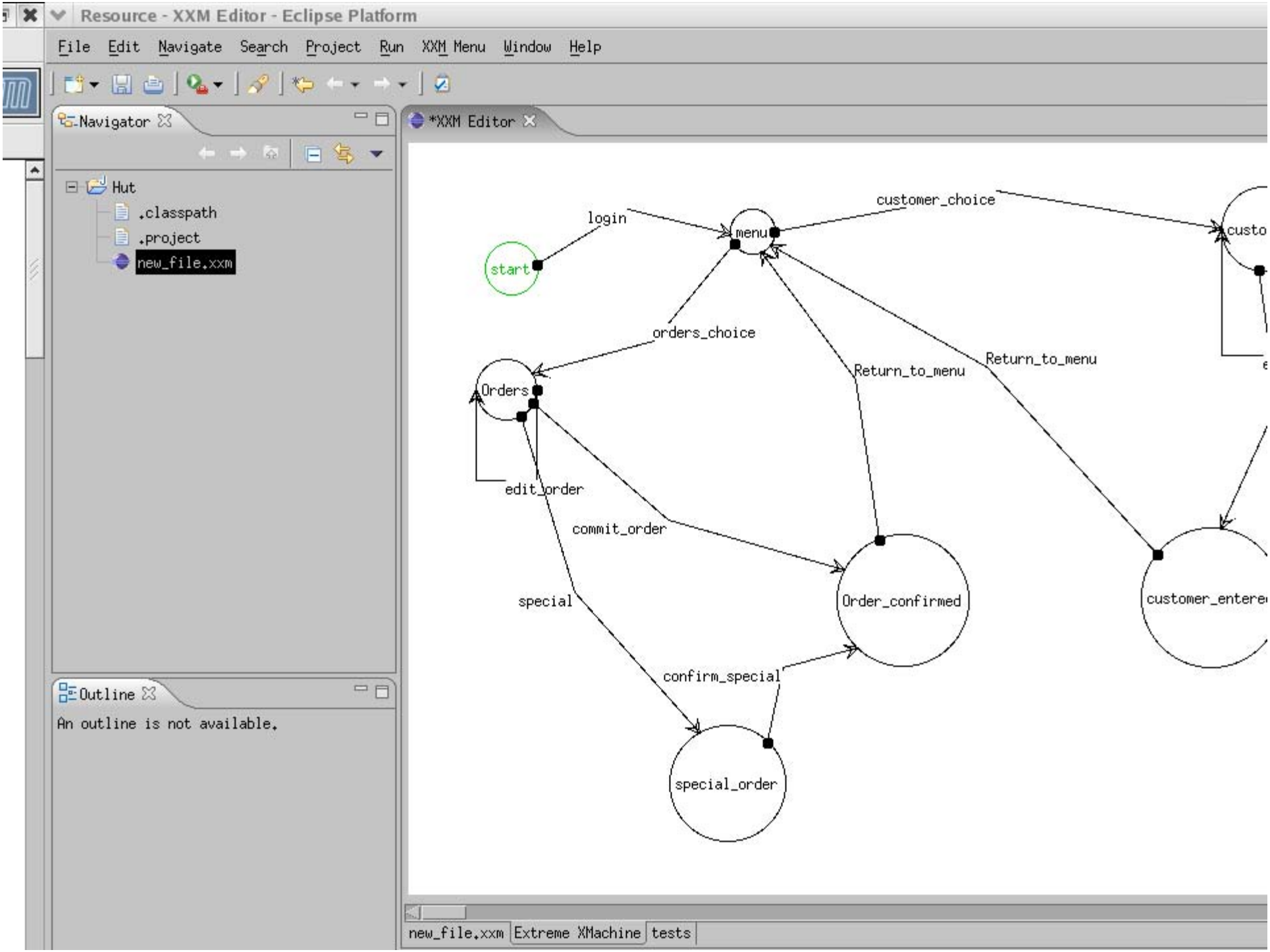
Story ID	Story Name	Description	Related stories	Man Hours	Functional Tests	Quality a
264	v2 Client/consultant Profile	The user edits their...	...	588	(Client Page) > clic...	data valid
270	v2 Client Logout	Performs all necessa...	...	577	(Client Page) > clic...	...
354	v2 Admin client Search	The administrator in...	...	0	- 'No query results'...	...
377	v2 Admin consultant Search	The administrator in...	Admin View consultan...	0	- 'No query results'...	...
605	v2 Validate Validation Queue	The administrator ch...	v2 Valid/Invalid Use...	123	...	...
718	v2 Client page	The client page acts...	Consultants search ...	599	...	- Correct
723	v2 Admin CP Log-in	Upon logging in with...	...	599	- Page displays corr...	- Compre
727	v2 Consultant Registration	The first part of th...	...	0	(Consultant Account ...	...
729	v2 Login page	When the user inputs...	Client page Admin C...	112	...	Good sec
732	v2 Register as a client	The user clicks on t...	...	112	...	...
736	v2 Register as a consultant	The user clicks on t...	...	112	...	...
740	v2 Admin Modify Client	The administrator di...	...	112	...	...
752	v2 Consultant Page	The consultant Page ...	...	632	...	Data disp
754	v2 Guest page	Initial page where a...	...	599	...	Interface
755	v2 Client CV	The user edits their...	Client Page...	588	...	Data vali

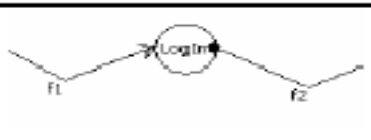
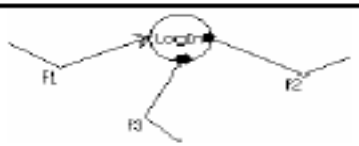
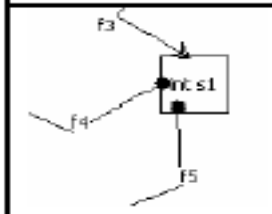
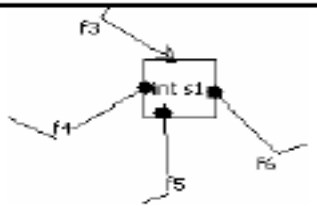
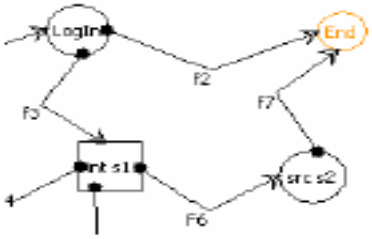
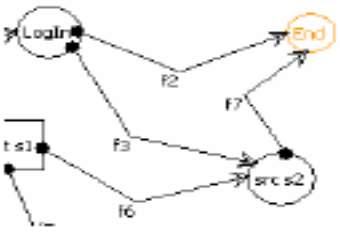
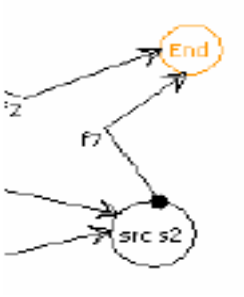
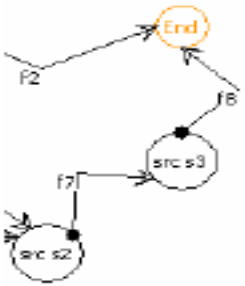
New Storycard

Story Name	v2 Admin consultant Search	
Description	The administrator inputs any number of criteria to query the list of registered consultants, and performs a search. If no matching records are found, the administrator is returned to the main admin page. If one or more records are found, a page is displayed showing those records in their brief form. Each is clickable, taking him to a full view. An option is available to return to the main admin page, with no further action taken.	

Extreme X-machines (XXM)




Resource - XXM Editor - Eclipse Platform
File Edit Navigate Search Project Run XXM Menu Window Help
Navigator
Hut
.classpath
.project
new_file.xxm
Outline
An outline is not available.
*XXM Editor
start
login
menu
customer_choice
orders_choice
Orders
Return_to_menu
Return_to_menu
edit_order
commit_order
special
Order_confirmed
confirm_special
special_order
customer_entere
new_file.xxm
Extreme XMachine
tests

Before	After	Comment
		Name changes in states and functions tend to denote one of 2 things: a typo, or less commonly a change of purpose.
		An additional exit function from a screen/state denotes an additional user option.
		An additional exit function from an internal state, suggests extended functionality.
		Functions can be redirected; this can lead to a significant change in functionality. Although the details of the implementation may experience only small change.
		The addition of an extra state into a path denotes a significant change in functionality. The more connected the new state is, the greater the impact of that change.

Evaluation

- We have been using the ADEPT framework for 18 months in 15 commercial projects with 10 teams
- Indications of its value are very promising
- Teams need some training - we provide about 5 hours
- Then we run the Speed Code Challenge!

Genesys Speed Code Challenge

1. Work in your usual teams (in this case SysAdmin work with Marketing);
2. Use you team space – in the lab and the Intranet;
3. *Languages*: Java, MySQL;
4. *Tools*: Eclipse, svn, ADEPT, Junit;
5. *Deliverables*: Story cards, Unit tests, incremental builds, X-machines, System tests, final build.

Target system: *GenPlay* - A web-based management tool for organising playlists of music tracks.

Tracks (songs) will have a *title, singer, genre* and *length* (time).

Functions:

1. add a new song;
2. delete a song;
3. query for songs on artist and genre;

Advanced feature

4. generate a random sequence of songs to last for a given (user specified) length of time

All this to be done between 11.10 and 5.00 on Wednesday October 12th. 2005 (6 hours)

Conclusions about the process

- Helped knowledge transfer between teams
- Easy to maintain model
- Easy to understand model
- Significant changes are documented, whilst trivial changes do not create more work.
- We are continuing to use and develop our methodology

Comments – Genesys Team 10, 2005

- We found the extreme machines very useful as they helped us to gain a fairly simple view of the whole system at a relatively early stage of the project
- When a mistake/better design is found, we simply update the corresponding XXM
- This process was quite efficient since the changes required to the XXM were usually minimal, and as long as the XXMs were kept up to date, anyone in any doubt had a quick reference to what the system was supposed to be

Comments – Genesys Team 6, 2005

- Extreme X-Machines were central to the development process, allowing fast turnaround of design documents in parallel with the equally rapid eXtreme implementation
- The simple, structured nature of the X-Machines ensured our time was focused on producing relevant, useful plans in lieu of more complex design approaches

Postscript: so what about X-machines?

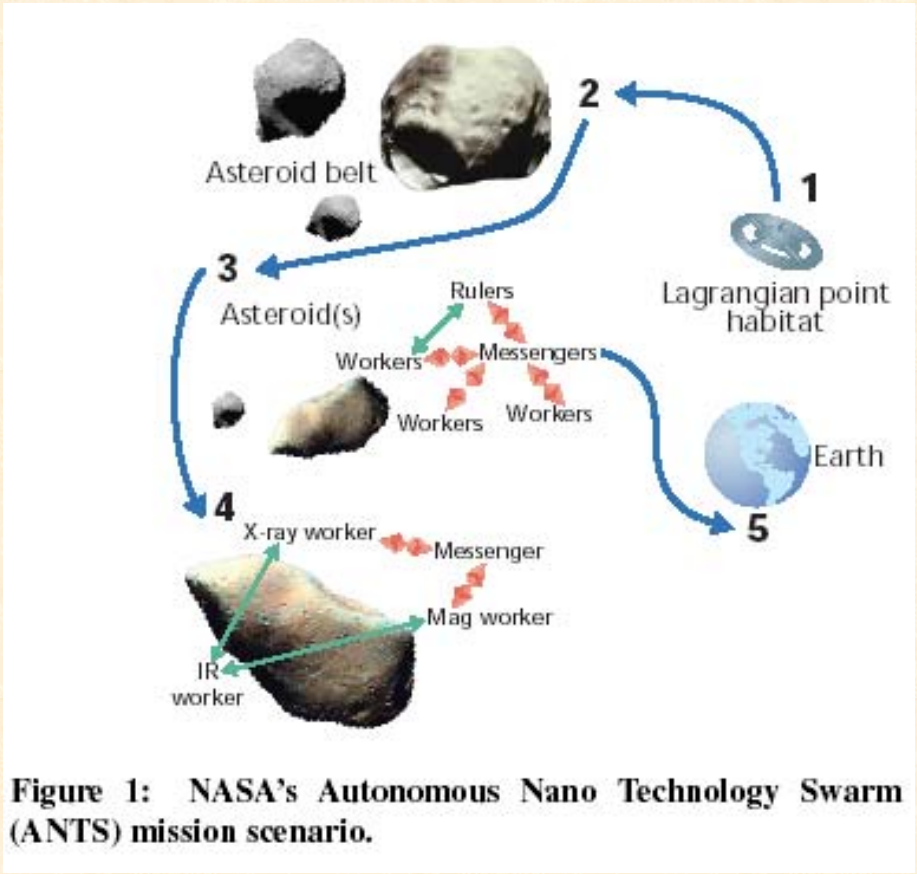


Figure 1: NASA's Autonomous Nano Technology Swarm (ANTS) mission scenario.

M. G. Hinchey et al FMICS'05, September 5–6, 2005, Lisbon, Portugal.

NASA Comparison of formal methods

Table 2: Comparison of integrated formal methods

Name	Concurrency Support	Algorithm Support	Tool Support	Formal Basis	Used in Agent-Based Specs.	Used in Swarm-Based Specs.
Communicating X-Machines	Yes	Yes	No	Yes	Yes	Yes
CSP-OZ	Yes	Yes	No	Yes	Yes	No
Object-Z and Statecharts	Yes	Yes	No	Yes	Yes	No
Temporal B	Yes	Yes	No	Yes	Yes	No
Temporal Petri Nets	Yes	No	No	Yes	Yes	No
Timed Communicating Object Z	Yes	Yes	No	Yes	Yes	No
Timed CSP	Yes	No	Yes	Yes	Yes	No
ZCCS	Yes	Yes	No	Yes	Yes	No

NASA Conclusions
Communicating X-machines + WSCCS

Swarm Formal Method Model and Outline

$$\Phi = \left\{ \begin{matrix} SendMessage, ReceiveMessage, \\ Reason, Process \end{matrix} \right\}$$

is a set of (partial) transition functions where each transition function maps

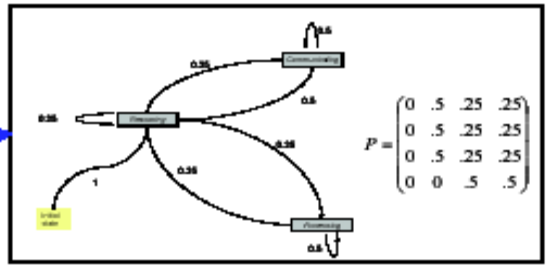
$$Memory \times Input \rightarrow Output \times Memory$$

$memory' = (Goals', Model', CommsTracking)$

[Communicating]ReasoningDeliberatve(Leader)[Reasoning]

[Reasoning] SendMessage (Leader,Worker)[Communicating]

[Processing] SendMessage (Leader, Worker) [Communicating]



$$P = \begin{pmatrix} 0 & .5 & .25 & .25 \\ 0 & .5 & .25 & .25 \\ 0 & .5 & .25 & .25 \\ 0 & 0 & .5 & .5 \end{pmatrix}$$

Agent State	Actions leading to the agent state	f	p
	Identity		
Communicating	SendMessageWorker	50	2
	SendMessageLeader	50	2
	SendMessageError	1	1
	ReceiveMessageWorker	50	2
	ReceiveMessageLeader	50	2
	ReceiveMessageError	1	1
Reasoning	ReasoningDeliberatve	50	2
	ReasoningReactive	50	2
Processing	ProcessingSortingAndStorage	17	2
	ProcessingGeneration	17	2
	ProcessingPrediction	17	2
	ProcessingDiagnosis	16	2
	ProcessingRecovery	16	2
	ProcessingRemediation	17	2

$$n\omega^{k+1} + m\omega^{k} = n\omega^{k+1} = m\omega^{k} + n\omega^{k+1}$$

$$n\omega^{k} + m\omega^{k} = (n+m)\omega^{k} = m\omega^{k} + n\omega^{k}$$

Refinements and Research question

- The hard part is writing unit tests before the code
- We want some teams to try to do this – we want to see if it produces better systems as Kent Beck says
- What makes it hard? Is it really hard?

References:

Kent Beck "Extreme programming explained." Addison-Wesley, 1999, 2005.

K. Beck & M. Fowler, "Planning extreme programming." Addison-Wesley, 2000.

Ron Jeffries, <http://www.XProgramming.com/>