Stijn Hoppenbrouwers

Requirements Engineering, Lecture 4:

Managing Requirements and People

&

Traceability

Radboud University Nijmegen

We'll talk about some highlights only now

- READ THE BOOK YOURSELF!
- Particularly good passage Nell and the Coffee Shop:
- *Feel* the advantages of short-cyclic, iterative development

- But if you have time, have a look at: http://noorderlicht.vpro.nl/afleveringen/3502225/ for a contrasting way of thinking. Who's "right"? Different underlying assumptions?

HI/I: Holistic Iterative/Incremental

- ***Methodologists in late 1990's combined available lifecycles.***
- ***Pronounced "hi-eye" (HI/I).***
- ***All use tools / techniques to keep up with massive changes in business today:***
  - ***Unified Process (UP)*** **(Rational** ***Univied Process: RUP)***
  - ***eXtreme Programming (XP)***
  - ***Adaptive SW Development (ASD)***
  - ***Agile SW Development (AgileSD)***
  - ***SCRUM***

Feedback Central

- ***All have rigorous focus on continuous feedback loops between development and its stakeholders (business).***
- ***Different from previous lifecycles in that they have built-in "holistic perspectives".***
- ***The view of the whole is never lost.***
- ***Accomplished with strong architectural vision, including business strategy.***

IRIS

Iterative (the “Totem Pole” metaphor)

- ***Iterative - redoing something several times, increasing richness, comprehensiveness and consistency each time.***
  - ***Not random hacking***
  - ***Not playpen for developers***
  - ***Not something that affects only developers***
  - ***Not redesigning over and over until chance fix found***
  - ***Not unpredictable***
  - ***Not excuse for failing to plan and manage***

IRIS

Incremental (the “House Construction” metaphor)

- ***Incremental – creating something piece by piece, integrating pieces into whole a little at a time.***
  - ***YET: following a (rough) design***
  - ***Also referred to as build-deliver-learn.***
  - ***In context for RE course: completing batches of use cases and business rules etc. together.***
  - ***Not all artefacts need to be completed together.***

IRIS

Holistic

- ***Holistic – keeping an eye on the big picture.***
- ***Architect must keep the end goal view in each iteration.***
- ***Business strategy alignment must also be kept in view, staying true to the business vision of the application.***
- ***This is found in your MVV.***

IRIS

Adaptivity

- **Adaptability – *capability of* being made *suitable or fitted to a specific situation (be adapted by* someone else*)***
- **Adaptivity – *capability of making suitable or fitting a specific situation. (adapt* yourself/itself*)***
- ***Is any software development team completely adaptive?***
- ***Very interesting research direction "Complex Adaptive Systems" http://www.caseresearch.com/***

IRIS

Development (of) systems

- What do “information systems” look like?
  - Computational?
  - Socio-technical?
- Can *fully automated* information systems exist?

- What is it that “creates” computational systems?
- What do “development systems” look like?
- Can *fully automated development systems* exist?

- What could I possibly mean by “2nd order information system”?

Wicked Problems (zie Proper's oratie)

- Je begrijpt een gemeen probleem pas goed als je er een oplossing voor hebt bedacht. Elke mogelijke oplossing brengt nieuwe aspecten van het probleem aan het licht, aspecten die verdere aanpassing van de oplossing vereisen.
- Gemene problemen hebben geen stopcriterium. Er is geen eenduidige en stabiele probleemdefinitie te geven. Als gevolg hiervan is het is niet duidelijk wanneer het probleem *echt* is opgelost.
- Oplossingen voor gemene problemen zijn niet simpelweg goed of fout. In plaats daarvan zijn ze "beter", "slechter" of "goed genoeg". Voor gemene problemen is het moeilijk om op een objectieve wijze de kwaliteit van een oplossing te beoordelen.
- Elke oplossing van een gemeen probleem krijgt slechts één kans. Elke realistische poging heeft direct consequenties. Je kunt niet eerst een Betuwelijn bouwen om te zien of het wel of niet zal werken in de praktijk.

Traceability

Definition

- **Traceability – “clarity of linkage between artifacts”.**
  - **Use cases can be tremendous tool.**
  - **We will take a look at what is good about traceability.**
  - **We will also look at trouble traceability can cause.**

Important

- Provides assurance that software matches stakeholders wishes at end of lifecycle
- **Why is traceability important?**
- If you were sponsoring a project, would you want to know at a given moment what the project team is working on
- And: *why* they are working on it?

Important

- Traceability is important because:
  - Everyone needs to feel like building what is supposed to be built.
  - Business often demands *provability* (that right thing is being built)
  - Large $$$ investments: one should be able to prove that work takes place on what is *promised* (in all phases).

- **IT'S ALL ABOUT CONTROL**

What and Why?

- Looking at the waterfall lifecycle.
- Imagine we are 60% through development...
- ...we show that we are working on something...
- ...but not that we are working on the RIGHT thing ...
- ...damn!

Changes

- When changes occur in project, we see impact rippling through lifecycle artefacts.
- This helps business understand costs of change.
- Helps team identify and change all related artefacts for a change.

Functionality

- **Stakeholders viewing system functions later in lifecycle see unexpected features.**
- **Traceability helps the decision making process that brought features into scope by making dependencies and consequences visible.**
- **Reduces “finger-pointing” (dealing of blame)**

Traceability is Hard

- **Stakeholders needs chang during lifecycle.**
- **Team members change**
- **A delivery cycle may get off track (especially if it is too long)**
- **Things are passed between team members as documents, not as hands-on knowledge**
- **Linkage is not easy between artefact types (for example, between contract-style req.'s and analysis/design artifacts).**

Traceability problems

Traceability supporting measures an create these problems if we are not careful:

- **Too many artefacts.**
- **Too many tools with too many integrations.**
- **Too many special 'gatekeeper' roles (bureaucrats)**

Traceability cures

- **Traceability can be helped by:**
  - **Use cases**
  - **Non-functionals**
  - **Short feedback loops between team and business stakeholders**
  - **Demonstrable “inch pebbles” (milestones, but for short cycles)**
  - **Integration between tools**
  - **Configuration managements**
  - **Egalitarianism (equality in the team)**
  - **See book, page 149-157 for more...**

Automated lifecycle Tools

- **Rational Suite (see book for details).**
- **eXpressroom**
- **Borland CodeWrite**
- **CompuWare**
- **Some freeware**
- **… !**

Tracing back to use cases

- **Use cases are the *central traceability artefact*.**
  - **Readable by business people, credible starting point.**
  - **Coarse grained, allowing large system to be divided into manageable parts.**
- **Use cases can have traceability links to:**
  - **Analysis/Design/Test models, sequence/class/state diagrams, planning documents, training documentation, etc...**
  - **See book for details.**

Analysis Model Traceability

- **Use cases provide easy traceability if OO analysis is used.**
- **Sequence diagram - use case steps translate to set of messages between objects.**
- **Class diagram – names are nouns in use case.**
- **State diagram – states are pre- post-conditions of use cases and transitions are messages from initiating actors or sequence diagrams.**
- **Analysis models contain entity classes. “Taking care of business rules and business processes”**

Test Model Traceability

- **Clearest traceability link is between use cases and test models:**
  - **Use cases ARE test cases!**
  - **Just moved to the front of lifecycle and called requirements**
  - **Test plan organized by use cases.**
  - **Add test data to use cases to become testable.**
  - **Test cases focus on each use case path.**

UI Design Traceability

- **UI design comes from requirements:**
  - **Use cases** PLUS **usability requirements (often non-functional).**
  - **UI storyboards use use cases as 'stories', screen mock-ups are user walk through of use case.**

Application Architecture

- **Provides the foundation for how components and objects are organized in target technical environment.**
- **Use cases help identify control objects or components.**
  - **Control objects manage other objects into transactions that are useful to users.**
  - **These transactions are patterned after use cases.**

Documentation / Training

- **Often overlooked traceability opportunity.**
- **Users learn application by use case, one story at a time.**
- **Organize end user training and user documentation by use case.**
- **Allows documentation and training professionals to fit into iterative/incremental lifecycle.**

Product Marketing

- **Use cases are an effective marketing tool.**
- **As new product is released, marketing has useful list of “stories to tell”.**
- **This subsequently provides new sections for product demonstration scripts.**