

Software on Internet Time ...and Time Again

Part I: Agile Development with Models

Marc J. Balcer



ModelCompilers.com
code at a higher level

Project Profiles

- Financial Services: 3 years, 50+ people
a mature project
- Logistics: 9 months, 7 people
just hitting its stride
- What do they have in common?



Business Characteristics

- Systems are core to business change
 - efficiency improvements
 - cost savings
 - heart of the business—
not just technology play

- Systems are unifying
 - across parts of a business
 - across business partners



Technical Characteristics

- Web application
 - easy to deploy
 - browser UI
 - used world-wide
 - J2EE and other buzzwords
- Web service
 - the applications “talk XML”
- Oh, and the team wants to do XP!



R.I.P. Analysis

- Has lightweight (“agile”) rendered formal analysis obsolete?
 - All that matters is code
 - Work in small iterations
 - Only write little storycards
 - Don’t waste time modeling, start coding today



Agile Approaches

- New breed of “methodologies” that have discipline without bureaucracy
- Most Popular Agile Approaches
 - XP (Extreme Programming)
 - Crystal / Highsmith ASD
 - Feature Driven Development
 - SCRUM
 - DSDM



Agile Manifesto

We Value:

Individuals and Interactions

over

Process and Tools

Working Software

over

Comprehensive Documents

Customer Collaboration

over

Contract Negotiation

Responding to Change

over

Following a Plan



Agile is Attractive

- to the development team...
 - promise of less bureaucracy
 - code-centric—what developers do best
 - instant gratification
- to the client
 - don't need to plan it all or know it all
 - gets to “drive”—adjust the plan
 - instant gratification



Agile and the Business Vision

- Client business has a long-term vision
 - not a detailed plan
 - vision will change over time
- Long-term vision does not mean long time to get something



Agile and the Development Approach

- Approach the big vision incrementally
- Plan the work as a series of three-month releases
- Further releases depend upon the business
 - How can business adopt change
 - Each release drives future requirements



Release I Objective

- Demonstrate value quickly
 - Solve a key business problem
 - Establish software architecture
 - Build working relationship and development practices
- Phase I really goes live—
it's not just demoware
 - Everything has real business value



Release I Objective

- Financial services:
**dealer enters a credit app on the web;
this is sent to a regional office**

- Shipping logistics:
**notify consignee and track shipment
through the “last mile”**



Release I Approach

- Time-boxed approach
 - Fix completion date & budget
(3 mo / \$750K)
- Manage requirements
 - Drive toward simplicity
and clarity of business vision
- Assume there will be more releases
 - Keep in mind the downstream impact
of the first release



And We're Off!

- Start the whole team on Day 1
 - customer
 - analysts
 - developers
- XP'ers say to start coding
 - but the customer doesn't know what they want
- "We should be writing storycards"



Agile Gets Scary

- to the development team...
 - where are my requirements?
 - the requirements keep changing...
 - someone is hiding something from me!
- to the client
 - didn't I just tell that to ____?
 - we're looking for the document with...
 - someone is hiding something from me!



eXtreme Zealotry Ahead!

- Agile, lightweight, XP, etc.
are reactions to heavyweight (overweight)
Methodologies
- Reaction or overreaction?
 - “You **can’t** do that, it’s not in the book”
 - “You **can’t** do that, you’re a Manager”
 - “We **can’t** tell you when it will be done”



Establish the Practices...

**Planning
Game**

**40 Hour
Week**

**Pair
Programming**

Metaphor

**Small
Releases**

On-site customer

**Simple
Design**

**Continuous
Integration**

**Coding
Standards**

Testing

Refactoring

**Collective
Ownership**

*Some are easier to
establish than others...*



...and have a Bootstrapping Plan

- Month 1: Understand
 - model application
 - prototype architecture using simple model
- Month 2: Develop
 - translate models into code
 - manage scope to meet targets (time-boxing!)
- Month 3: Deploy
 - early feedback is key
 - prepare to go live to selected real users

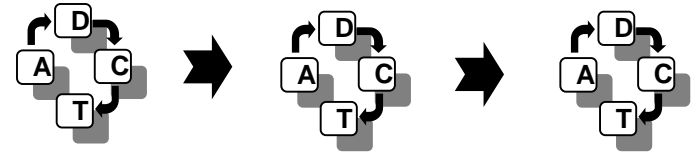


Release I Methodology

- Looks like traditional waterfall (design – code – test)

but:

- faster & shorter (3 months)
- tolerant of incompleteness
- anticipates a next release
- can be divided into smaller development increments



Models?!

- Good artifacts facilitate communication
 - in their production (thinking)
 - in their brevity (signal-to-noise ratio)
- Formal and Verifiable beats light and fluffy
 - picture beats 1000 words
 - models expose detail
 - follow XP testing practice:
Don't create anything that's not testable



Key Model Artifacts

- Domain chart
 - simple: app, UI, architecture, technologies
- Application domain models
 - information & state models
 - use cases & model tests
- Models of other domains
 - loose concept of UI model
 - architecture still ad hoc



Don't Underestimate Architecture!

- Architecture is often overlooked
- Most work in Phase I is architecture
 - developing mechanisms & structures does not require complete application
- Keep driving to consistency on architecture and service domains
 - find & use patterns; keep models
 - refactor mercilessly
 - look to generate boring & repetitive code



Code Generation

- Really doing it
 - J2EE entity bean classes & DB schema (from IM)
 - XML messages (from application events)
 - on-line project notebook
- Could do more
 - but requires “settling” on architecture
 - still knee-jerk “can’t be done”
- At least take on the boring & repetitive



The Date Approaches

- Stick to time and budget
- Manage scope
 - Watch out for scope creep
 - Don't let development overcomplicate
 - Stick to what was modeled
 - "YAGNI"
- Expect to deliver on time
 - Plan backwards from delivery
 - Dress rehearsals of deployment
 - Have a backup plan



...and you deliver! On to Release II

- Evaluate the first release (“retrospective”)
 - Functionality vs. business needs?
 - Architecture vs. performance?
 - Teams and roles?
- Reset direction
 - Release I will expose different requirements
 - Technology changes
 - People learn to work together



Release II Approach

- Deliver more in same (3 month) time
- Build upon Release I
 - architectural framework is there
 - overall application model is there
 - team development process is there
- Establish and prioritize needs
 - still can't do it all
 - stay realistic
- Keep to a timeboxed approach



Release II Methodology

- Different than pilot – we can do more
 - we can do more
 - we know more
 - we know how to do more
- Still involves tradeoffs
 - choose based on business value
- Still incremental
 - produce three one-month increments
 - development vs. customer increments



Agile Gets Confusing

- “I don’t see any difference between storycards and use cases”
 - tendency to revert to old habits
 - single-artifact methodology
“only tool a hammer—
every problem a nail”
 - over-reliance on by-the-book interpretation of a method
 - eXtreme Zealotry can cloud the issue



Incremental Workproducts

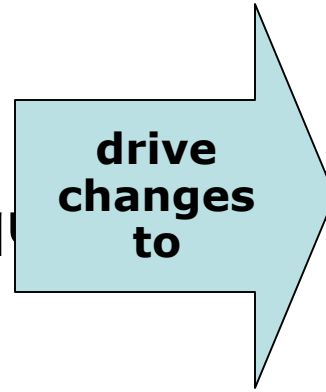
- Incremental development requires workproducts to:
 - capture the work for the increment
 - keep the overall picture
- Need for both kinds not well understood
 - Big-Bang development has no need for incremental workproducts
 - Incremental (XP in particular) tends to ignore all but code



Incremental vs. Ongoing

■ Incremental Workproducts

- requirements
- bug reports
- enhancement reqs
- "storycards"
- work plans

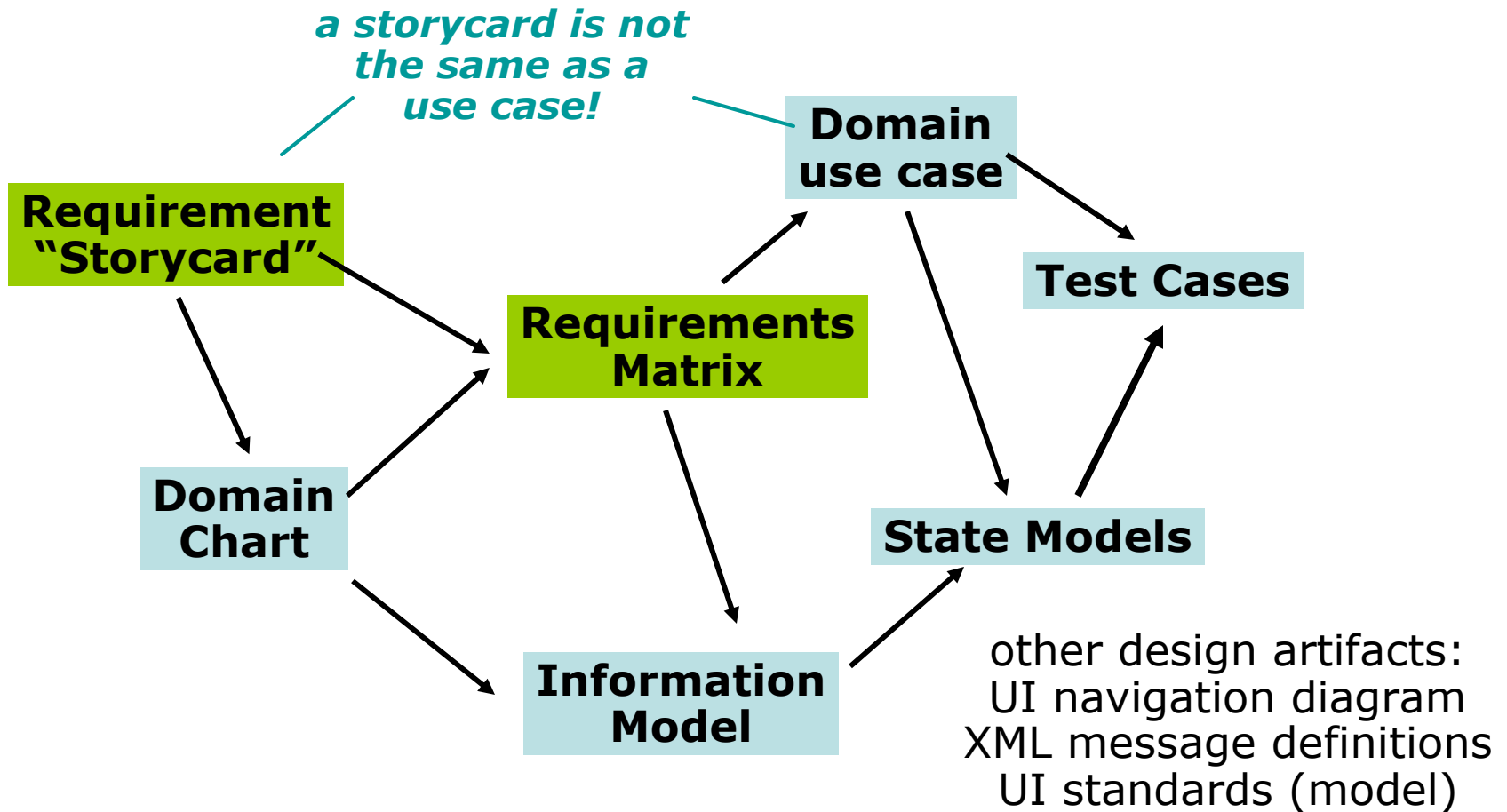


■ Ongoing Workproducts

- domain chart
- use cases
- domain models
 - information model
 - state models
- test cases
- code



Process Flow



Service Domains

- Code refactoring generally identifies intermediate abstractions
- Refactor the domain partitioning



After 6 months

- We have
 - a working system
so business is recovering costs
 - a delivery partnership
continually refined and sustainable
- Building a system has sharpened focus in a way no mere “planning” could do.
- Customer knows what they’re getting



After 6 months

- We also have a set of models
 - collective knowledge
 - means for planning next steps
- We have useful service domains
 - your lifeline if you're in consulting
- We have a powerful architecture and toolset

- Most importantly...



We are Agile and SMUG

- Don't forget what you know already!
- Executable UML Models are like code
 - they are testable
 - they can be developed incrementally
- Models are better than code alone
 - they help maintain collective memory
 - they help manage scope

