Agile Development in the 'Old Economy'

Between the Desirable and Reality

Ove Armbrust

March 13, 2013

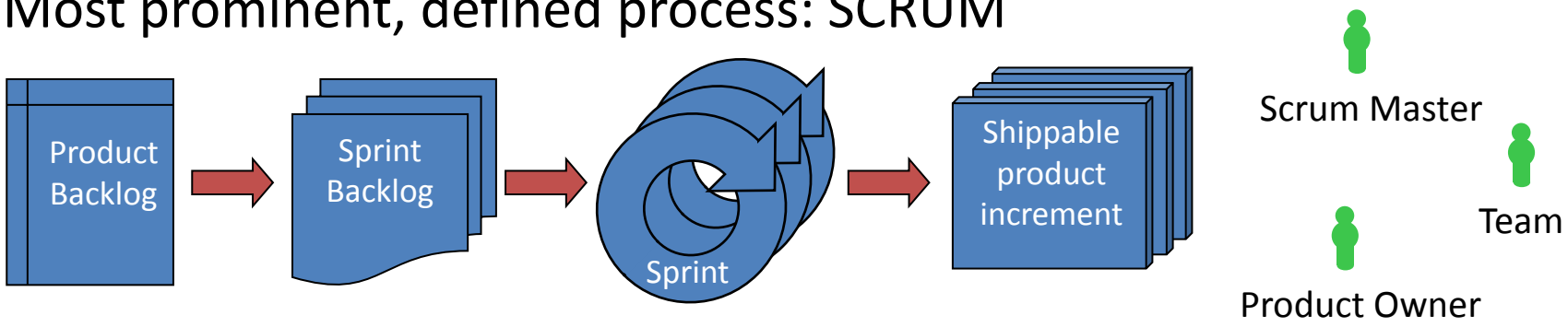
Contents

- Why use agile development at all?

- Challenge #1: external context
- Challenge #2: internal context
- Challenge #3: documentation

- Conclusions

- In terms of the Agile Manifesto
  - Self-organization – no central “project manager” instance
  - Co-location – global knowledge shared amongst all team members
  - Pair programming – two developers coding together
  - Customer collaboration – daily interaction, shared ownership
  - Incomplete requirements – changes in delivered product as they emerge
  - Rapid delivery – release early, release often
  - Limited documentation – “the documentation is in the code”
- Most prominent, defined process: SCRUM



Why agile development in classic non-software domains?

- Many tangible products contain major software portions
- Software accounts by far for most user-visible innovation
- Massive, late changes are the norm for software
- Software development must accommodate this
  - Short innovation/release cycles
  - Incorporate new features late during development
- Agile approaches facilitate that – so what is the problem?

Challenges #1: non-agile context (external)

- Customers in “real-goods” industry often don’t think software
  - They have a washer, an ECU, a head unit developed – not software
  - They think in tangible goods development
  - They are not from a software domain – they are concerned with hardware availability, production costs, and profit per unit sold
  - “Software can be changed at any time, with little effort, because it’s just **soft**ware”
- Customers often don’t know (or understand, or care about) software development lifecycles
  - In particular, agile approaches with constant, direct customer involvement
  - They are not used to being involved in development on a daily basis – possibly on site – and typically don’t support this approach

How to cope with a non-agile context (external)

- Teach the customer some software development lifecycle
  - Being able to change direction every month comes at a price
  - Make that price understandable and explicit
- Replicate the customer locally as part of the team
  - Dedicated team member who speaks for the customer
  - Is in constant (daily!) contact with the customer
  - Customer still needs to be willing to cooperate so closely – and allocate time for this
  - Costly solution – uses engineering resources to do the customer's work

Challenges #2: non-agile context (internal)

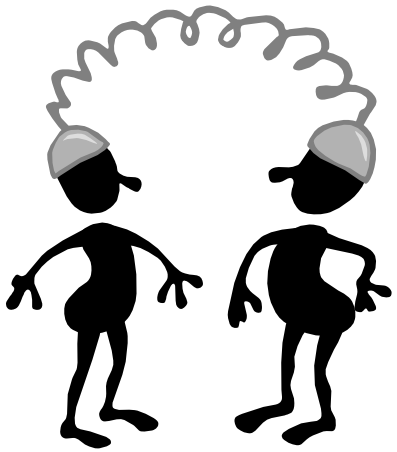
- "Pair programming wastes two developers to do the job of one"
- Co-located development not always possible (global teams)
- Team focusing on one project hard to achieve, particularly in small organizations
- Agile development transfers significant responsibility to the developers and the team
  - Team self-management increases freedom and responsibility for team members – not everybody is comfortable with this
  - Team self-management means loss of control for managers – not always accepted

How to cope with a non-agile context (internal)

- True agile development (or SCRUM) means a radical change to the complete organization, including titles, roles, management
- Major amounts of money, time, effort required for transformation
- If this is not understood and supported 100% by all levels: DON'T TRY TO FORCE IT
- Instead: build a process that support your goals
  - Highly iterative project lifecycles (sprints), frequent customer releases
  - Have a project manager – but estimate & schedule with complete team
  - Daily short meetings for communication
  - …

Challenge #3: is vital documentation omitted?

- Agile approaches are not only seen as better suited to react quickly to changed requirements, but also as delivering more product in less time
  - This is achieved in part by omitting some documentation – “cut the slack”
  - E.g., design documentation is limited
  - But: classic system design incorporates reviews, for example for scalability
  - What if there is no (complete) design to review?
- Global knowledge of development team may lead to not documenting “trivial” aspects of the software
  - But: what if the software has to be maintained for 10 years, by a different team?



How to cope with documentation omission

- Anticipate insufficient documentation right from the beginning
  - Define a feature as “completed” not as “there is working code”, but also considering a minimal design documentation, reviews, …
  - Have technical writers on the team, creating such documentation
- Plan consolidation & documentation sprints
  - After ever 3rd/4th/5th/… feature development sprint, there is a mandatory consolidation sprint where no new features are implemented
- If project time until deliver does not permit consolidation sprints, plan them after delivery
  - I.e., the team can’t start with the next project right away!
- All of the above require great discipline

- Internal agile development challenges can be addressed
  - But at high costs, and requiring major organizational changes
- External challenges are often outside our control
  - We can't force customers to work and think SCRUM-compliant
  - This by itself prevents the application of approaches like SCRUM in their entirety
- It is feasible and beneficial to adapt some agile practices to yield their benefits
- Agile/sprint-based approaches require more self-discipline by team members than classic approaches

Questions?

Contact

Dr. Ove Armbrust

ove.armbrust@gmail.com

(310) 850-0793