Construction, testing, deployment & maintenance in plan-driven vs agile software development processes. DevOps (development + operations)

INF2-SEPP

Adriana Sejfia
School of Informatics, University of Edinburgh

Quite a few lectures ago…

- The basics of plan-driven vs agile processes (Lecture 2).
- Requirements engineering in plan-driven vs agile processes (Lecture 6)
- Design in plan-driven vs agile processes (Lecture 12)

Last few lectures

- Construction
- Verification, validation and testing
- Deployment and maintenance

in general.

This lecture

- Construction (implementation) in plan-driven vs agile processes:
    - Differences in process
    - Some myths about agile and high-quality code
- Testing in plan-driven vs agile
    - Differences in process
    - Some myths about agile and high software quality
- Differences between plan-driven and agile teams in terms of construction and testing
- Deployment and maintenance in plan-driven vs agile processes and the emergence of DevOps (Development + Operations)
- DevOps advantages, principles, automation

Construction: differences in process between plan-driven and agile

- Plan-driven software development processes:
  - Each SE activity is a separate stage

Construction: differences in process between plan-driven and agile

- Plan-driven software development processes:
  - Each SE activity is a separate stage
  - Construction (implementation) is one of the stages

Construction: differences in process between plan-driven and agile

- Plan-driven software development processes:
  - Each SE activity is a separate stage
  - Construction (implementation) is one of the stages

- Agile software development processes:
  - Construction (as well as design) main focus of each iteration because prioritisation of 'working software'

Construction: differences in process between plan-driven and agile

- Plan-driven software development processes:
  - Each SE activity is a separate stage
  - Construction (implementation) is one of the stages

- Agile software development processes:
  - Construction (as well as design) main focus of each iteration because prioritisation of 'working software'
  - Requirements engineering interleaved with design and construction in each iteration

Construction: differences in process between plan-driven and agile

- Plan-driven software development processes:
  - Each SE activity is a separate stage
  - Construction (implementation) is one of the stages

- Agile software development processes:
  - Construction (as well as design) main focus of each iteration because prioritisation of 'working software'
  - Requirements engineering interleaved with design and construction in each iteration
  - Each iteration concerned with developing a subset of features

Myths about agile and high-quality code

- Myth 1: 'Code is the only documentation', concluded from 'Working software over comprehensive documentation’

Myths about agile and high-quality code

- Myth 1: 'Code is the only documentation', concluded from 'Working software over comprehensive documentation'
  - Wrong!

Myths about agile and high-quality code

- Myth 1: 'Code is the only documentation', concluded from 'Working software over comprehensive documentation'
  - Wrong!
  - The point is that effort should be made into making code as readable and clear as possible

Myths about agile and high-quality code

- Myth 1: 'Code is the only documentation', concluded from 'Working software over comprehensive documentation'
  - Wrong!
  - The point is that effort should be made into making code as readable and clear as possible
  - Of course, not everybody does this well!

Myths about agile and high-quality code

- Myth 1: 'Code is the only documentation', concluded from 'Working software over comprehensive documentation'
  - Wrong!
  - The point is that effort should be made into making code as readable and clear as possible
  - Of course, not everybody does this well!
  - Documentation still exists, but it is kept 'lightweight'

Myths about agile and high-quality code

- Myth 1: 'Code is the only documentation', concluded from 'Working software over comprehensive documentation'
  - Wrong!
  - The point is that effort should be made into making code as readable and clear as possible
  - Of course, not everybody does this well!
  - Documentation still exists, but it is kept 'lightweight'

- See this article by Fowler: https://martinfowler.com/bliki/CodeAsDocumentation.html

Myths about agile and high-quality code

- Myth 2: 'In agile, construction leads to better code quality'

Myths about agile and high-quality code

- Myth 2: 'In agile, construction leads to better code quality'
    - Wrong!

Myths about agile and high-quality code

- Myth 2: 'In agile, construction leads to better code quality'
  - Wrong!
  - Agile in general has nothing to do with code quality

Myths about agile and high-quality code

- Myth 2: 'In agile, construction leads to better code quality'
  - Wrong!
  - Agile in general has nothing to do with code quality
  - Some agile processes (e.g. Scrum) focus on the quality of the process, not that of the technical quality

Myths about agile and high-quality code

- Myth 2: 'In agile, construction leads to better code quality'
  - Wrong!
  - Agile in general has nothing to do with code quality
  - Some agile processes (e.g. Scrum) focus on the quality of the process, not that of the technical quality
  - Exception: XP (coding standards, pair programming practices)

Myths about agile and high-quality code

- Myth 2: 'In agile, construction leads to better code quality'
  - Wrong!
  - Agile in general has nothing to do with code quality
  - Some agile processes (e.g. Scrum) focus on the quality of the process, not that of the technical quality
  - Exception: XP (coding standards, pair programming practices)
  - Risk of iterative nature of process to degrade the design and quality of code

Myths about agile and high-quality code

- Myth 2: 'In agile, construction leads to better code quality'
  - Wrong!
  - Agile in general has nothing to do with code quality
  - Some agile processes (e.g. Scrum) focus on the quality of the process, not that of the technical quality
  - Exception: XP (coding standards, pair programming practices)
  - Risk of iterative nature of process to degrade the design and quality of code

- One of the authors of the Agile Manifesto, Robert Martin, even wrote code quality books, e.g. "Clean Code: A Handbook of Agile Software Craftsmanship"

Testing: differences in process between plan-driven and agile

- Plan-driven software development processes:
  - Testing is a a stage in itself

Testing- differences in process between plan-driven and agile

- Plan-driven software development processes:
  - Testing is a a stage in itself
  - Often done late in the software development, when anything is expensive to fix

Testing- differences in process between plan-driven and agile

- Plan-driven software development processes:
  - Testing is a a stage in itself
  - Often done late in the software development, when anything is expensive to fix
  - Black box tests are based on requirements that have been written (usually) by requirements engineers or business analysts long before starting to write any code

Testing- differences in process between plan-driven and agile

- Plan-driven software development processes:
  - Testing is a a stage in itself
  - Often done late in the software development, when anything is expensive to fix
  - Black box tests are based on requirements that have been written (usually) by requirements engineers or business analysts long before starting to write any code
  - Testing often ends up being 'squished' at the very end, risking not to be done thoroughly, because of delays and approaching deadline

Testing: differences in process between plan-driven and agile

- Agile software development processes:
  - At the end of each small iteration the code is tested

Testing: differences in process between plan-driven and agile

- Agile software development processes:
  - At the end of each small iteration the code is tested
  - Many (but not all!) agile teams nowadays use the test-driven development (TDD) approach, coming up with tests only days/hours before the coding of the next features

Testing: differences in process between plan-driven and agile

- Agile software development processes:
  - At the end of each small iteration the code is tested
  - Many (but not all!) agile teams nowadays use the test-driven development (TDD) approach, coming up with tests only days/hours before the coding of the next features
  - Use of functional tests derived from business experts

Testing: differences in process between plan-driven and agile

- Agile software development processes:
  - At the end of each small iteration the code is tested
  - Many (but not all!) agile teams nowadays use the test-driven development (TDD) approach, coming up with tests only days/hours before the coding of the next features
  - Use of functional tests derived from business experts
  - Acceptance testing and feedback from users at end of each iteration

Testing: differences in process between plan-driven and agile

- Agile software development processes:
  - At the end of each small iteration the code is tested
  - Many (but not all!) agile teams nowadays use the test-driven development (TDD) approach, coming up with tests only days/hours before the coding of the next features
  - Use of functional tests derived from business experts
  - Acceptance testing and feedback from users at end of each iteration
  - Increased need for test automation to speed up testing and development

Some myths about agile, testing and high software quality

Myth 1: Some testers fear to transition to agile development because it 'equates with chaos, lack of discipline, lack of documentation, and an environment that is hostile to testers'

Some myths about agile, testing and high software quality

Myth 1: Some testers fear to transition to agile development because it 'equates with chaos, lack of discipline, lack of documentation, and an environment that is hostile to testers'

- Wrong for companies doing agile well

Some myths about agile, testing and high software quality

Myth 1: Some testers fear to transition to agile development because it 'equates with chaos, lack of discipline, lack of documentation, and an environment that is hostile to testers'

- Wrong for companies doing agile well
- True agile is about valuing team members, repeatable quality, efficiency.

Some myths about agile, testing and high software quality

Myth 2: 'Agile is all about speed'

Some myths about agile, testing and high software quality

Myth 2: 'Agile is all about speed'

- Wrong for companies doing agile well

Some myths about agile, testing and high software quality

Myth 2: 'Agile is all about speed'

- Wrong for companies doing agile well
- Agile is intended to be all about producing high-quality software in a time that maximizes its business value

Some myths about agile, testing and high software quality

Myth 2: 'Agile is all about speed'

- Wrong for companies doing agile well
- Agile is intended to be all about producing high-quality software in a time that maximizes its business value

- Recommended reading: Lisa Crispin: “Agile Testing: A Practical Guide For Testers And Agile Teams” Chapter 1.

Differences between plan-driven and agile teams in terms of construction and testing

- Teams in plan-driven software development processes:

Differences between plan-driven and agile teams in terms of construction and testing

- Teams in plan-driven software development processes:
  - Individual programmers working on their own part of the code

Differences between plan-driven and agile teams in terms of construction and testing

- Teams in plan-driven software development processes:
  - Individual programmers working on their own part of the code
  - Developers needing to follow the requirements, and having no say about the features and functionality

Differences between plan-driven and agile teams in terms of construction and testing

- Teams in plan-driven software development processes:
  - Individual programmers working on their own part of the code
  - Developers needing to follow the requirements, and having no say about the features and functionality
  - Testers seen as primarily responsible for quality, with limited control of how the code is written

Differences between plan-driven and agile teams in terms of construction and testing

- Teams in agile software development processes:

Differences between plan-driven and agile teams in terms of construction and testing

- Teams in agile software development processes:
  - Whole-team approach to development and testing, with common goal of producing high-quality software

Differences between plan-driven and agile teams in terms of construction and testing

- Teams in agile software development processes:
  - Whole-team approach to development and testing, with common goal of producing high-quality software
  - Team members seen as important within the team

Differences between plan-driven and agile teams in terms of construction and testing

- Teams in agile software development processes:
  - Whole-team approach to development and testing, with common goal of producing high-quality software
  - Team members seen as important within the team
  - Constant collaboration between programmers, testers, (potential) customers, other experts

Deployment and maintenance in plan-driven vs agile processes and the emergence of DevOps

Traditionally, development, deployment and customer support different teams, including in companies adopting agile. Maintenance also potentially a separate team. Effects:

Deployment and maintenance in plan-driven vs agile processes and the emergence of DevOps

Traditionally, development, deployment and customer support different teams, including in companies adopting agile. Maintenance also potentially a separate team. Effects:

- Communication delays between teams, leading to days until bugs or security vulnerabilities addressed and a new release made available to customers

Deployment and maintenance in plan-driven vs agile processes and the emergence of DevOps

Traditionally, development, deployment and customer support different teams, including in companies adopting agile. Maintenance also potentially a separate team. Effects:

- Communication delays between teams, leading to days until bugs or security vulnerabilities addressed and a new release made available to customers
- Different tools and skill sets between teams, not understanding each other, culture of mistrust

Deployment and maintenance in plan-driven vs agile processes and the emergence of DevOps

Traditionally, development, deployment and customer support different teams, including in companies adopting agile. Maintenance also potentially a separate team. Effects:

- Communication delays between teams, leading to days until bugs or security vulnerabilities addressed and a new release made available to customers
- Different tools and skill sets between teams, not understanding each other, culture of mistrust

Even in agile teams, not enough collaboration between development and operations (deployment and support) teams, and not enough documentation to fall back to.

Deployment and maintenance in plan-driven vs agile processes and the emergence of DevOps

- Solution: in 2008, Andrew Clay and Patrick Debois came up with DevOps (Development+Operations) set of practices.

Deployment and maintenance in plan-driven vs agile processes and the emergence of DevOps

- Solution: in 2008, Andrew Clay and Patrick Debois came up with DevOps (Development+Operations) set of practices.

- DevOps has become quite a buzzword, and is considered an outgrowth or extension of agile development beyond code.

Deployment and maintenance in plan-driven vs agile processes and the emergence of DevOps

- Solution: in 2008, Andrew Clay and Patrick Debois came up with DevOps (Development+Operations) set of practices.

- DevOps has become quite a buzzword, and is considered an outgrowth or extension of agile development beyond code.

- It is particularly suited to software products.

DevOps: overview, advantages

DevOps= the integration of development and operations (deployment, customer support) within a single team responsible for all these activities.

DevOps: overview, advantages

DevOps= the integration of development and operations (deployment, customer support) within a single team responsible for all these activities.

- Advantages
  - Faster deployment, due to reduced communication delays

DevOps: overview, advantages

DevOps= the integration of development and operations (deployment, customer support) within a single team responsible for all these activities.

- Advantages
  - Faster deployment, due to reduced communication delays
  - Reduced risk: the possibility to produce frequent increments for release, and identify causes of failures and outages easier

DevOps: overview, advantages

DevOps= the integration of development and operations (deployment, customer support) within a single team responsible for all these activities.

- Advantages
  - Faster deployment, due to reduced communication delays
  - Reduced risk: the possibility to produce frequent increments for release, and identify causes of failures and outages easier
  - Faster repair: all working together rather than waiting for responsible team to fix each issue

DevOps: overview, advantages

DevOps= the integration of development and operations (deployment, customer support) within a single team responsible for all these activities.

- Advantages
  - Faster deployment, due to reduced communication delays
  - Reduced risk: the possibility to produce frequent increments for release, and identify causes of failures and outages easier
  - Faster repair: all working together rather than waiting for responsible team to fix each issue
  - More productive teams; IMPORTANT! This heavily relies on organisational culture and the need for mutual respect and sharing.

DevOps principles

- Everyone is responsible for everything

DevOps principles

- Everyone is responsible for everything
- Everything that can be automated should be automated

DevOps principles

- Everyone is responsible for everything
- Everything that can be automated should be automated
- Measure first, change later: DevOps processes and tools continually adapted based on the data collected.

DevOps automation

- Continuous integration (from XP)= creating and testing an integrated version of the system each time a change is committed to the shared code repository.

DevOps automation

- Continuous integration (from XP)= creating and testing an integrated version of the system each time a change is committed to the shared code repository.
- Continuous delivery= testing the system in a replica of a production environment to ensure that environmental factors do not lead to bugs, each time a change is made.

DevOps automation

- Continuous integration (from XP)= creating and testing an integrated version of the system each time a change is committed to the shared code repository.
- Continuous delivery= testing the system in a replica of a production environment to ensure that environmental factors do not lead to bugs, each time a change is made.
- Continuous deployment (for cloud-based systems only)= deploying a system as a cloud service after each change is made to it.

DevOps automation

- Continuous integration (from XP)= creating and testing an integrated version of the system each time a change is committed to the shared code repository.
- Continuous delivery= testing the system in a replica of a production environment to ensure that environmental factors do not lead to bugs, each time a change is made.
- Continuous deployment (for cloud-based systems only)= deploying a system as a cloud service after each change is made to it.
- Infrastucture as code= automatically updating software on a company's servers using a model of the infrastructure written in machine-processable language.

DevOps automation

- Continuous integration (from XP)= creating and testing an integrated version of the system each time a change is committed to the shared code repository.
- Continuous delivery= testing the system in a replica of a production environment to ensure that environmental factors do not lead to bugs, each time a change is made.
- Continuous deployment (for cloud-based systems only)= deploying a system as a cloud service after each change is made to it.
- Infrastucture as code= automatically updating software on a company's servers using a model of the infrastructure written in machine-processable language.
- Homework: read advantages from Sommervile ESP Chapter 10.

Reading

- Essential: This article by Martin Fowler: https://martinfowler.com/bliki/CodeAsDocumentation.html
- Recommended if you can get a copy of the book: Lisa Crispin: “Agile Testing: A Practical Guide For Testers And Agile Teams” Chapter 1.
- Essential: Sommervile ESP Chapter 10 apart from 10.1.1