How Much Undocumented Knowledge is there in Agile Software Development?

S. Saito, Y. Iimura, A.K. Massey, A. Anton

Presented by
Shane McCulley

roadmap

- ➔ motivation
- ➔ project and case study
- ➔ analysis of undocumentation
- ➔ conclusions

The**AGILE**Manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Kent**Beck** Mike**Beedle** Arie**vanBennekum** Alistair**Cockburn**
Ward**Cunningham** Martin**Fowler** James**Grenning** Jim**Highsmith**
Andrew**Hunt** Ron**Jefferies** Jon**Kern** Brian**Marick** Robert**C.Martin**
Steve**Mellor** Ken**Schwaber** Jeff**Sutherland** Dave**Thomas**

12 Principles of Agile Software

01 Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

02 Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

03 Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

04 Business people and developers must work together daily throughout the project.

05 Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

06 Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

07 Working software is the primary measure of progress.

08 The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

09 Continuous attention to technical excellence and good design enhances agility.

10 Simplicity—the art of maximizing the amount of work not done—is essential.

11 The best architectures, requirements, and designs emerge from self-organizing teams.

12 At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

The**AGILE**Manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Kent**Beck** Mike**Beedle** Arie**vanBennekum** Alistair**Cockburn**
Ward**Cunningham** Martin**Fowler** James**Grenning** Jim**Highsmith**
Andrew**Hunt** Ron**Jefferies** Jon**Kern** Brian**Marick** Robert**C.Martin**
Steve**Mellor** Ken**Schwaber** Jeff**Sutherland** Dave**Thomas**

12 Principles of Agile Software

01 Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

02 Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

03 Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

04 Business people and developers must work together daily throughout the project.

05 Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

06 Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

07 Working software is the primary measure of progress.

08 The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

09 Continuous attention to technical excellence and good design enhances agility.

10 Simplicity—the art of maximizing the amount of work not done—is essential.

11 The best architectures, requirements, and designs emerge from self-organizing teams.

12 At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

The**AGILE**Manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Kent**Beck** Mike**Beedle** Arie**vanBennekum** Alistair**Cockburn**
Ward**Cunningham** Martin**Fowler** James**Grenning** Jim**Highsmith**
Andrew**Hunt** Ron**Jefferies** Jon**Kern** Brian**Marick** Robert**C.Martin**
Steve**Mellor** Ken**Schwaber** Jeff**Sutherland** Dave**Thomas**

12 Principles of Agile Software

01 Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

02 Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

03 Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

04 Business people and developers must work together daily throughout the project.

05 Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

06 Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

07 Working software is the primary measure of progress.

08 The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

09 Continuous attention to technical excellence and good design enhances agility.

10 Simplicity—the art of maximizing the amount of work not done—is essential.

11 The best architectures, requirements, and designs emerge from self-organizing teams.

12 At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

The**AGILE**Manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Kent**Beck** Mike**Beedle** Arie**vanBennekum** Alistair**Cockburn**
Ward**Cunningham** Martin**Fowler** James**Grenning** Jim**Highsmith**
Andrew**Hunt** Ron**Jefferies** Jon**Kern** Brian**Marick** Robert**C.Martin**
Steve**Mellor** Ken**Schwaber** Jeff**Sutherland** Dave**Thomas**

12 Principles of Agile Software

01 Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

02 Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

03 Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

04 Business people and developers must work together daily throughout the project.

05 Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

06 Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

07 Working software is the primary measure of progress.

08 The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

09 Continuous attention to technical excellence and good design enhances agility.

10 Simplicity—the art of maximizing the amount of work not done—is essential.

11 The best architectures, requirements, and designs emerge from self-organizing teams.

12 At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

The**AGILE**Manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Kent**Beck** Mike**Beedle** Arie**vanBennekum** Alistair**Cockburn**
Ward**Cunningham** Martin**Fowler** James**Grenning** Jim**Highsmith**
Andrew**Hunt** Ron**Jefferies** Jon**Kern** Brian**Marick** Robert**C.Martin**
Steve**Mellor** Ken**Schwaber** Jeff**Sutherland** Dave**Thomas**

12 Principles of Agile Software

01 Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

02 Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

03 Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

04 Business people and developers must work together daily throughout the project.

05 Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

06 Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

07 Working software is the primary measure of progress.

08 The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

09 Continuous attention to technical excellence and good design enhances agility.

10 Simplicity—the art of maximizing the amount of work not done—is essential.

11 The best architectures, requirements, and designs emerge from self-organizing teams.

12 At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

The**AGILE**Manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Kent**Beck** Mike**Beedle** Arie**vanBennekum** Alistair**Cockburn**
Ward**Cunningham** Martin**Fowler** James**Grenning** Jim**Highsmith**
Andrew**Hunt** Ron**Jefferies** Jon**Kern** Brian**Marick** Robert**C.Martin**
Steve**Mellor** Ken**Schwaber** Jeff**Sutherland** Dave**Thomas**

12 Principles of Agile Software

01 Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

02 Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

03 Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

04 Business people and developers must work together daily throughout the project.

05 Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

06 Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

07 Working software is the primary measure of progress.

08 The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

09 Continuous attention to technical excellence and good design enhances agility.

10 Simplicity—the art of maximizing the amount of work not done—is essential.

11 The best architectures, requirements, and designs emerge from self-organizing teams.

12 At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Documentation is part of Agile
Documentation is part of Agile
Documentation is part of Agile
Documentation is part of Agile
Documentation is part of Agile
Documentation is part of Agile
Documentation is part of Agile
Documentation is part of Agile
Documentation is part of Agile
Documentation is part of Agile
Documentation is part of Agile

implementation vs documentation

➔ **Do standard agile approaches properly document required knowledge?**

implementation vs documentation

- ➔ Do standard agile approaches properly document required knowledge?
  - **cooperative game: current vs. future**

implementation vs documentation

- ➔ Do standard agile approaches properly document required knowledge?
  - **cooperative game: current vs. future**

A. Cockburn. Agile Software Development: The Cooperative Game, Addison-Wesley Professional, 2006, 2nd edition.

implementation vs documentation

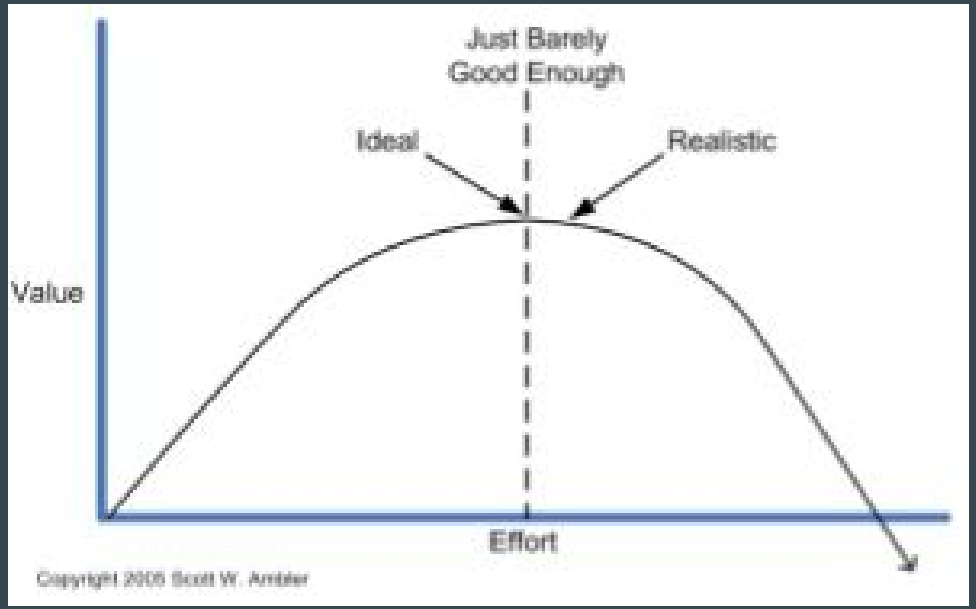
- ➔ Do standard agile approaches properly document required knowledge?
  - cooperative game: current vs. future

- ➔ **Analysis -- missing documentation versus unnecessary/unused documentation**

Agile Manifesto meets Project Requirements

Documentation should:

1. Start conversations
2. Lightweight
3. Capture requirements
4. Be dynamic



software artifacts

➔ **What counts as documentation?**

software artifacts

→ **What counts as documentation?**

Phase	Type	Artifacts	Volume
Planning/ Inception	Documents	User Stories	30 pages
		Algorithms and User Interface Sketches	15 pages
Iterations 1-2	Source code (Java)	Main Programs	5,931 steps
		Test Programs	2,093 steps
	Documents	Tickets	102 tickets
		User Manual	54 pages

software artifacts

- ➔ What counts as documentation?
  - **How are issues tracked?**

software artifacts

➔ What counts as documentation?

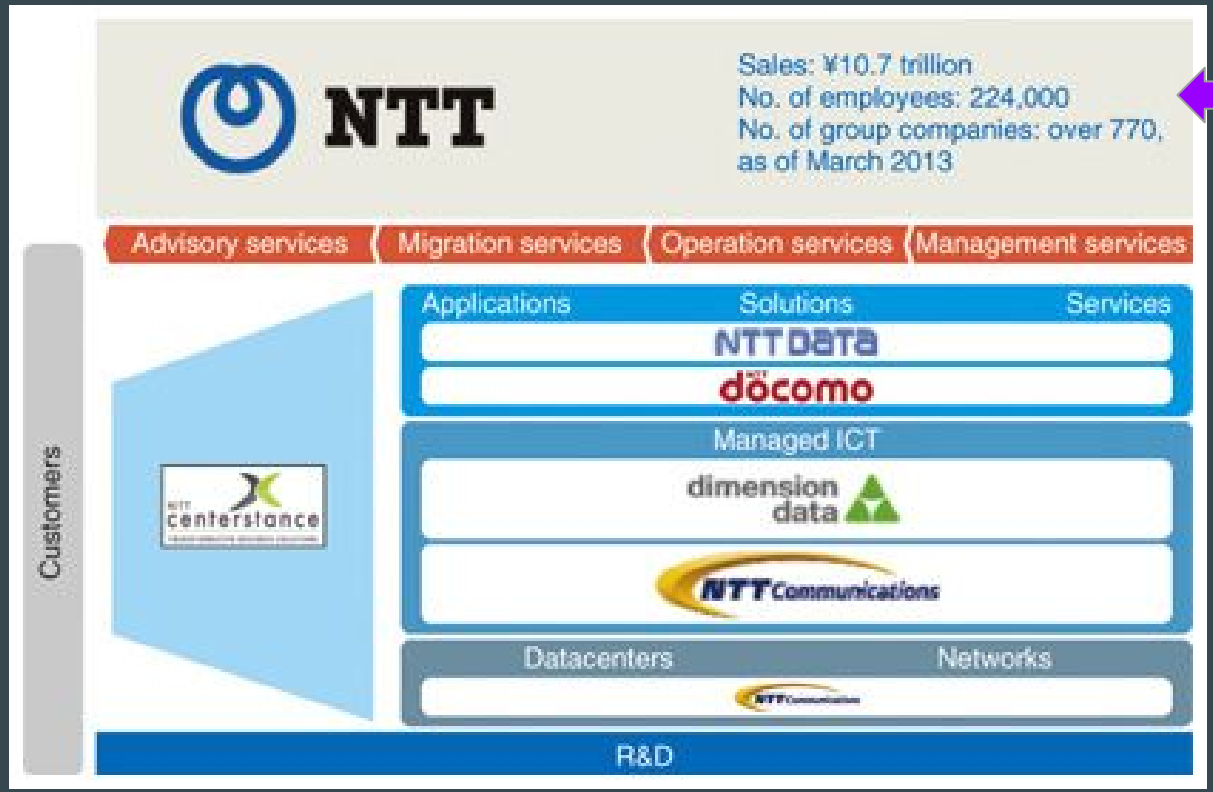
- How are issues tracked?
- **How is code maintained?**

goal

Examine undocumented knowledge in agile development at NTT Labs

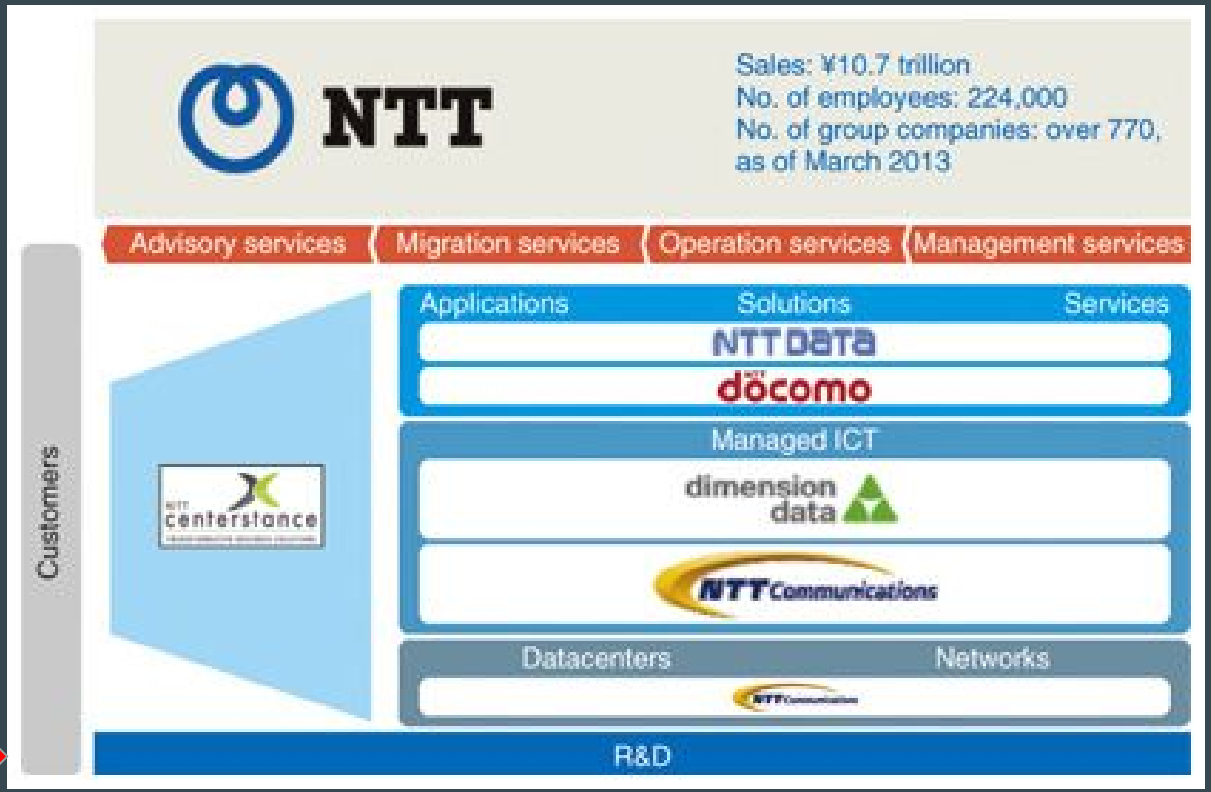
goal


NTT
Sales: ¥10.7 trillion
No. of employees: 224,000
No. of group companies: over 770,
as of March 2013
Advisory services
Migration services
Operation services
Management services
Customers
NTT centerstance
Applications
Solutions
Services
NTT DATA
docomo
Managed ICT
dimension data
NTT Communications
Datacenters
Networks
R&D


NTT
Sales: ¥10.7 trillion
No. of employees: 224,000
No. of group companies: over 770,
as of March 2013
Advisory services
Migration services
Operation services
Management services
Customers
Applications
Solutions
Services
NTT DATA
docomo
Managed ICT
dimension data
NTT Communications
Datacenters
Networks
R&D

goal

goal


NTT
Sales: ¥10.7 trillion
No. of employees: 224,000
No. of group companies: over 770,
as of March 2013
Advisory services
Migration services
Operation services
Management services
Customers
Applications
Solutions
Services
NTT DATA
docomo
Managed ICT
dimension data
NTT Communications
Datacenters
Networks
R&D

case study

- ➔ NTT laboratories
  - ◆ 3000 researchers

case study

➔ NTT laboratories

- 3000 researchers
- 100 software development projects annually

case study

➔ NTT laboratories

- 3000 researchers
- 100 software development projects annually.
- Projects used by NTT group (274,000 employees)

case study

➔ NTT laboratories

- 3000 researchers
- 100 software development projects annually.
- Projects used by NTT group (274,000 employees)
- Users can only rely on documentation

methodology

→ Collected artifacts for 3 months

methodology

→ Collected artifacts for 3 months

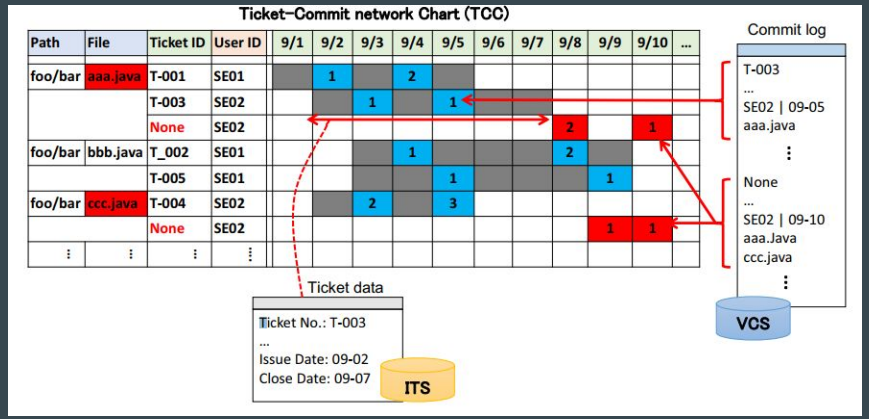
159 VCS commit logs

102 ITS tickets

methodology

➔ Collected artifacts for 3 months

159 VCS commit logs ⟷ **102 ITS tickets**



project overview

➔ **Prototype graphical modeling tool**

project overview

➔ Prototype graphical modeling tool

➔ **Support system development engineers**

project overview

- ➔ Prototype graphical modeling tool
- ➔ Support system development engineers
- ➔ **Budget: $10 million**

project overview

Three phases:

- planning (3 weeks)
- iteration 1 (4 weeks)
- iteration 2 (4 weeks)

project overview

Three phases:

- planning (3 weeks)
- iteration 1 (4 weeks)
- iteration 2 (4 weeks)

Three roles:

- software user (4)
- requirements engineer (2)
- software engineer (2)

project overview

Planning phase:

➔ **User stories**

project overview

Planning phase:

- ➔ User stories
- ➔ **Algorithms**

project overview

Planning phase:

- ➔ User stories
- ➔ Algorithms
- ➔ **Implementation tasks**

project overview

Iteration phase:

➔ **Each ITS ticket reflects one task**

project overview

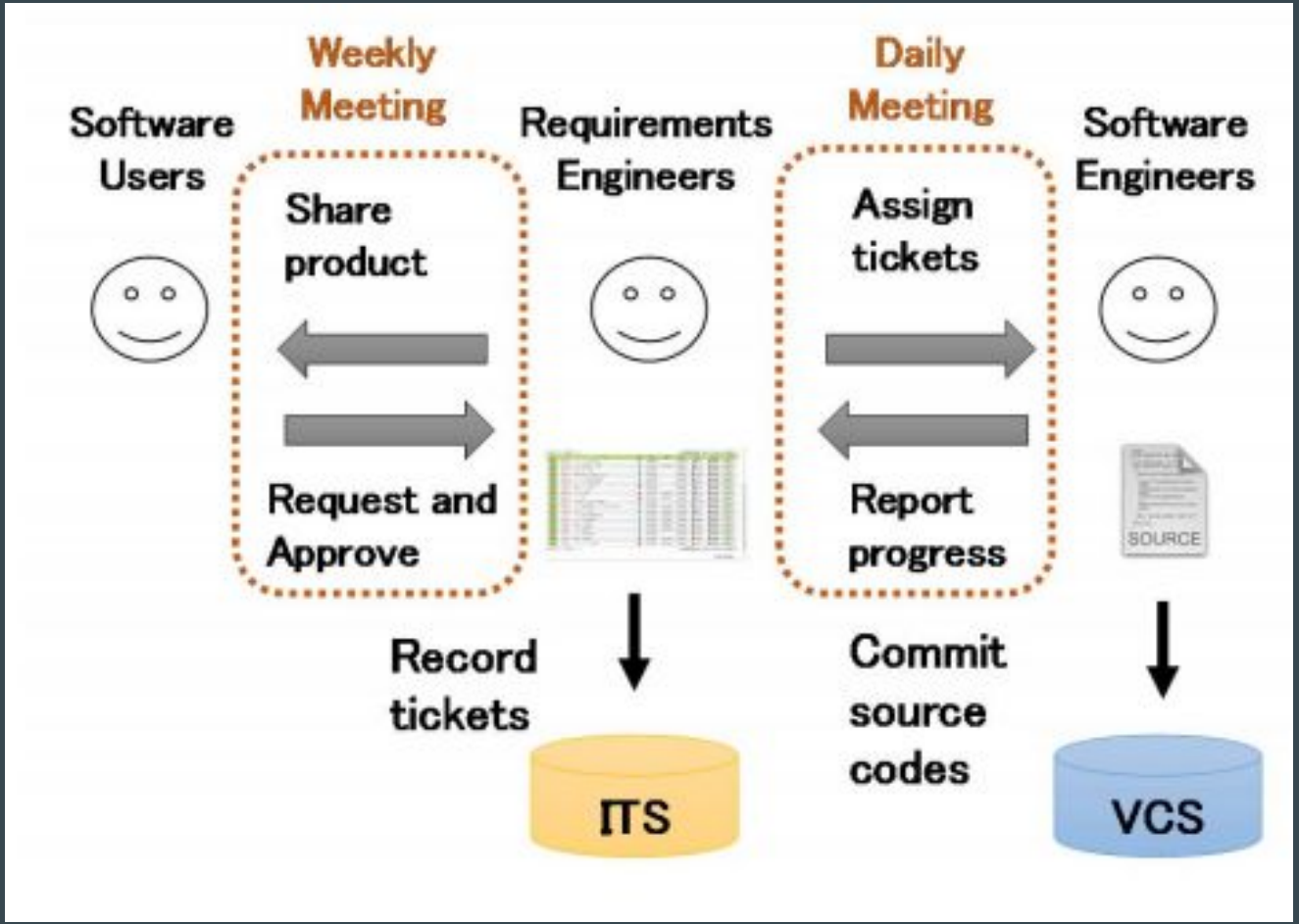
Iteration phase:

- ➔ Each ITS ticket reflects one task
- ➔ **Daily and weekly meetings**

project overview

Iteration phase:

- ➔ Each ITS ticket reflects one task
- ➔ Daily and weekly meetings
- ➔ **Commits must reference ticket**


Weekly Meeting
Daily Meeting
Software Users
Requirements Engineers
Software Engineers
Share product
Request and Approve
Assign tickets
Report progress
SOURCE
Record tickets
Commit source codes
ITS
VCS

dev requirements

→ **Must enter ticket ID in commit**

dev requirements

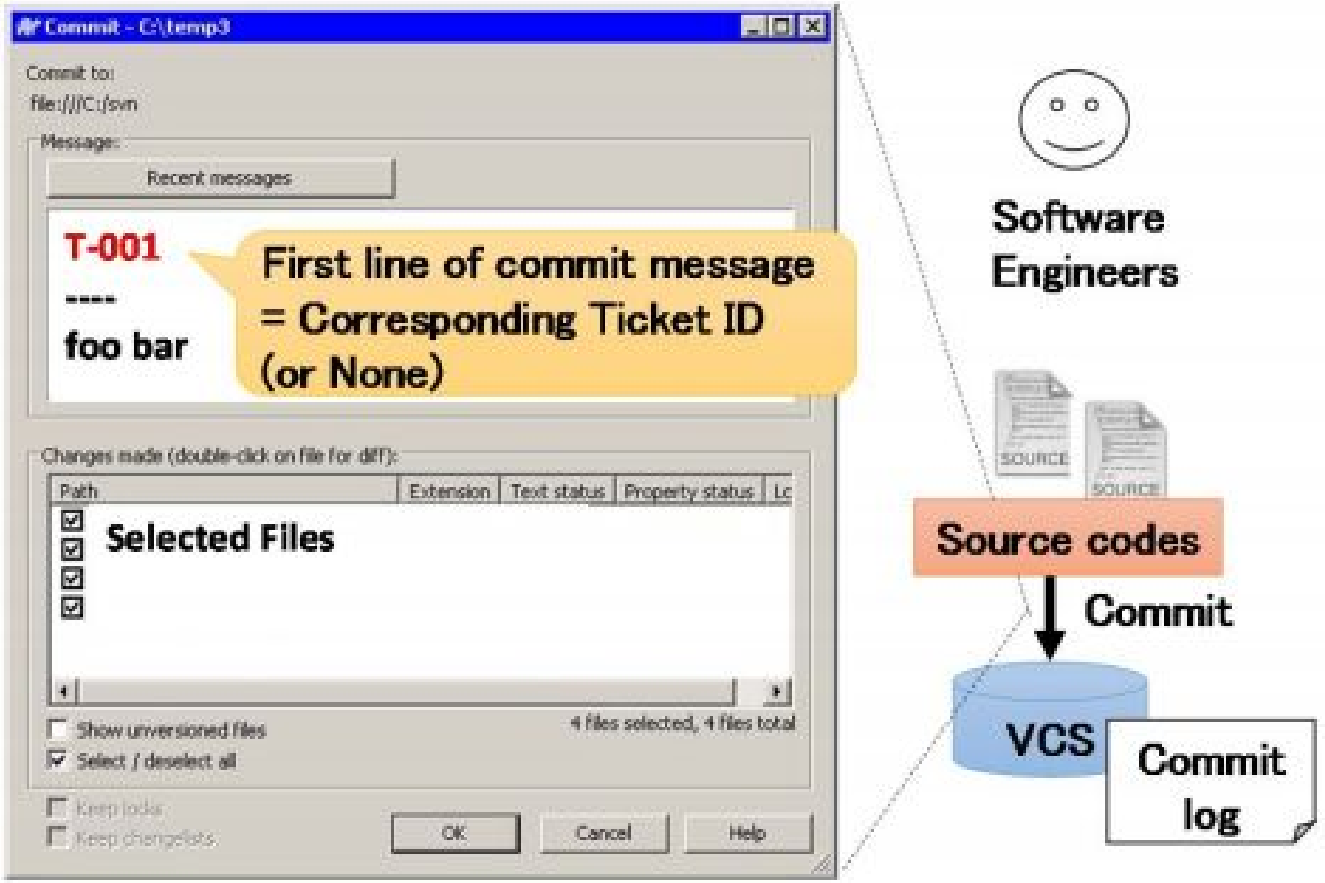
- ➔ Must enter ticket ID in commit

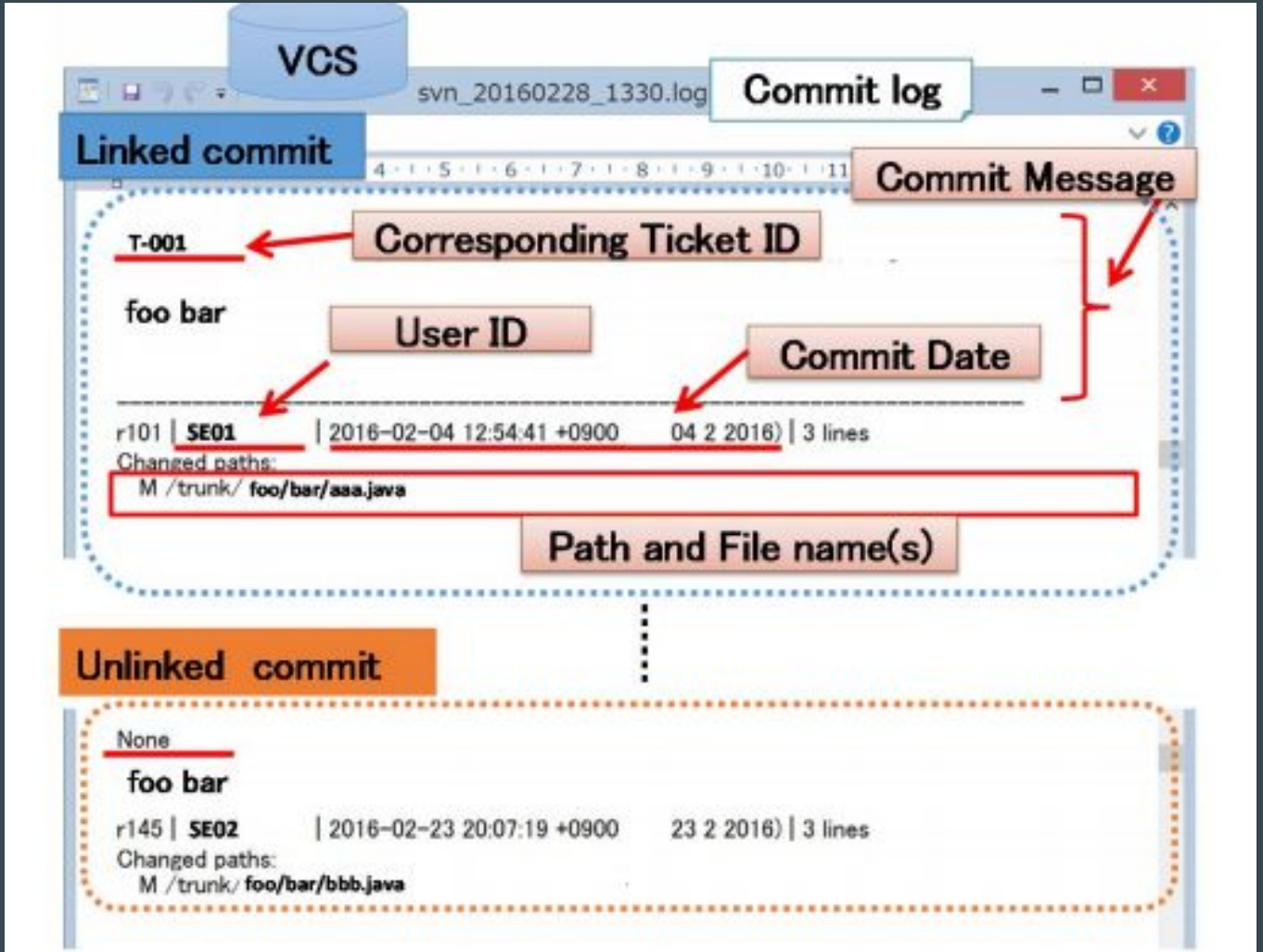
- ➔ **Only one ticket ID**

dev requirements

- ➔ Must enter ticket ID in commit
- ➔ Only one ticket ID
- ➔ **No ticket?**

Screen of VMS




VCS
svn_20160228_1330.log
Commit log
Linked commit
Commit Message
T-001
Corresponding Ticket ID
foo bar
User ID
Commit Date
r101 | SE01 | 2016-02-04 12:54:41 +0900 04 2 2016) | 3 lines
Changed paths:
M /trunk/ foo/bar/aaa.java
Path and File name(s)
Unlinked commit
None
foo bar
r145 | SE02 | 2016-02-23 20:07:19 +0900 23 2 2016) | 3 lines
Changed paths:
M /trunk/ foo/bar/bbb.java

unlinked commits

1. **What source code was committed but not linked?**

unlinked commits

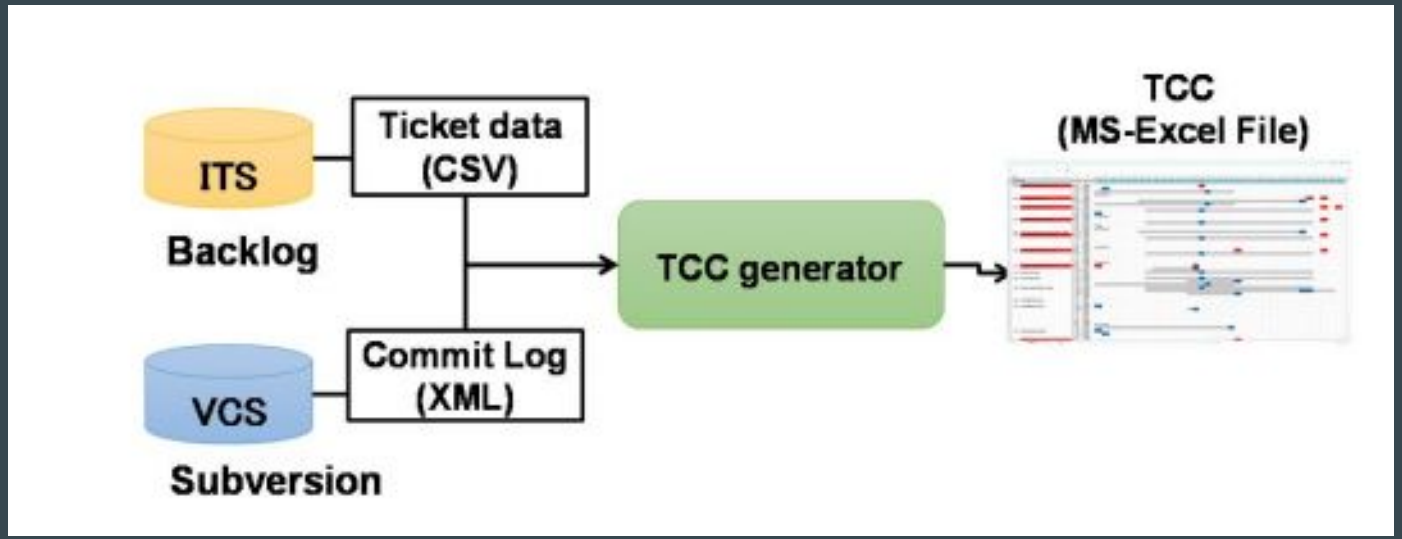
1. What source code was committed but not linked?

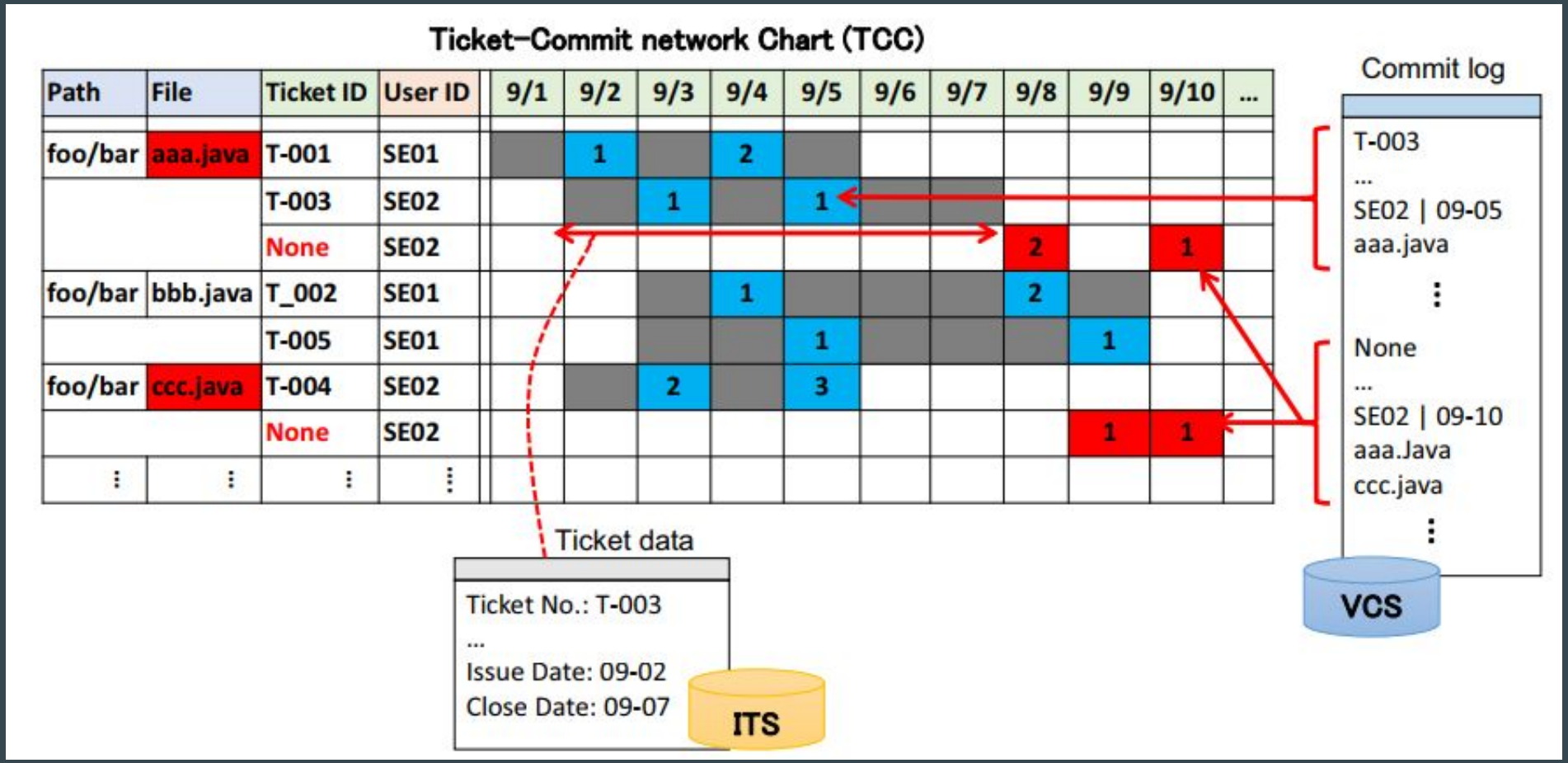
2. **When did unlinked commits occur?**

unlinked commits

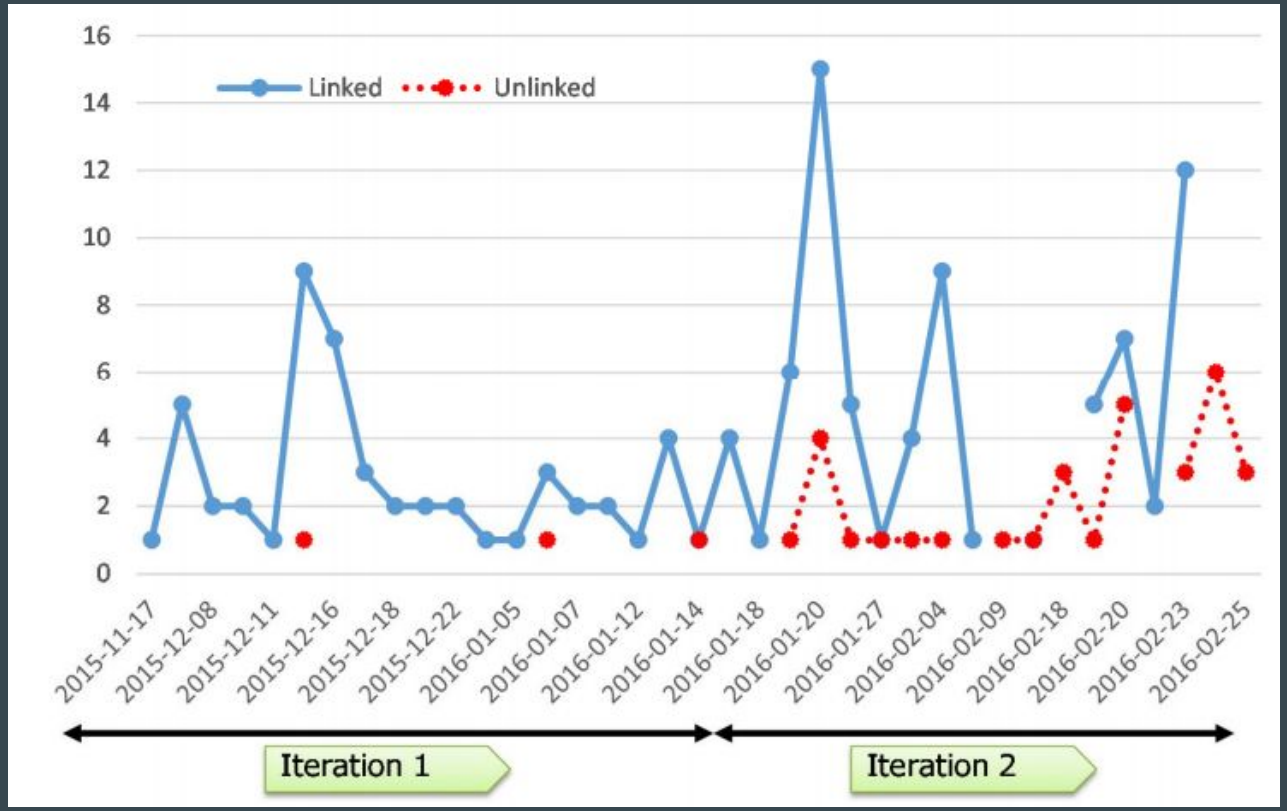
1. What source code was committed but not linked?
2. When did unlinked commits occur?
3. **How many times did unlinked commits occur?**

Ticket-Commit network Chart (TCC)




Ticket-Commit network Chart (TCC)
Path
File
Ticket ID
User ID
9/1
9/2
9/3
9/4
9/5
9/6
9/7
9/8
9/9
9/10
...
foo/bar
aaa.java
T-001
SE01
T-003
SE02
None
SE02
foo/bar
bbb.java
T_002
SE01
T-005
SE01
foo/bar
ccc.java
T-004
SE02
None
SE02
Commit log
T-003
...
SE02 | 09-05
aaa.java
None
...
SE02 | 09-10
aaa.Java
ccc.java
VCS
Ticket data
Ticket No.: T-003
...
Issue Date: 09-02
Close Date: 09-07
ITS

Number of commits/day



research questions

1. How many unissued tickets?

research questions

1. How many unissued tickets?

2. How much undocumented knowledge is **required** for future operation or modification?

results

Type No.	Types of task descriptions	No. of Not linked Commits	No. of unissued tickets	No. of tickets necessary for software modification	No. of tickets necessary for software operation	Examples of subjects described in recovered tickets
(1)	External function changes	6	4	1 [25% (=1/4)]	4 [100% (=4/4)]	Add function for displaying calculation result
(2)	Business logic changes	2	2	2 [100% (=2/2)]	2 [100% (=2/2)]	Modify calculation algorithms
(3)	User interface (system screen) changes	9	5	1 [20% (=1/5)]	5 [100% (=5/5)]	Change layout of input forms in system screens
4)	System property changes	4	3	2 [66% (=2/3)]	0 [0% (=0/3)]	Add/remove parameters in initial file
5)	Bug fixing	2	0	-	-	-
6)	Source code refactoring	3	3	0 [0% (=0/3)]	0 [0% (=0/3)]	Create utility class for aggregating common methods
7)	Development environment change	9	9	0 [0% (=0/9)]	0 [0% (=0/9)]	Rename brunch and tag. Eliminate unreachable code (dead code).
	Total	35	26 [20% (=26/(102+25)]	6 [23% (=6/26)]	11 [42% (=11/26)]	

35 commits without a ticket

results

Type No.	Types of task descriptions	No. of Not linked Commits	No. of unissued tickets	No. of tickets necessary for software modification	No. of tickets necessary for software operation	Examples of subjects described in recovered tickets
(1)	External function changes	6	4	1 [25% (=1/4)]	4 [100% (=4/4)]	Add function for displaying calculation result
(2)	Business logic changes	2	2	2 [100% (=2/2)]	2 [100% (=2/2)]	Modify calculation algorithms
(3)	User interface (system screen) changes	9	5	1 [20% (=1/5)]	5 [100% (=5/5)]	Change layout of input forms in system screens
4)	System property changes	4	3	2 [66% (=2/3)]	0 [0% (=0/3)]	Add/remove parameters in initial file
5)	Bug fixing	2	0	-	-	-
6)	Source code refactoring	3	3	0 [0% (=0/3)]	0 [0% (=0/3)]	Create utility class for aggregating common methods
7)	Development environment change	9	9	0 [0% (=0/9)]	0 [0% (=0/9)]	Rename brunch and tag. Eliminate unreachable code (dead code).
	Total	35	26 [20% (=26/(102+25)]	6 [23% (=6/26)]	11 [42% (=11/26)]	

20% of tickets were unissued

results

Type No.	Types of task descriptions	No. of Not linked Commits	No. of unissued tickets	No. of tickets necessary for software modification	No. of tickets necessary for software operation	Examples of subjects described in recovered tickets
(1)	External function changes	6	4	1 [25% (=1/4)]	4 [100% (=4/4)]	Add function for displaying calculation result
(2)	Business logic changes	2	2	2 [100% (=2/2)]	2 [100% (=2/2)]	Modify calculation algorithms
(3)	User interface (system screen) changes	9	5	1 [20% (=1/5)]	5 [100% (=5/5)]	Change layout of input forms in system screens
4)	System property changes	4	3	2 [66% (=2/3)]	0 [0% (=0/3)]	Add/remove parameters in initial file
5)	Bug fixing	2	0	-	-	-
6)	Source code refactoring	3	3	0 [0% (=0/3)]	0 [0% (=0/3)]	Create utility class for aggregating common methods
7)	Development environment change	9	9	0 [0% (=0/9)]	0 [0% (=0/9)]	Rename brunch and tag. Eliminate unreachable code (dead code).
	Total	35	26 [20% (=26/(102+25)]	6 [23% (=6/26)]	11 [42% (=11/26)]	

23% needed for “future game”

results

Type No.	Types of task descriptions	No. of Not linked Commits	No. of unissued tickets	No. of tickets necessary for software modification	No. of tickets necessary for software operation	Examples of subjects described in recovered tickets
(1)	External function changes	6	4	1 [25% (=1/4)]	4 [100% (=4/4)]	Add function for displaying calculation result
(2)	Business logic changes	2	2	2 [100% (=2/2)]	2 [100% (=2/2)]	Modify calculation algorithms
(3)	User interface (system screen) changes	9	5	1 [20% (=1/5)]	5 [100% (=5/5)]	Change layout of input forms in system screens
4)	System property changes	4	3	2 [66% (=2/3)]	0 [0% (=0/3)]	Add/remove parameters in initial file
5)	Bug fixing	2	0	-	-	-
6)	Source code refactoring	3	3	0 [0% (=0/3)]	0 [0% (=0/3)]	Create utility class for aggregating common methods
7)	Development environment change	9	9	0 [0% (=0/9)]	0 [0% (=0/9)]	Rename brunch and tag. Eliminate unreachable code (dead code).
	Total	35	26 [20% (=26/(102+25)]	6 [23% (=6/26)]	11 [42% (=11/26)]	

42% needed for “current game”

discussion

➔ **Two reasons for unissued tickets**

discussion

- ➔ Two reasons for unissued tickets
- ➔ **Undocumented knowledge is very high**

discussion

- ➔ Two reasons for unissued tickets
- ➔ Undocumented knowledge is very high
- ➔ **Required for maintenance and operation**

solution?

➔ Issue every ticket

solution?

➔ Issue every ticket

solution?

- ➔ Issue ~~every~~ ticket
  - Documentation becomes a burden

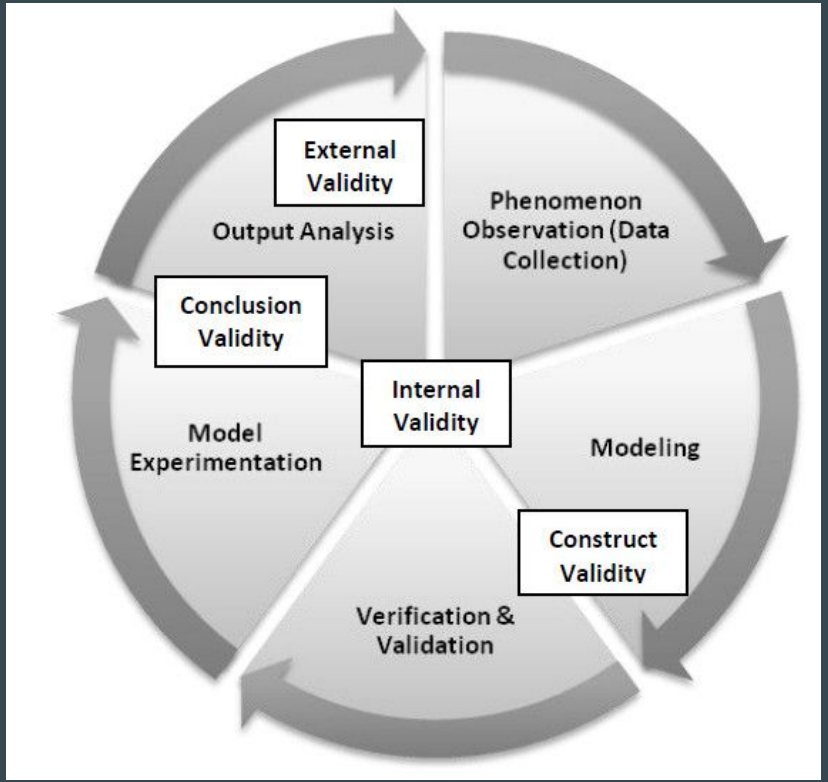
solution?

- ➔ Issue every ticket
  - Documentation becomes a burden
  - Agile suffers

solution!

- Issue every ticket
  - Documentation becomes a burden
  - Agile suffers
- TCC in review/retrospective meetings

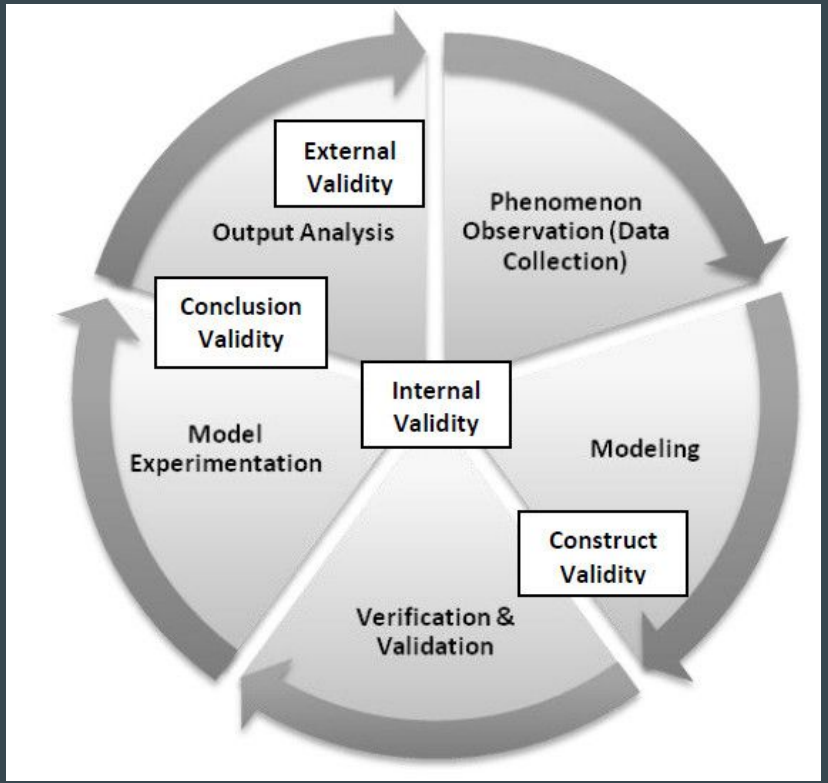
limitations



limitations

- Construct

Single data source/project



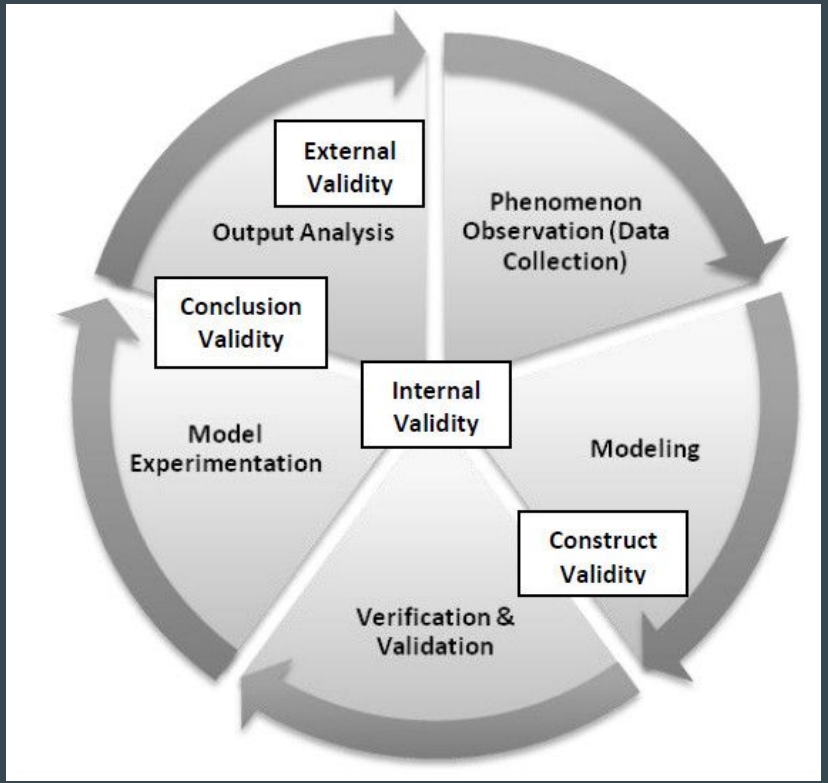
limitations

- Construct

Single data source/project

- External

Single study



limitations

- Construct

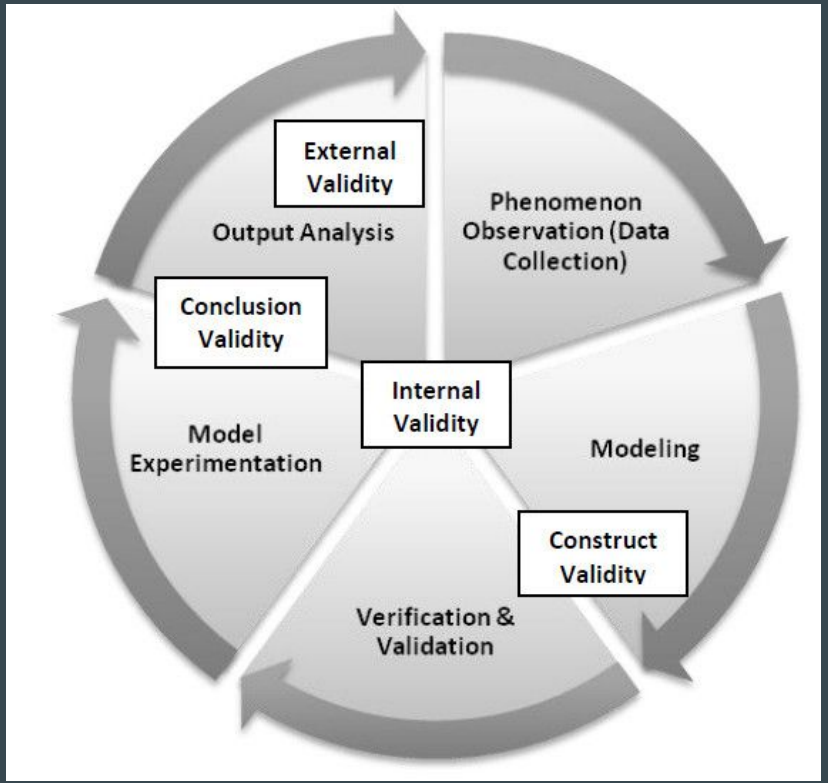
Single data source/project

- External

Single study

- Internal

No causal inferences



summary

➔ Case study on undocumented knowledge

summary

➔ Case study on undocumented knowledge

➔ Visualization tool to locate knowledge gaps

summary

- ➔ Case study on undocumentation
- ➔ Visualization tool to locate knowledge gaps
- ➔ Unissued tickets contain required knowledge

Questions?

Thank you

mitigating documentation

→ Implemented features vs. initial requirements

mitigating documentation

➔ Implemented features vs. initial requirements

➔ **Scrum solution: sprint review/retrospectives**

mitigating documentation

- ➔ Implemented features vs. initial requirements
- ➔ Scrum solution: sprint review/retrospectives
- ➔ **Formal vs. informal**