The Social Context of Software Maintenance

Jonathan Sillito and Eleanor Wynn
<table>
<thead>
<tr>
<th>Group</th>
<th>Manager(s)</th>
<th>Member(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A1</td>
<td>A2, A3, A4, A5</td>
</tr>
<tr>
<td>B</td>
<td>B1</td>
<td>B2, B3, B4, B5</td>
</tr>
<tr>
<td>C</td>
<td>C1</td>
<td>C2</td>
</tr>
<tr>
<td>D</td>
<td>D1</td>
<td>D3, D4, D5</td>
</tr>
<tr>
<td>E</td>
<td>E1, E2</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>F1</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>G1, G2</td>
<td>G3</td>
</tr>
<tr>
<td>H</td>
<td>H1</td>
<td>H2, H3, H4, H5</td>
</tr>
<tr>
<td>I</td>
<td>I1</td>
<td>I2, I3, I4, I5, I6, I7, I8</td>
</tr>
</tbody>
</table>
1. Decomposing work and code
   Work is embedded and organized around modules, layers, expertise, roles and branches.

2. Social dependencies
   Coordination and negotiation effort can dominate other planning and development activities.

3. Branches and redundancy
   Branches are used to manage the work of groups though this complicates maintenance activities.

4. Individually developed code
   Large body of code outside of normal consideration. Can lead to significant inconsistencies and redundancies.

5. Cross-group changes
   Such changes are difficult to make and often avoided (for social and organizational reasons).

6. Change processes
   Formal change processes can require more effort than the actual changes to the code.

7. Identifying dependencies
   Awareness of/access to code falls along org. lines. Identifying and working through dependencies is done in a social context.

8. Confidence in testing
   SE's do not have complete confidence in tests developed for systems. Crossing boundaries is difficult.

9. Crossing time-zones
   Working across widely separated time-zones affects many aspects of maintenance and shapes code and project structure.
1. Decomposing work and code

Members of group H actively work to maintain a focus on their area of expertise which involves connecting systems and transforming “data from one format to another” [H2].

All participating groups maintained code that was embedded in a larger system. Work was organized around source code modules, architectural layers, expertise, development roles and configuration management branches.
2. Social dependencies

Negotiating this documentation at the level of detail prescribed is “probably the most time during the design phase, that arguing back and forth” [H2].

Group F was “pretty isolated” [F1] from related groups.

“The testing team doesn’t always have the full picture” [G2].

“Architects are completely disconnected from the reality of the implementations and support” [A2].

Source code dependencies are at times described in terms of dependencies between people and groups. Coordination and negotiation effort can dominate other activities while planning and developing software.
3. Branches and redundancy

Often members of group E “replicate the same work by separate people because they are different groups” [E1]. “I am sure that it’s probably not going to work the first time around” [E1].

*Multiple configuration management branches of a code base are maintained to manage the work of different groups though this complicates maintenance activities.*
4. Individually developed code

“I work on small projects” [A3].

Solutions implemented by another group “won’t work here without at least some work” and so “we waste time solving a problem already solved elsewhere” [A5].

“People reinventing the wheel at each cycle” [D5].

Small, individually written programs and scripts can represent a large body of code outside of normal consideration. A local focus around such code can lead to significant inconsistencies and redundancies.
5. Cross-group changes

Most difficult changes are when “you have so many people depending on your code” [D4]. “Very difficult to know what changes are safe, and so there is a natural reluctance to make changes” [D4].

Find solution that would have “the minimal impact on the source code” [I3].

“This could use some improving but everything is tied into this, so no one wants to mess with it” [H3].

Many issues around making changes are fundamentally social or organizational issues. High-level changes to a code base that is shared between groups, or other changes with possible effects across groups, are difficult to make and often avoided.
6. Change processes

“When we change a core component it usually takes a long time, even for small changes, and for the most part we just don’t change them” [H3].

They put lots of effort into upfront negotiation “to make them so that we are not expecting to make any changes until the next release” [H2].
7. Identifying dependencies

“When you integrate that module into the bigger system, something goes wrong... what tends to happen, organizationally, you have a person on [team A] who is just assigned to talk to [team B] and then on [team B] you have someone who is assigned to talk to [team A]. And they're constantly trying to tell each other how [to] translate, ‘well that's what this means in our world’” [D4].

Awareness of and access to code tends to fall along organizational lines. Identifying and working through dependency issues is often done in a social context.
8. Confidence in testing

“We have also noticed that developers sometimes write useless unit tests that really don't test what they were intended to test... simply because you see unit tests attached to code doesn't mean that the code is actually being sufficiently tested” [I3].

They “ran a whole bunch of [transactions] through to [third-party] and the business people in the background were talking to the folks at [third-party] who were checking that the right [data was] received into their system and things look right” [H3].

Software engineers do not have complete confidence in tests developed for systems and automatically testing some aspects of a system is infeasible. Testing is particularly challenging when crossing organizational boundaries.
9. Crossing time-zones

“Time-zone makes it hard to trouble-shoot and interact with customers” [A1].

“Working across time-zones in this way results in lots of misunderstandings and rehashing--you think you are in agreement but then later realize you are not” [G3].

Working tightly across time-zones causes “work-life balance problems” [G1].

Working across widely separated time-zones introduces issues that affect many aspects of maintenance. We observed a tendency to minimize dependencies between time-zones, shaping both the code and project structure in the process.
A reluctance to make changes
Programmers are reluctant to make some kinds of changes. What are the long term consequences for the code base?

Communicating about source code
Communicating and coordinating are important and difficult (in particular between projects). More research is needed into support for communicating about source code.
1. Decomposing work and code
   Work is embedded and organized around modules, layers, expertise, roles and branches.

2. Social dependencies
   Coordination and negotiation effort can dominate other planning and development activities.

3. Branches and redundancy
   Branches are used to manage the work of groups though this complicates maintenance activities.

4. Individually developed code
   Large body of code outside of normal consideration. Can lead to significant inconsistencies and redundancies.

5. Cross-group changes
   Such changes are difficult to make and often avoided (for social and organizational reasons).

6. Change processes
   Formal change processes can require more effort than the actual changes to the code.

7. Identifying dependencies
   Awareness of/access to code falls along org. lines. Identifying and working through dependencies is done in a social context.

8. Confidence in testing
   SE’s do not have complete confidence in tests developed for systems. Crossing boundaries is difficult.

9. Crossing time-zones
   Working across widely separated time-zones affects many aspects of maintenance and shapes code and project structure.