

Towards a Theory of Continuously Personalized Design in Mass Markets

Position Statement

Design Requirements Workshop

June 3 – 6, 2007

Case Western Reserve University

Cleveland, Ohio, USA

William N. Robinson

Computer Information Systems

Suite 929, 35 Broad St., NW

Georgia State University

wrobinson@gsu.edu

1 Design is unique

Software developers have struggled to meet diverse customer expectations. In the attempt to please everyone, developers please no one with their complex software. Most of us will use only a fraction of their functionality, if we can manage to navigate their complex interfaces.

Each design is a unique. The simplest bridge must be designed to fit its environment—the span, elevation, and abutment. Software should be unique. It is *soft*-ware, which presumably is more malleable than hardware. Software, however, is largely produced in factories, with the notation that one design fits all—well, at least, one design can be configured for all. To bring about an appearance of customization, mass-production techniques from product line engineering have been adapted to software design and development.

Each software design, or family of designs, uniquely brings functionality to the masses. With proper guidance, users can “skin” software interfaces. Consultants customize larger enterprise packages. Yet, the underlying design, and functionality remain constant. Consumers must await the next round software updates to obtain the newest version of the mass-produced software. Although software design

can be unique, it is a commodity to the user—some have even suggested that software is no longer a strategic differentiator for organizations.

2 Software is personal

Software can be unique. It can be personalized for a single individual. For some, software must be personalized. More than one million adults in the U.S. are diagnosed each year with cognitive impairments (CI) due to neurological disease or trauma (e.g., traumatic brain injury, stroke, tumor, epilepsy, infectious disease). Cognitively impaired patients often experience social isolation and its ancillary effects. Naturally, one might expect that some form of computer assistive technology (AT) would aid CI patients—for example, in communicating via email. Sadly, AT has not been shown to assist CI patients. One study, however, suggests that personalized software may help[1, 2].

The Think and Link (TAL) project provides a specialized emailing system to CI patients[1-5]. In contrast to other AT studies, TAL patients continue to expand their email system usage. This unusual success is attributed, in part, to the personalized software. By continually monitoring clinical and user goals, TAL can responsively provide personalized software adaptations. With usage, the software used by each TAL CI patient is unique in its interface and functionality.

The approximately 53 million disabled Americans are only one group that may benefit from software personalization. Many of us have used Microsoft Word, Adobe Photoshop, or similarly complex software, only to wish for a reduced, simplified version that did just what we wanted, and no more. Microsoft introduced adaptable menus with this aim in mind. Instead of helping, the constant menu variation became another user burden. Consequently, most users disabled the adaptive menus. This technology is absent in the most recent version of Microsoft Office (2007).

Personalized software is still desired by users and organizations, as evidenced by the web pages, books, and consultants that provide customization guidance. A recent article questions the relevance of IT for strategic advantage[6], in part because software is a commodity. It may be that “IT doesn’t matter”, in part, because much of IT is not unique. Without some unique aspect, IT is not an intellectual property or strategic differentiator, and therefore, it is simply an infrastructure cost, like the manufacturing plant or electricity. In contrast to commodity software, personalized software is unique, and therefore it can be intellectual property and a strategic differentiator.

3 Leveraging continuous requirements analysis

Requirements analysis contributes to projects success, according to many studies[7-12]. Requirements representing the needs of the system stakeholders support developers in their efforts to develop and enhance software satisfying stakeholder needs. Bad or no requirements analysis contributes to project failure, despite the best intentions of the project team.

Customization is increasingly common in product and service development. Organizations seek to get “closer to the customer” to understand their needs, thereby enabling valued customizations, increased customer satisfaction and increased market share. Designers, from the old-economy automobiles to the new-economy digital products seek to reduce product cycle time. All products and services reside within a product family. New variants are released in response to new knowledge acquired through customer

feedback and analysis. Designers seek to quickly, better their products and services by reducing the time between releases by obtaining good, timely feedback.

Software enables fast feedback. Designers of physical products, like automobiles, must await customer survey cards, focus group and sales analysis. In contrast, software can monitor the user-computer interactions and provide usage information directly to the designers. Consequently, feedback may be instantaneous, and subsequent redesign can follow immediately. Moreover, individuals may be monitored, thereby supporting individual software customization. Software supports customization not only because it can be modified quickly, but particularly because design feedback may be obtained quickly. Software allow for post-deployment requirements re-analysis and evolution, unlike like physical products.

4 Mass-personalized software development

To address the demand for personalized software, we propose new research on *mass-personalized software development*. MPSD is the large-scale production of software products that are continually customized to individual customer needs. Unlike mass customization approaches, which address physical product production, MPSD uniquely addresses product customization after a customer begins using a digital (software) product. The MPSD approach contrasts from other customization approaches in that individual needs dominate over product commonality; moreover, the changing user needs are met with continuous product adaptations.

MPSD manages three important models in support of post-purchase personalization:

1. *Software product line model*: A software product line engineering approach is applied, where commonality and variability models help manage the production of product variants[13, 14].
2. *User environmental model*: A model of each user's environment, including their product variant history, is maintained.
3. *User model*: A model of each user's needs, skills, and achievements is maintained. The user model guides the product adaptations that improve satisfaction of partially satisfied user needs.

Customization is the process of managing product adaptations, which occur in response to monitored changes in the user's model. A new product variant is selected from the product line and is sent to the user when the user model warrants a product update.

When a user attains a higher skill-level, such that she may better utilize another product variant to satisfy unachieved goals, then another product variant is deployed. As an illustration, consider a computer video game in which certain game scenarios become available only after the user has demonstrated required gaming skills. This just-in-time software feature approach avoids the bloated and confusing menus found in much commercial software. It supports Carroll's minimalist learning theory, which proposes that users only be exposed to the minimal, necessary services that enable them to achieve their goals[15]. With MPSD, features become available just-in-time, when the user has the appropriate skills and goals.

MPSD is not the only theory of product customization, of course. In fact, MPSD is the refinement of clinical requirements engineering and related theories of mass customization. Those theories are briefly summarized next.

4.1 Supporting Theories

Five supporting MPSD theories are summarized in Table 1. Although the theories have much in common, the table highlights their differences. Business strategies for customized physical products are the focus of mass customization. *Collective customer commitment* (C3) refines mass customization with co-design and customer commitment; CRE can be considered a refinement of C3 focused on uniquely personal software[16-18]. *Software product line engineering* (SPLE)[13, 14] applies the general mass-customization principles to software development. Evolutionary software development (ESD) demonstrates how software evolution can fit within a customization approach [19-26]. Finally, MPSD widens the applicability of *clinical requirements engineering* (CRE) by incorporating SPLE and automation[5].

Table 1 A Summary contributions of customization approaches.

Customization theory	Major Contribution
Mass customization	Business strategies and tactics for the mass production of timely, low cost, customized physical products.
C3	Business strategies and tactics that incorporate co-design and customer commitment within mass customization.
SPLE	Software development methodology supporting the mass customization of software products.
ESD	Software development principles for the continual evolution of software products.
CRE	Software development methodology supporting the continual personalization of software products.
MPSD	Software development methodology supporting the continual personalization of software products for mass markets.

MPSD contributes to the more established theories by addressing (1) uniquely personal requirements and (2) their continual satisfaction through monitoring and adaptation. If MPSD is to be successful, then monitoring and adaptation must be scaled to mass populations. Monitoring is considered a prerequisite to adaptation because it guides adaptation and because it is a manually burdensome activity, in part because of the large mass of data that must be analyzed. Thus, monitoring contributes by bridging the automation gap between MPSD theory and MPSD practice—without automation, MPSD will not scale, and thus it cannot fulfill its mass personalization mission.

4.2 Mass-Personalized Challenges

The future of mass-personalized software development depends on successful results in three challenging research areas:

1. *Requirements specification and evolution*: Precise requirements specification continues to be a challenge. The evolutionary nature of MPSD depends on good requirements modeling over time. This expands the challenges of SPLE to address requirements transformations and effects analysis.
2. *Requirements monitoring and acquisition*: MPSD depends on continuous monitoring of user satisfaction and requirements re-analysis. Preliminary research on real-time user-model analysis must extend the variety of user-computer properties addressed, efficiently[27]. Such results must feedback into requirements evolution and onto software evolution.
3. *Software evolution*: MPSD depends on effective software evolution techniques. Research on software derivation, configuration, and evolution must become practical. Rather than generate whole programs from specification, component-based selection and integration techniques show promise.

Together, these three areas address what has been termed the Requirements Interaction Management life cycle[28]. From initial acquisition to through run-time system monitoring and adaptation, requirements interaction—including user-system interactions—must be continually analyzed, resolved, and updated.

5 References

1. Todis, B., et al., *Making electronic mail accessible: Perspectives of people with acquired cognitive impairments, caregivers and professionals*. Brain Injury, 2005. **19**(6): p. 389-402.
2. Sohlberg, M.M., et al., *The longitudinal effects of accessible email for individuals with severe cognitive impairments*. Aphasiology, 2005. **19**(7): p. 651-681.
3. Sutcliffe, A., S. Fickas, and M.M. Sohlberg, *Personal and Contextual Requirements Engineering*. Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on, 2005: p. 19-30.
4. Fickas, S., W. Robinson, and M. Sohlberg. *The Role of Deferred Requirements: A Case Study*. in *International Conference on Requirements Engineering (RE'05)*. 2005. Paris, France: IEEE.
5. Fickas, S., *Clinical requirements engineering*. Proceedings of the 27th international conference on Software engineering, 2005: p. 28-34.
6. Nicholas, G.C., *IT doesn't matter*. Harvard Business Review, 2003. **81**(5): p. 41.
7. Boehm, B., *Software engineering economics*. 1981, Englewood Cliffs, N.J.: Prentice-Hall.
8. Keil, M., et al., *A framework for identifying software project risks*. 1998, ACM Press New York, NY, USA. p. 76-83.
9. Lyytinen, K., Hirschheim, R., *Information systems failures-a survey and classification of the empirical literature*. Oxford Surveys in Information Technology, 1987. **Vol. 4**: p. pp. 257-309.
10. Neumann, P.G., *Computer Related Risks*. 1995: Addison-Wesley.
11. Hofmann, H.F., F. Lehner, and G. Motors, *Requirements engineering as a success factor in software projects*. 2001. p. 58-66.
12. Munro, M.C. and B.R. Wheeler, *Planning, Critical Success Factors, and Management's Information Requirements*. 1980, JSTOR. p. 27-38.
13. Pohl, K., G. Böckle, and F.J.v.d. Linden, *Software Product Line Engineering : Foundations, Principles and Techniques* 2005, Berlin Heidelberg: Springer.

14. Clements, P. and L. Northrop, *Software Product Lines: Practices and Patterns*. 2002, Boston, MA: Addison-Wesley.
15. Carroll, J.M., *The Nurnberg funnel: designing minimalist instruction for practical computer skill*. 1990, Cambridge MA: MIT Press.
16. Piller, F., et al., *Overcoming mass confusion: Collaborative customer co-design in online communities*. Journal of Computer-Mediated Communication, 2005. **10**(4): p. article 8.
17. Elofson, G.S. and W.N. Robinson, *Creating a Custom Mass-Production Channel on the Internet*. Communications of the ACM, 1998. **41**(3): p. 56-62.
18. Ogawa, S. and F. Piller, *Collective Customer Commitment: Reducing the risks of new product development*. MIT Sloan Management Review, 2006. **47**(2): p. 65-72.
19. Stidolph, W., *Evolutionary design of complex software (EDCS) demonstration days 1999*. ACM SIGSOFT Software Engineering Notes, 2000. **25**(1): p. 95-108.
20. Tracz, W., *Evolutionary Design of Complex Software (EDCS) Kick Off Workshop Summary*. ACM SIGSOFT Software Engineering Notes, 1996. **21**(5): p. 40-42.
21. Punter, T., A. Trendowicz, and P. Kaiser, *Evaluating Evolutionary Software Systems*. 4th International Conference on Product Focused Software Process Improvement PROFES, 2002.
22. Lehman, M.M., *Software Evolution*. Encyclopedia of Software Engineering, 2002. **2**: p. 1507-1513.
23. Lehman, M.M. and J.F. Ramil, *Rules and Tools for Software Evolution Planning and Management*. Annals of Software Engineering, 2001. **11**(1): p. 15-44.
24. Lehman, M.M., et al., *Metrics and Laws of Software Evolution-The Nineties View*. Proc. Metrics, 1997. **97**: p. 5-7.
25. Lehman, M.M., *Laws of Software Evolution Revisited*. European Workshop on Software Process Technology, 1996: p. 108-124.
26. Lehman, M.M., D.E. Perry, and W.M. Turski, *Why is it so hard to find Feedback Control in Software Processes*. Proc. of the 19th Australasian Computer Science Conference.
27. Robinson, W.N., *Requirements Monitoring for Enterprise Systems*. Requirements Engineering Journal, 2006. **11**(1): p. 17-41.
28. Robinson, W.N., S. Pawlowski, and V. Volkov, *Requirements Interaction Management*. ACM Computing Surveys (CSUR), 2003. **35**(2): p. 132 - 190.